INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Solução médica orientada a micro aplicações

DIOGO FILIPE MESQUITA MORAIS SOARES
Outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Solução médica orientada a micro aplicações

Diogo Filipe Mesquita Morais Soares

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

Orientador: Ricardo Almeida

Co-Orientador: Nuno Escudeiro

Supervisor: André Mota

Porto, 15 de outubro, 2022

Resumo

Este documento explicita o processo de conceptualização e desenvolvimento da versão *Minimum Valuable Product* (MVP) de um sistema *Computerized Physician Order Entry* (CPOE), para ser incluído na aplicação hospitalar Viewer, da Glintt-HS. Estes sistemas representam soluções agregadoras de diferentes fontes de informação, permitindo centralizar os pedidos médicos de diferentes áreas.

Dado que não seria concretizável incidir em várias áreas hospitalares para o âmbito desta dissertação, este CPOE foca-se unicamente na área de monitorização de sinais vitais. Esta é uma área em crescimento, onde se identificou e selecionou o projeto *Wireless biOmonitoring stickers and smart bed architecture: toWards untethered patients*, para impulsionar esta versão MVP. Sinteticamente, o projeto envolve a utilização de dispositivos de monitorização sem fios e aderentes à pele para realizar monitorizações.

Assim, esta dissertação responde à hipótese de conceptualização de um CPOE que permita a gestão transversal à área de monitorização de sinais vitais, gerindo corretamente todos os pedidos do profissional de saúde. Esta hipótese acrescenta valor ao estado da arte, inovando e oferecendo uma abordagem diferenciadora, através da adição da área de monitorização de sinais vitais a um sistema CPOE, que habitualmente inclui apenas as áreas de imagiologia, laboratório e farmácia.

Para resolver este problema foi selecionada a metodologia *Design Science Research Methodology* (DSRM), de forma o obter os resultados esperados, ou seja, um software que permita associar os dispositivos a pacientes, criar pedidos de monitorização com recorrência temporal e consultar os resultados das monitorizações efetuadas. Tudo isto, com um alto grau de qualidade, usabilidade e suportabilidade.

Desta forma, é efetuado o levantamento do estado da arte, com vista a expandir o conhecimento da área de negócio e perceber o que o mercado oferece. É feito ainda o levantamento de requisitos e documentado todo o processo de conceptualização da solução. Finalmente, é evidenciado o seu desenvolvimento, sendo posteriormente avaliado.

No final conclui-se que a solução atingiu as expetativas, cumprindo a hipótese, os objetivos e os requisitos definidos.

Palavras-chave: *Computerized Provided Order Entry*, Monitorização, Sinais Vitais, *Wireless biOmonitoring stickers and smart bed architecture: toWards untethered patients*

Abstract

This document explains the conceptualization and development process of the *Minimum Valuable Product* (MVP) version of a *Computerized Physician Order Entry* (CPOE) system, to be included in *Viewer* hospital application, of *Glintt-HS*. These systems represent solutions that aggregate different sources of information, allowing the centralization of medical orders from different areas.

Since it would not be feasible to focus on several hospital areas for the scope of this dissertation, this CPOE focuses solely on the vital signs monitoring area. This is a growing area, where the project *Wireless biOmonitoring stickers and smart bed architecture: toWards untethered patients* was identified and selected to drive this MVP version. Briefly, the project involves the use of wireless, skin-tethered monitoring devices to perform monitoring.

Thus, this dissertation answers the hypothesis of conceptualizing a CPOE that allows the transversal management of the vital signs monitoring area, correctly managing all requests from the healthcare professional. This hypothesis adds value to the state of the art, innovating and offering a differentiating approach by adding the vital signs monitoring area to a CPOE system, which usually only includes the medical imaging, laboratory, and pharmacy areas.

To solve this problem, the Design Science Research Methodology (DSRM) was selected, to obtain the expected results, i.e., a software that allows the association of devices to patients, the creation of monitoring requests with temporal recurrence and consultation of monitoring results. All this, with a high degree of quality, usability, and supportability.

Thus, the state of the art is studied to expand the knowledge of the business area and understand what the market offers. The requirements are specified and the whole process of conceptualization of the solution is documented. Finally, the development is evidenced and then evaluated.

In the end it was concluded that the solution met expectations, fulfilling the hypothesis, objectives and requirements defined.

Keywords: *Computerized Provided Order Entry*, Monitoring, Vital Signs, *Wireless biOmonitoring stickers and smart bed architecture: toWards untethered patients*

Agradecimentos

Agradeço à minha família, especialmente aos meus pais por todo o apoio que têm proporcionado ao longo destes anos.

À minha namorada, Susana, por ter estado ao meu lado em todos os momentos.

A todos os meus grandes amigos que fiz antes e durante o meu percurso académico.

Aos colegas da Glintt que me deram um enorme apoio na realização deste projeto, confiando-me essa responsabilidade.

Por último, agradeço também ao meu orientador Ricardo Almeida, coorientador Nuno Escudeiro e supervisor André Mota por toda disponibilidade e ajuda, fundamental na realização deste trabalho.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos

Lista de Acrónimos

APIs	*Interfaces de Programação de Aplicação*
CHUC	*Centro Hospitalar e Universitário de Coimbra*
CPOE	*Computerized Physician Order Entry*
CRUD	*Create Read Update Delete*
DSRM	*Design Science Research Methodology*
EHR	*Electronic Health Record*
FHIR	*Fast Healthcare Interoperability Resources*
MVP	*Minimum Viable Product*
QEF	*Quantitative Evaluation Framework*
UML	*Linguagem de Modelação Unificada*
WoW	*Wireless biOmonitoring stickers and smart bed architecture: toWards untethered patients*

1 Introdução

Este capítulo apresenta o projeto elaborado no contexto da unidade curricular de Tese/Dissertação/Estágio (TMDEI) do Mestrado em Engenharia Informática (MEI) do Instituto Superior de Engenharia do Porto (ISEP).
Os tópicos abordados incluem a contextualização, a descrição do problema, os objetivos a alcançar, a abordagem preconizada e, finalmente, a estrutura do documento.

1.1 Contexto

A Glintt Healthcare Solutions (Glintt-HS), do grupo Global Intelligent Technologies (Glintt), é uma multinacional de tecnologia e consultoria com mais de 20 anos de experiência. Esta organização conta com mais de 1100 colaboradores que operam a partir de 10 escritórios em 6 países: Portugal, Espanha, Angola e Brasil, Reino Unido e Irlanda. Atualmente opera nos ramos de consultoria e serviços tecnológicos na Saúde, tendo soluções em cerca de 430 hospitais e 500 clínicas. No setor da Farmácia, mais de 14.000 farmácias na Península Ibérica utilizam o software de gestão suportado pela Glintt-HS, que disponibiliza ainda um portfólio mais vasto que engloba conceção e projeção de espaço de lojas, automação, infraestruturas, consumíveis, entre outros. [1]

Nesta organização, durante o ano de 2020, surgiu um novo projeto, o *Viewer*. Este projeto procurava implementar uma nova solução orientada a microaplicações, cujo propósito é o de suprir necessidades informáticas clínicas de uma infraestrutura hospitalar. Apesar da organização ter soluções semelhantes, o projeto procurou uma abordagem diferenciadora: adotar um melhor processo de desenvolvimento, tirar partido de tecnologias mais recentes e mais capazes, aprender com os erros de outras soluções e, consequentemente, entregar um produto que traga valor acrescentado ao cliente.

Assim, a visão do produto identifica que a solução deve permitir aos profissionais de saúde dos hospitais minimizar o tempo despendido no uso do software. A solução procura não só ter tempos de resposta mais rápidos, como também organizar a informação e os processos, de forma que os utilizadores demorem menos tempo a realizar as mesmas operações. Quanto menos tempo o médico ou o enfermeiro se ocuparem com o uso da aplicação, mais tempo vão dispor para os doentes, beneficiando todos os envolvidos.

[1] Retirado em fevereiro de 2022 de https://www.glintt.com/pt/o-que-somos/sobreaglintt/Paginas/default.aspx

As consultas médicas são um exemplo concreto onde o médico pode maximizar o tempo útil dedicado ao doente, através da otimização do tempo aplicado no uso do software. A solução permite ao médico focar-se em escutar e examinar o paciente, através de um software que lhe ocupará menos tempo.

Na ótica de maximizar o tempo do profissional de saúde, esta dissertação documenta o processo de conceptualização e implementação de uma versão *Minimum Viable Product* (MVP) de um *Computerized Physician Order Entry* (CPOE), a ser consumido pela aplicação *Viewer*. Este desenvolvimento representa um subproduto desta solução e irá incidir sobre a área de sinais vitais.

1.2 Problema

Dentro do contexto explicado na Secção 1.1, surgiu a necessidade de criar um processo para o profissional trabalhar o plano de tratamento de cada doente. Muitas soluções mostram-se pouco adequadas para dar a resposta mais útil para o profissional de saúde. Os vários pedidos de diferentes áreas, para o tratamento do doente estão muitas vezes dispersos pelas aplicações com que os profissionais de saúde trabalham. O tempo do utilizador não é otimizado devido a esta experiência múltipla a que é sujeito.

Assim, nasce a oportunidade de criar um sistema que permita aos profissionais de saúde ter uma experiência agregadora, unitária e ágil no plano de tratamento do doente.
Isto é possível através da agregação de criação de pedidos de prescrição de exames, laboratório e imagiologia numa única experiência, oferecendo ao utilizador uma visão centralizada para o tratamento do doente, concentrando no mesmo local os pedidos inseridos no seu plano de tratamento.
É exatamente isto que oferece um CPOE. No entanto, o problema prevê também que a experiência de tratamento do doente permita ter uma visão integrada e pluridisciplinar do plano de cuidados do doente, incluindo mais áreas funcionais.

Dado que o comum CPOE prevê apenas as áreas de exames, laboratório e imagiologia, é necessária a conceptualização e implementação de um CPOE com a evolução de ter responsabilidades em mais áreas funcionais. Esta dissertação pretende fazê-lo, com uma prova de conceito, através de um *Minimum Viable Product* (MVP) de um CPOE focado na área de sinais vitais.

Este trabalho será feito no contexto de um projeto - *Wireless biOmonitoring stickers and smart bed architecture: toWards untethered patients (WoW)* – que está a desenvolver uma iniciativa vanguardista nas áreas de saúde digital e hospitalização domiciliária, através da criação de dispositivos sem fios e aderentes à pele, com funções de medição de variados sinais vitais.

A solução deverá facilitar ao profissional de saúde o processo leituras e de requisição de pedidos no âmbito das necessidades da área de Sinais Vitais, através da aplicação *Viewer*.

1.3 Objetivos e contributos esperados

Como objetivo principal, identifica-se a conceptualização e desenvolvimento da solução MVP, do subproduto CPOE para a área de Monitorização Sinais Vitais.

A investigação sobre CPOE e arquiteturas que o sirvam, assim como a área de Sinais Vitais, serão fundamentais para averiguar o estado da arte e oferecer diferentes conceptualizações de propostas com potencial para solucionar o problema.

Os contributos esperados incluem o acréscimo de valor à solução *Viewer*, através do CPOE implementado, assim como valor acrescentado à comunidade científica através da conceptualização de um sistema CPOE diferenciador, dado que acrescenta desde logo responsabilidades de áreas extraordinárias a este sistema.

De forma a atingir o objetivo do projeto, foram identificadas as seguintes tarefas:

- Efetuar o levantamento de requisitos.
- Estudar CPOE e o seu estado da arte, sistematizando o conhecimento existente e diferentes soluções no mercado.
- Estudar a área de Sinais Vitais, a evolução tecnológica e medicinal nesta área e o seu estado da arte.
- Sistematizar o conhecimento existente de protocolos de comunicação de *healthcare*, o que inclui o domínio do standard de saúde *HL7 FHIR*.
- Sistematizar diferentes abordagens para a resolução do problema, através do estudo de diferentes arquiteturas e tecnologias. O foco estará em estudar alternativas confrontando os seus benefícios, desvantagens e limitações. Com estas diferentes abordagens será possível justificar a escolha da arquitetura.
- Desenhar a solução para o problema.
- Implementar a solução MVP para o problema.
- Avaliar a solução através de testes unitários, de integração, de aceitação e de carga, de forma a garantir que os requisitos são cumpridos.

1.4 Abordagem preconizada

A Glintt-HS, possui e continua a desenvolver soluções com mais de 20 anos. Estas por vezes apresentam algumas falhas, nomeadamente no que concerne a boas práticas de engenharia de software. Problemas como alto acoplamento e baixa coesão, falta de testagem, falta de documentação e uso de tecnologias que foram perdendo a adequação ao longo dos anos, são fatores que ajudaram a motivar o novo projeto e uma nova abordagem.

Assim, é importante compreender e aprender com erros que possam ter levado à degradação de qualidade de outras soluções, de forma a preencher lacunas detetadas. A nova solução deve ser considerada uma mais-valia, pelo que terá de trazer valor acrescentado para a organização.

Desta forma, e para dar a melhor resposta possível ao problema, o autor tomou a decisão de adotar a metodologia *Design Science Research Metodology (DSRM)* (Gregório *et al.*, 2021).

A *DSRM* cria e avalia artefactos orientados para resolver problemas organizacionais identificados. Esta metodologia deve ser baseada em teorias e conhecimento existentes, produzindo uma solução útil, de qualidade e avaliada com rigor, para um problema e área específica. Por outro lado, o desenvolvimento do artefacto é um processo de investigação que capitaliza teorias e conhecimentos do estado da arte, levando à solução do problema. O artefacto criado tem de resolver um problema relevante de uma forma inovadora (Gregório *et al.*, 2021).

De seguida, são descritas as diferentes fases que a metodologia segue:

A fase de **Identificação e Motivação do Problema** procura definir o problema a ser investigado e o valor de uma solução. Para responder da melhor forma a esta fase, são descritos o problema e os benefícios de uma possível solução, respondendo o presente capítulo, na Secção 1.2, a estas questões.

A fase de **Definição de Objetivos para a Solução** infere os objetivos da solução através da definição do problema e o conhecimento do que é concretizável. Mais uma vez, o presente capítulo documenta e responde aos *outcomes* pretendidos para esta fase, na Secção 1.3.

A fase de **Design e Desenvolvimento** materializa-se na criação do artefacto, o que inclui também a definição da sua arquitetura e a funcionalidade desejada. Esta fase é documentada nos capítulos Desenho da Solução e Implementação da Solução.

A fase de **Demonstração** comprova que o artefacto resolve o problema. O capítulo Experimentação e Avaliação documenta esta fase, através da avaliação do cumprimento da hipótese em investigação.

A fase de **Avaliação** engloba a observação e medição da aptidão da solução em resposta ao problema. O capítulo Implementação da Solução, na Secção 6.4, documenta esta fase, apresentando testes adequados para a apreciação de diferentes parâmetros da solução final.

Por fim, a fase de **Comunicação** implica a partilha de informação e descobertas feitas. Esta dissertação, no seu todo, documenta esta fase.

2 Contextualização

Neste capítulo é contextualizada toda a envolvente do projeto. Assim, inicialmente é feito um estudo sobre as capacidades e funções de um *Computerized Physician Order Entry* (CPOE), seguido de uma avaliação dos seus benefícios e desvantagens.

Dado que o âmbito do projeto prevê um CPOE inicialmente orientado à área de monitorização de sinais vitais, é feita uma análise a esta área. Adicionalmente, é explicado o projeto de inovação selecionado para impulsionar o CPOE.

2.1 Computerized Physician Order Entry

Um *Computerized Physician Order Entry* (CPOE) é um sistema computacional que permite a um médico, ou outro profissional de saúde autorizado, fazer pedidos médicos (Salvaneschi, 2010).

Os erros médicos nos hospitais são comuns e potencialmente uma ameaça para a segurança e saúde dos pacientes. Enquanto erros de medicação podem ocorrer em várias fases, é na prescrição que é particularmente comum acontecerem, frequentemente devido à documentação incorreta das ordens de medicação pretendidas. Assim, os sistemas CPOE são frequentemente propostos como um elemento importante para a melhoria da segurança na medicação (Jungreithmayr et al., 2020).

Explicando o seu funcionamento, os pedidos são transferidos através de uma rede computacional para os profissionais de saúde, incluindo enfermeiros, terapeutas e médicos, e também para imagiologia, farmácia e laboratório. Este é um sistema que converte a forma de prescrição manual ou com recurso a papel, para uma operação eletrónica, ajudando a reduzir custos (Aghazadeh, Aliyev and Ebrahimnejad, 2011).

Uma solução CPOE pode ainda funcionar integrada numa infraestrutura tecnológica mais alargada, estendendo habitualmente a funcionalidade EHR[2].

A Figura 1 ilustra a integração e *workflow* de um sistema CPOE com um EHR, fazendo uso dos dados dos pacientes. O CPOE pode comunicar também com um sistema de apoio à decisão (CDSS *system*) e com um sistema de gestão, que é responsável por aprovações de pedidos (*Practice management system*). Toda esta ergonomia possibilita e agiliza o processo de prescrição para diferentes áreas clínicas por parte do CPOE, nomeadamente as áreas de Farmácia, de Laboratório e de Imagiologia.

[2] Retirado em fevereiro de 2022 de https://www.altexsoft.com/blog/cpoe-systems-and-electronic-prescribing-software/

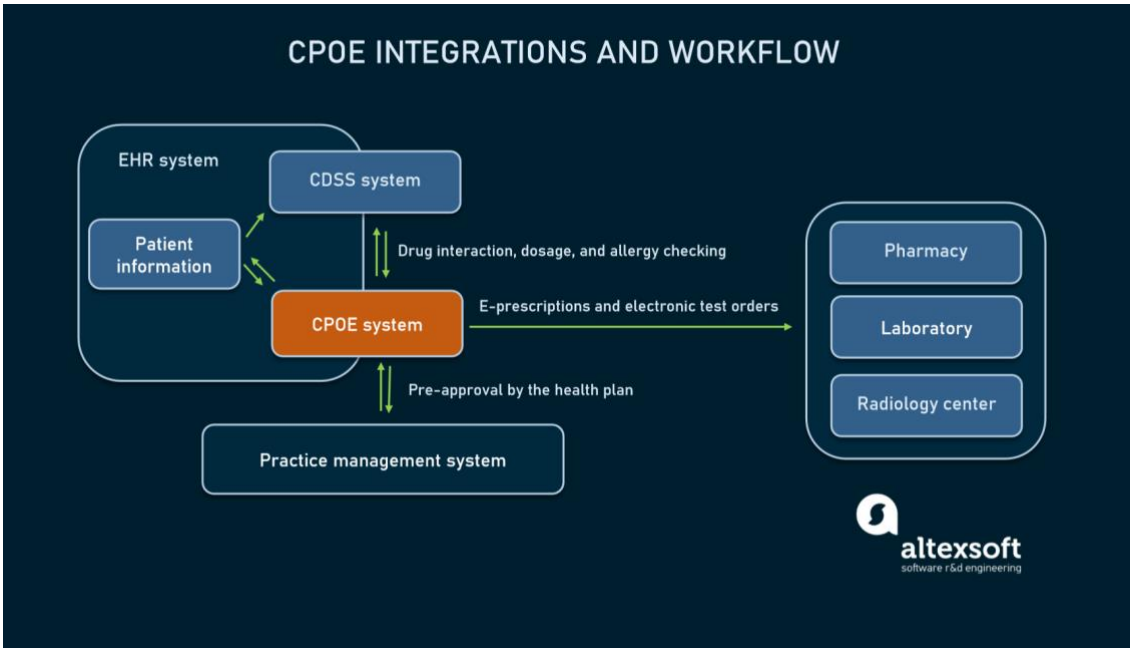

Figura 1 - Sistema CPOE integrado numa infraestrutura de software hospitalar[2]

Um cenário de uso pelo profissional de saúde, baseado num sistema como o exemplificado na Figura 1, inclui:

1. Login no EHR e início do processo de ordem de pedidos, através do CPOE.
2. Criação ou modificação de ordem de medicação, laboratório ou imagiologia de um paciente.
3. Validação e envio para outra solução ou serviço (farmácia, laboratório, entre outros) por parte do sistema.
4. Inclusão automática, no EHR, do pedido no histórico do paciente.

Todos estes passos tornam possível o acompanhamento dos pedidos, aumentado também a sua transparência[2].

Os sistemas CPOE têm vindo a ser adotados à escala global (Baysari *et al.*, 2017), sendo promovidos como relevantes para a segurança dos pacientes e para melhoria da qualidade e modernização da prática da saúde. No entanto, estes sistemas afetam de uma forma complexa a prestação de cuidados de saúde, acarretando benefícios, mas também riscos e desvantagens (Khanna and Yen, 2014).

Relativamente a **benefícios**, estudos demonstram que o CPOE pode ser um mecanismo efetivo para melhorar a segurança dos pacientes através da redução de erros de medicação e de atuação médica (Salvaneschi, 2010). Mais concretamente, erros como duplicação de medicação, overdoses, contraindicações e desconsideração de alergias são reduzidos, contribuindo na prática para a prevenção global de eventos adversos relacionados com medicação (Griffon *et al.*, 2017). Verificou-se também, em várias análises retrospetivas, que a implementação destes sistemas levou, frequentemente, à diminuição de erros de medicação através da eliminação de

pedidos ilegíveis. Entre outras fontes comuns de erro, os sistemas CPOE têm um potencial significativo para reduzir prescrições ambíguas, omissões de dados (Jungreithmayr *et al.*, 2020) e erros associados a prescrições de medicações com nomes semelhantes (Subramanian *et al.*, 2007).

Alguns investigadores indicam ainda que implementar um CPOE diminui em 80% erros médicos, 55% destes categorizados como severos e perigosos. Um outro estudo demonstrou que, na comparação de prescrições de medicação, as investigações reportam um decréscimo de 66% em erros na prescrição de medicação no grupo etário dos idosos (Aghazadeh, Aliyev and Ebrahimnejad, 2011).

Além da diminuição de diversos erros, o CPOE é também beneficamente associado a uma maior rapidez de envio de ordens para terceiros, como por exemplo a farmácia (Subramanian *et al.*, 2007).

O CPOE automatizou o processo de pedidos médicos, resultando em pedidos mais legíveis, completos e normalizados (Dhamanti *et al.*, 2021), possivelmente contribuindo para os benefícios supracitados.

Apesar dos dados demonstrarem que as implementações CPOE trouxeram muitos benefícios, como a diminuição de erros de medicação, o aumento do retorno de investimento, diminuição de despesas e uma melhoria no tempo de resposta superior, há desvantagens e continuam a existir problemas (Wang and Liu, 2019).

Abordando as **desvantagens**, verifica-se a grande dependência na tecnologia (Korb-Savoldelli *et al.*, 2018), situações relativas ao aumento de carga de trabalho para os funcionários médicos que usam o sistema (Korb-Savoldelli *et al.*, 2018; Wang and Liu, 2019) e o aparecimento de novos erros de prescrição de medicação, relacionados com erros do software e falta de conhecimento do sistema por parte dos profissionais de saúde (Wang and Liu, 2019).

O CPOE nem sempre diminui os erros médicos, ocasionalmente contribuindo para o seu aumento (Khanna and Yen, 2014). Estes erros podem ser explicados especialmente devido a deficiências da interface, ecrãs que induzem o utilizador em erro, *workflows* incorretos e má utilização do sistema.

Estes erros de medicação promovidos pelo sistema podem consistir em (Jungreithmayr *et al.*, 2020):

- Seleção incorreta de medicamentos nos menus de *dropdown*.
- Colocação incorreta de dados.
- Falhas na definição das configurações padrão de CPOE.

Analisando a vertente estatística, um ensaio realizado num hospital depois da implementação de um CPOE verificou um aumento de eventos adversos a medicamentos e um outro estudo

demonstrou desvantagens como aumento de eventos da mesma categoria, apesar de uma diminuição em 33% dos casos considerados evitáveis (Khanna and Yen, 2014).

O aumento de determinados erros, após implementação de sistemas CPOE, pode estar relacionado com o impacto inicial ou de curto prazo que estes provocam.

Para determinar o impacto que estes sistemas têm em erros de prescrição, foi feito um estudo (Rouayroux *et al.*, 2019) nos departamentos de diabetes e cardiologia de um hospital, imediatamente após a implementação do sistema CPOE. Os resultados demonstraram que, inicialmente, as duas categorias de erro mais comuns foram as de erros relacionados com a informatização e com a não conformidade com diretrizes ou contraindicações. Anteriormente, as duas categorias de erro mais comuns eram indicações não tratadas e dosagens supraterapêuticas.

Estes resultados indicaram que o perfil de erro de prescrição se alterou, com os erros de informatização a tornarem-se os mais frequentes.

No entanto, um ano após a implementação do sistema, verificou-se um decréscimo deste tipo de erros, possivelmente associado a uma melhor formação por parte dos profissionais de saúde para o uso destes sistemas e à melhoria da ergonomia deste software (Rouayroux *et al.*, 2019).

Os principais benefícios e desvantagens, extraídos da literatura supracitada, são sumariados na Tabela 1.

Tabela 1 – Prós e contras: CPOE

Prós	**Contras**
Redução de erros de medicação e atuação médica	Dependência na tecnologia
Aumento da rapidez de comunicação com sistemas terceiros	Aumento na carga de trabalho dos profissionais de saúde
Aumento do retorno no investimento	Erros iniciais associados ao uso do software

2.2 Âmbito CPOE

Um CPOE é uma solução que visa abranger várias áreas funcionais hospitalares. Não acrescenta funcionalidades apenas a uma área, mas sim a várias, dada a sua natureza.

No entanto, devido à sua elevada complexidade e âmbito, o documento foca-se na conceptualização e desenvolvimento das funcionalidades relativas a apenas uma das áreas, através de um MVP.

As próximas subsecções visam a revisão de literatura sobre essa área, Sinais Vitais e a sua monitorização.

2.2.1 Sinais Vitais

Os sinais vitais (Sapra, Malik and Bhandari, 2022) são uma medida objetiva das funções fisiológicas essenciais de um organismo vivo.

Definem-se como "vitais" pelo facto da sua medição e avaliação representarem o primeiro passo crítico para avaliações clínicas. São por isso alvo de avaliação nos processos de triagem de tratamentos e urgências hospitalares, refletindo e definindo a prioridade dos pacientes no acesso aos cuidados médicos.

A avaliação dos sinais vitais de um paciente pode ainda prever futuros resultados clínicos, assim como eventuais regressos às urgências e a frequência de readmissão nos hospitais. Inclusivamente, pode prever a utilização futura de determinados recursos de cuidados médicos.

Independentemente do paciente se encontrar nas urgências ou noutro departamento do hospital, a deteção precoce de agravamentos no quadro clínico é importante. Por exemplo, esta deteção de alterações nos sinais vitais é tipicamente correlacionada com uma deteção mais célere de alterações cardiopulmonares do paciente.

Atualmente existem ferramentas de alerta precoce que, através da deteção de anomalias nos sinais vitais do paciente, são críticas na previsão de paragem cardiorrespiratória e morte nas 48 horas após a sua medição. Estas ferramentas, através dos resultados das medições, calculam pontuações médias ponderadas, permitindo previsões e determinando a periocidade das medições e observações posteriores (Sapra, Malik and Bhandari, 2022).

Entre os Sinais Vitais, identificam-se[3]:

- **Frequência Cardíaca** – Mede o número de batidas do coração por minuto. Num adulto saudável, em repouso, este valor varia entre 60 e 100.
- **Frequência Respiratória** – É o número de respirações por minuto. Os valores normais, em repouso, para um adulto, variam entre 12 e 16 respirações por minuto.

[3] Retirado em agosto de 2022 de https://blog.earlysense.com/5-important-patient-vital-signs-to-ensure-patient-safety

- **Pressão Arterial** – Representa a força do sangue exercida sobre as paredes arteriais. Resulta em duas medidas, a sistólica – pressão na contração cardíaca - e a diastólica – pressão no relaxamento entre batimentos[4]. Em relação aos valores considerados saudáveis, a pressão sistólica deve ser inferior as 120 mmHg e a pressão diastólica deve ser inferior a 80 mmHg[5].
- **Temperatura** – Corresponde à temperatura corporal, variando normalmente entre 36.5 e 37.2 graus Celcius, num adulto saudável.
- **SpO2** ou **Saturação de Oxigénio** – Representa a fração da hemoglobina saturada de oxigénio, em relação à hemoglobina total presente no sangue. Os níveis normais de oxigénio no sangue estão compreendidos entre 95% e 100%.

2.2.2 Monitorização e Sensores

A monitorização dos sinais vitais é o método mais simples e barato de reunir informação clínica relevante dos pacientes nos hospitais. Analisando estatísticas, a má monitorização clínica tem sido associada a 31% de mortes evitáveis em Inglaterra, sendo este um problema à escala global (Kellett and Sebat, 2017).

Aliado a isto, o aumento de perturbações crónicas, como a hipertensão, condições respiratórias e diabetes, assim como o surgimento da pandemia da covid-19, têm contribuído para um aumento e valorização do mercado de dispositivos de monitorização de sinais vitais. Este mercado, em 2019 era avaliado em 4.63 biliões de dólares, estimando-se que o seu valor chegue aos 11.55 biliões até 2027[6].

Este crescimento deve-se também à incorporação do uso destes dispositivos de biomonitorização fora de contexto hospitalar, como em atletas, para a melhoria do sua desempenho desportivo (Li *et al.*, 2016; Salvo *et al.*, 2018), em diabéticos para a monitorização de glucose e administração de insulina (Cappon *et al.*, 2019) e em utilizadores de próteses para o controlo das mesmas (Tavakoli, Benussi and Lourenco, 2017).

2.2.2.1 Tipos de Sensores de Monitorização

Identificam-se dois tipos distintos de dispositivos de monitorização (Li, Chen and Lu, 2021):

- **Biossensores** – realizam monitorização, não invasiva, de biomarcadores específicos presentes em biofluídos, como suor, lágrimas, saliva e fluído intersticial (Kim *et al.*, 2019). Estes dispositivos incorporam elementos de reconhecimento biológico nos sensores.

[4] Retirado em agosto de 2022 de https://www.sns24.gov.pt/tema/doencas-do-coracao/hipertensao-arterial/#sec-1

[5] Retirado em agosto de 2022 de https://www.hospitaldaluz.pt/pt/dicionario-de-saude/pressao-arterial-e-hipertensao

[6] Retirado em agosto de 2022 de https://www.fortunebusinessinsights.com/vital-signs-monitoring-devices-market-103359

- **Sensores físicos** – monitorizam através da medição de sinais elétricos produzidos por deformações mecânicas, indicando os estados fisiológicos correspondentes.

Relativamente aos **biossensores**, na atualidade existem vários tipos diferentes que respondem a necessidades distintas.

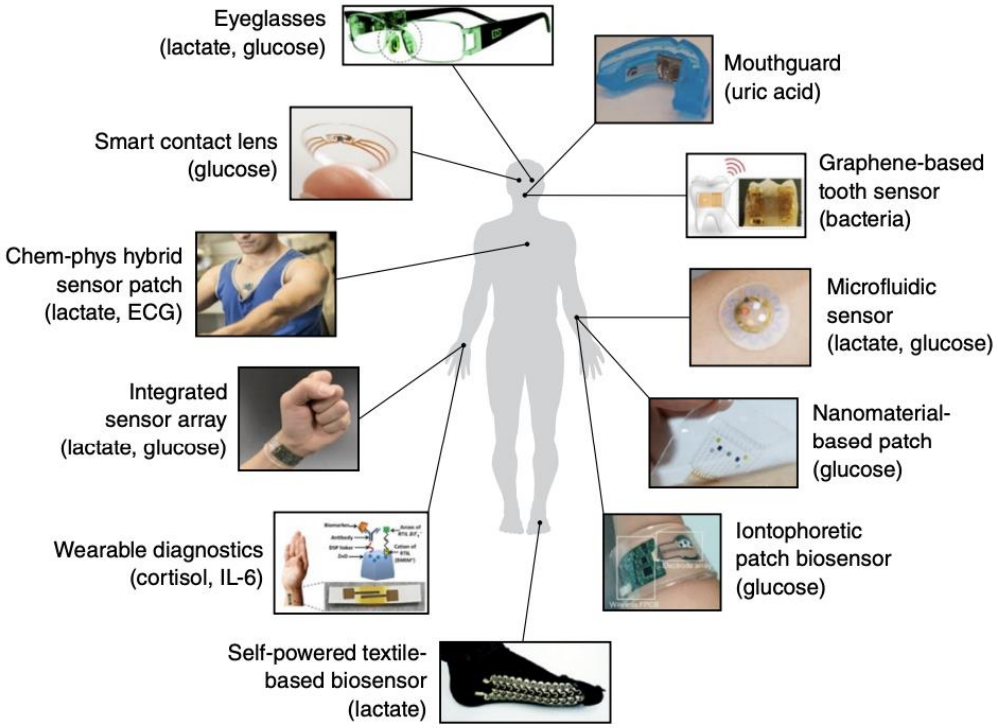

Figura 2 - Exemplos de biossensores (Kim *et al.*, 2019)

Na Figura 2 estão representados alguns exemplos de biossensores. Apesar de terem tipologias diferentes, seja material adesivo, têxtil, implante ou outro, todos eles realizam as monitorizações através do contacto com biofluídos.

Em adição à Figura 2, como resposta à pandemia de covid-19, têm surgido também biossensores para diagnosticar esta doença, importantes para o controlo da propagação do vírus SARS-CoV-2. (Yasri and Wiwanitkit, 2022).

Analisando os **sensores físicos**, estes funcionam com base no sinal de *output* relativo às variações nos seus parâmetros elétricos, detetando e quantificando-as. Para isto, fazem uso de uma variedade de transdutores[7] - dispositivos responsáveis por converter energia de uma forma para outra.

[7] Retirado em agosto de 2022 de https://gmw.com/transducers/

Estes sensores são utilizados na área da saúde, mas também no setor agrícola, na área militar e no desenvolvimento industrial (Ahmad and Salama, 2018). No contexto deste documento será apenas feito um levantamento de sensores ligados à saúde, os sensores físicos biomédicos.

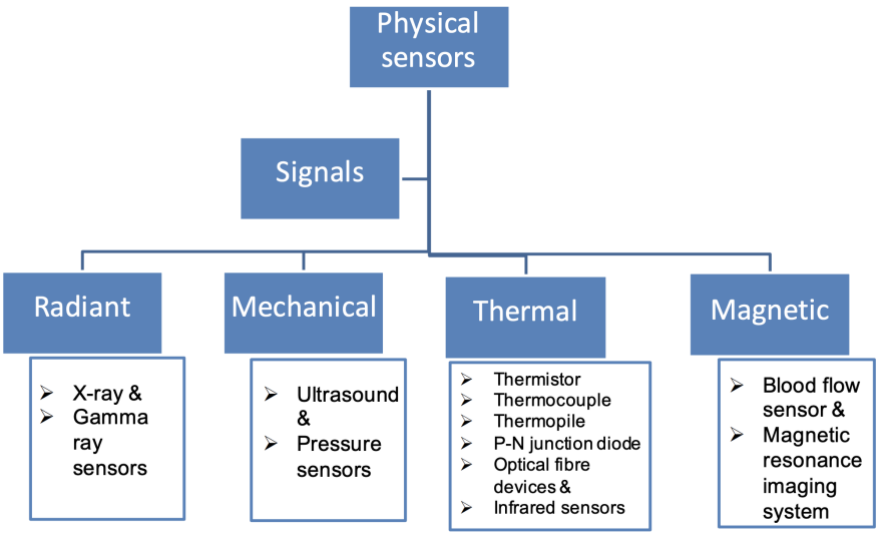

Figura 3 - Tipos de sensores físicos biomédicos baseados nos seus sinais (Ahmad and Salama, 2018)

Através da análise da Figura 3, identificam-se 4 categorias distintas de sensores físicos biomédicos, apresentando ainda exemplos para cada uma. A categorização baseia-se no tipo de sinais do sensor, podendo o sinal ser de radiação, mecânico, térmico ou magnético:

- **Radiação** - utilizados principalmente em Imagiologia e para tratamentos.
- **Mecânico** – permite monitorizar condições médicas, sem envolver procedimentos invasivos. Este sinal permite a medição da pressão arterial, como se pode ver na Figura 4, onde esta é monitorizada através do uso do sensor.
- **Térmico** – mede a temperatura corporal.
- **Magnético** – utilizados em tratamentos de hipertermia e em administração medicamentosa.

Em suma, apesar de existirem diferenças na forma de atuação entre biossensores e sensores físicos biomédicos, estes são, por vezes, alternativas para a mesma monitorização.

No entanto, os biossensores podem estar em desvantagem na precisão dos resultados das monitorizações. Em contrapartida, podem ter vantagem em relação aos sensores físicos, dado que estes equipamentos são mais suscetíveis a falhas e avarias (Li, Chen and Lu, 2021).

Dependendo do objetivo e da monitorização pretendida, é possível determinar qual tipo de sensor é mais adequado.

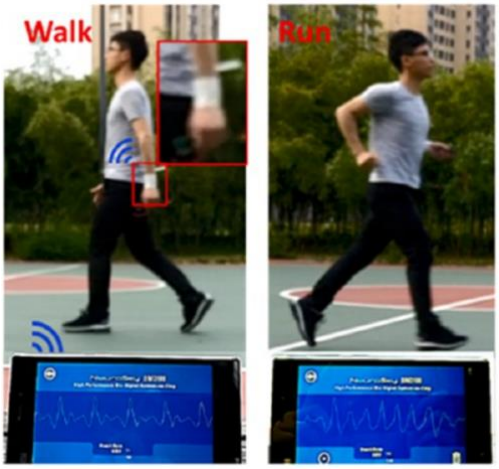

Figura 4 - Monitorização wireless de pressão arterial (Li, Chen and Lu, 2021)

2.3 Wireless biOmonitoring stickers and smart bed architecture: toWards untethered patients

No âmbito funcional de Monitorização de Sinais Vitais foi escolhido o projeto de inovação *Wireless biOmonitoring stickers and smart bed architecture: toWards untethered patients* (WoW), para a fase inicial de desenvolvimento do CPOE.

O projeto WoW[8], liderado pela Glintt-HS em consórcio com o Instituto Superior de Robótica, Universidade de Coimbra, Centro Hospitalar e Universitário de Coimbra (CHUC) e Universidade Carnegie Mellon, foi um projeto selecionado pelo programa "Go Portugal – Global Science and Technology Partnerships Portugal".

Conta com o apoio da Autoridade de Gestão Operacional Competitividade e internacionalização (COMPETE 2020) no âmbito do Sistema de Incentivos à Investigação e Desenvolvimento Tecnológico em Copromoção e envolve um investimento elegível de 1.1 milhões de euros, resultando num incentivo do Fundo Europeu de Desenvolvimento Regional (FEDER) de 743 mil euros.

Nas próximas subsecções o projeto é analisado ao pormenor através de informação que advém de conhecimentos obtidos junto dos fornecedores e da equipa de produto da Glintt-HS, bem como de documentação interna vinculada ao projeto.

[8] Retirado em agosto de 2022 de https://www.glintt.com/pt/o-que-somos/noticias/Paginas/projeto-wow-glintt.aspx

2.3.1 Contextualização do Projeto

Os pacientes hospitalizados são frequentemente ligados a vários instrumentos de medição quando é necessária uma biomonitorização contínua. Isto confina-os a camas, torna-os desconfortáveis e limita a sua mobilidade.

Estes instrumentos estão muitas vezes equipados de elétrodos de medição e o desprendimento destes do corpo do paciente, provocado por movimentos do mesmo, é uma das principais fontes de falsos alarmes médicos.

A hospitalização domiciliária, que contribui para reduzir os custos hospitalares, foi aprovada em Portugal em 2018. No entanto, as limitações nas infraestruturas para monitorização remota limitaram a lista de paciente elegíveis para hospitalização domiciliária. Isto inclui limitações na fiabilidade de hardware de dispositivos portáteis de monitorização, assim como em dispositivos seguros para os pacientes e ainda em soluções de software fiáveis para recolher e transferir dados de forma remota e distribuída.

A solução proposta pelo projeto *WoW* permite que sejam dadas altas hospitalares antecipadas a certos pacientes, sem que seja comprometida a sua saúde, através de monitorização pós hospitalar.

2.3.2 Objetivo

O objetivo do projeto *WoW* passa pela monitorização sem fios de sinais vitais, adicionando valor à hospitalização domiciliária.

O projeto propõe uma arquitetura centrada em adesivos com sensores de biomonitorização para pacientes, impressos a um custo entre 0.5 e 20 euros, de película fina e sem fios.

Adicionalmente, prevê a implementação de um novo software e arquitetura. Estes devem tornar possível a monitorização simultânea de múltiplos pacientes, através da recolha, processamento e transmissão centralizada de dados para um software hospitalar.

2.3.3 Dispositivos

A proposta do *WoW* passa por construir um sistema baseado em dispositivos de biomonitorização sem fios que se fixam facilmente à pele, através da epiderme, recolhendo diferentes sinais vitais e outros dados.

Estes dispositivos dividem-se entre **sensores** e ***smartboxes***.

Os **sensores**, que podem ser biossensores ou sensores físicos, realizam as monitorizações e pretendem cumprir as seguintes condições:

- Aderência à epiderme forte, não descolando com movimentação intensa ou com suor do paciente.
- Facilmente removíveis por prestadores de cuidados através de ferramentas específicas.
- Elásticos / extensíveis.
- Permitir a morfologia dinâmica da pele, sem comprometer a fixação dos sensores de medição à epiderme.
- Ser biodegradáveis.
- Utilizar materiais que, em contacto com a pele, são seguros para o paciente.
- No caso dos sensores físicos, os elétrodos do dispositivo devem ser também aderentes à epiderme, através do uso de um hidrogel biocompatível, constituído por poliacrimida e ácido acrílico.
- Usar energia *wireless*.
- Ser capacitados para trabalhar com transferência de dados.
- Associar-se a uma e só uma *smartbox*.

As ***smartboxes*** são nós que podem ser ligados às camas dos pacientes, em casa ou no hospital. Estas são unidades de energia e de transferência de dados recolhidos pelos sensores.

Uma *smartbox* comunica através das tecnologias *wireless* Radio Frequency Identification (RFID)[9] e *bluetooth*[10], sendo constituída por uma antena RFID, um leitor RFID, um módulo de *Wi-Fi*[11], um módulo de *bluetooth* e um computador.

Cada *smartbox* procura cumprir as seguintes condições:

- Estar, em cada momento, associada a um e só um paciente.
- Estar equipada numa cama, que passa assim a designar-se *smartbed*.
- Permitir a associação de sensores adesivos, dos quais recebe os dados de monitorização.
- Comunicar com os sensores através de RFID ou Bluetooth, funcionando corretamente em distâncias até 5 metros entre *smartbox* e sensores.
- Armazenar os dados das monitorizações em servidores locais ou em *cloud*, que comunica resultados e indicadores com o software hospitalar.
- Obter dados de monitorização, em tempo real, através de pedidos feitos no software hospitalar.

[9] Retirado em agosto de 2022 de https://www.techtarget.com/iotagenda/definition/RFID-radio-frequency-identification
[10] Retirado em agosto de 2022 de https://www.makeuseof.com/tag/what-is-bluetooth/
[11] Retirado em agosto de 2022 de https://www.cisco.com/c/en/us/products/wireless/what-is-wifi.html

Cada paciente e cada cama têm ainda *trackings* RFID, conferindo-lhes uma identidade digital. O hospital tem um dispositivo *gateway* RFID, que permite a associação entre pacientes e camas, completando este ciclo de relações, que inclui também as *smartboxes* e os sensores.

2.3.4 Arquitetura WoW

A arquitetura do sistema WoW, que engloba os dispositivos descritos na Subsecção 2.3.3, está representado na Figura 5.

A imagem esclarece as relações e forma de comunicação entre as várias peças de arquitetura.

O Paciente está associado aos sensores, que comunicam através de Bluetooth com uma *smartbox*. Esta comunica com a Gateway através da tecnologia *WI-Fi*, que por sua vez comunica através de *FHIR* com um módulo de interoperabilidade. Posteriormente, os dados já tratados são enviados para o software *WoW Services*. Este software é responsável por todas as funcionalidades a serem usadas pelos profissionais de saúde na interface do utilizador do *Viewer*, que consome esta peça de software.

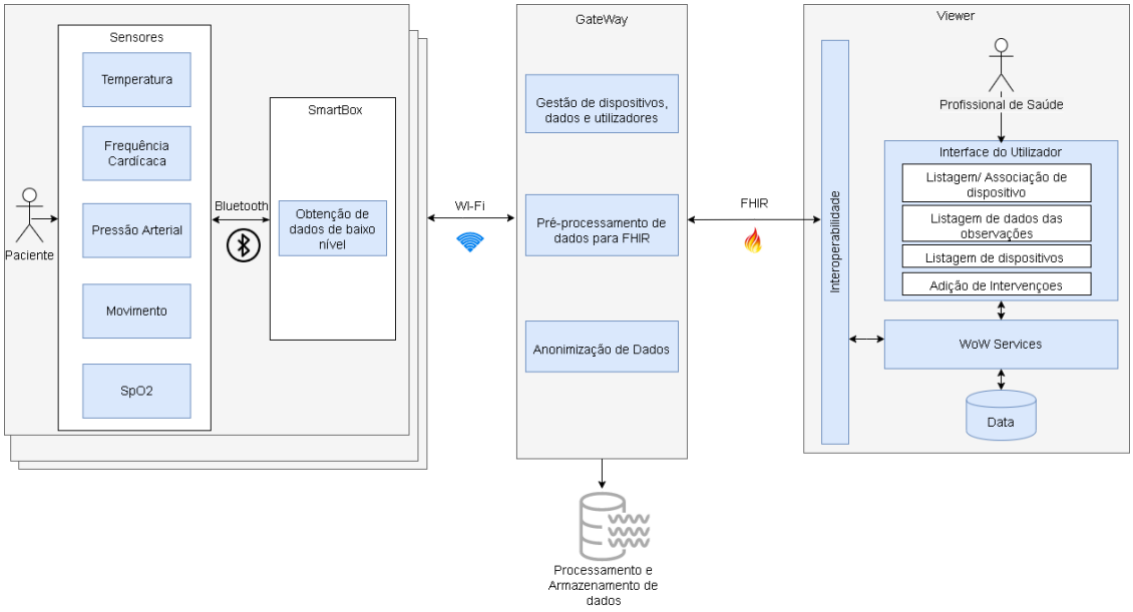

Figura 5 - Arquitetura do Sistema WoW

Numa perspetiva de utilizador mais visual, este sistema sem fios está representado na Figura 6, onde se pode ver a *gateway* do hospital, a *smartbox* equipada na cama e um sensor em uso por um paciente. Na parte inferior esquerda da imagem, pode observar-se que os dados das monitorizações são disponibilizados no software hospitalar de monitorização e, neste caso, armazenados em *cloud*.

O software apresentado na imagem é o *Globalcare*, também propriedade da Glintt-HS. No entanto, no contexto do documento o software é o *Viewer*.

Figura 6 – Representação Visual do uso dos dispositivos WoW

2.3.5 Monitorizações

Os tipos de monitorizações de sinais vitais incluídos foram escolhidos com base nas respostas a um questionário realizado ao staff médico do CHUC e em restrições identificadas pelos responsáveis.

De acordo com estes dados, os sensores incluídos no projeto foram:

- Frequência cardíaca
- Frequência respiratória
- Temperatura Corporal
- SpO2
- Pressão arterial

3 Estado da Arte

Este capítulo procura apresentar e explorar soluções de gestão hospitalar e soluções CPOE, comparando-as de acordo com aspetos relevantes para o projeto do autor.

Adicionalmente, são expostas tecnologias e ferramentas adequadas para a resolução do problema.

3.1 Soluções de software de gestão médica e hospitalar

Tendo em conta que o CPOE em estudo será consumido por aplicações hospitalares, segue-se a exploração do estado da arte deste tipo de softwares.

Este estudo ajuda a compreender o contexto no qual o CPOE se vai inserir, a oferta ao mercado e as caraterísticas principais de soluções desta área.

Dada a colaboração profissional do autor com a *Glintt-HS*, que lidera o mercado nacional com uma quota de mercado de 75%[12], é adequado fazer uma análise à sua principal solução, o *Globalcare*. Em contrapartida, é feita a análise a um outro software – Sistema Integrado de Gestão Hospitalar (SIGEHP) - concorrente no mercado português, enriquecendo o estudo a nível nacional.

De forma a conhecer alternativas no mercado internacional, serão também analisados outros softwares. Estes serão o *DrChrono* e o *athenaHealth*, os únicos avaliados com pontuações iguais ou superiores a 90/100, de acordo com avaliações realizadas pela *Business News Daily*[13], segundo critérios não divulgados.

3.1.1 Globalcare

O Globalcare[12] é um software de gestão hospitalar desenvolvido pela Glintt-HS.

Este software tem uma oferta modular, com várias funcionalidades que respondem a necessidades hospitalares e clínicas.

[12] Retirado em julho de 2022 de https://globalcare.glintt.com
[13] Retirado em julho de 2022 de https://www.businessnewsdaily.com/10914-best-electronic-health-records-systems.html

O *Globalcare* subdivide-se nos seguintes módulos:

- Sistema de Gestão Hospitalar
- Clínico
- Meios Complementares de Diagnóstico e Terapêutica (MCDT)
- Farmácia e Logística
- Interoperabilidade
- Envolvimento do Paciente

Sistema de Gestão Hospitalar

Este módulo é constituído pela Gestão de pacientes e pela Faturação.

A Gestão de pacientes tem as responsabilidades de natureza administrativa associadas à identificação e gestão do paciente. Possibilita o registo e monitorização de todo o processo e fluxo do paciente, desde o primeiro contacto com a unidade até à alta administrativa, permitindo também o acompanhamento de pacientes em ambulatório. Além disso, permite o controlo de listas de espera de consultas, exames e cirurgias, o processo de validação e verificação de seguros, a entrada de sinistrados e encaminhamentos. Quanto à consulta de informação operacional, dá acesso às listas de doentes, gestão de sala de espera, atendimentos, entre outros.

Suporta também a gestão em *backoffice*, disponibilizando a gestão de agendas, gestão de meios complementares de exames (termos, entrega de resultados) e gestão de sinistros.

A Faturação tem a responsabilidade de tratamento e emissão de documentos financeiros e de tesouraria.

Clínico

O módulo Clínico é constituído pelo *Eletronic Health Record* (EHR). O EHR gere a interação entre o médico e o paciente, permitindo aceder e registar todas as informações do foro clínico.

Entre estas informações incluem-se o acompanhamento da evolução clínica, os diagnósticos, alertas e problemas identificados, marcação de uma próxima consulta e um módulo de relatórios personalizável.

Meios Complementares de Diagnóstico e Terapêutica

O módulo Meios Complementares de Diagnóstico e Terapêutica (*MCDT)* permite, entre outros, o acompanhamento de todo o processo de exames.

Isto garante uma gestão centralizada do fluxo de exames, permitindo a realização de exames com características especificas por especialidade, a execução de relatórios clínicos predefinidos

por especialidade, tipo de exame ou médico. Permite ainda fazer o rastreio das ações realizadas pelos diferentes intervenientes no processo de execução de exames.

Farmácia e Logística

Este módulo é constituído pelo Circuito do medicamento. Permite a agilização de todas as fases do circuito do medicamento, com gestão e rastreio ao longo das várias fases – prescrição pelo médico, validação pelo farmacêutico e administração terapêutica pelo enfermeiro.

Interoperabilidade

O módulo de *Interoperabilidade* consiste numa plataforma que liga sistemas, permitindo o consumo de informações entre eles. Esta plataforma está ligada a um conjunto de conectores "*out-of-the-box*" fornecido pela base tecnológica *Mulesoft*[14] para disponibilizar, de uma forma ágil e incremental, uma oferta alargada, tanto em âmbito como em tipos de interfaces suportados.

Esta camada procura dar aos clientes uma maior autonomia no controlo dos seus processos de integração, permitindo a monitorização em tempo real dos mesmos. Um dos grandes objetivos da *Interoperabilidade* é garantir simultaneamente a coerência e a integridade dos dados a comunicar entre sistemas de terceiros e o *Globalcare*.

Envolvimento do Paciente

Este módulo representa aplicações, mobile e web, direcionadas para o uso do paciente, dividindo-se entre uma *app* e o portal do paciente.

Ambas as aplicações permitem: remarcação e cancelamento de consultas; consultar o histórico de visitas; obter informação detalhada sobre consultas, profissionais de saúde e preparação de exames; aceder a resultados de exames (relatórios e imagens); aceder ao histórico de pagamentos efetuados; consultar o pagamento de faturas pendentes.

3.1.2 Sistema Integrado de Gestão Hospitalar

O Sistema Integrado de Gestão Hospitalar[15] (SIGEHP) é um software desenvolvido pela *sisbit*[16], uma organização com soluções para a área da saúde.

O SIGEHP propõe-se a responder às necessidades de informatização global de hospitais, permite configuração e adaptação às especificidades das diversas organizações hospitalares.

[14] Retirado em fevereiro de 2022 de https://www.mulesoft.com/pt/
[15] Retirado em Fevereiro de 2022 de https://www.sisbit.pt/wp-content/uploads/2019/09/Flyer_SIGEHP_2019.pdf
[16] Retirado em Fevereiro de 2022de https://www.sisbit.pt/

O software é composto pelos seguintes subsistemas aplicacionais:

- Subsistema de Gestão de Doentes (SIGED)
- Processo Clínico Eletrónico (SISCLI)
- Subsistema de Farmácia Hospitalar (SISFARM)
- Subsistema de Aprovisionamento Hospitalar (SISPRO)
- Subsistema de Recursos Humanos Hospitalares (SIGERH)
- Subsistema de Serviços Financeiros Hospitalares (SISCONT)

Subsistema de Gestão de Doentes

O Sistema de Gestão de Doentes (SIGED) gere as admissões e altas dos doentes. Gere também os meios complementares de diagnóstico e terapêutica.

Processa todos os dados necessários à faturação, entidades responsáveis e aos doentes e tem ainda um módulo de apoio de gestão documental, que permite guardar os documentos de proveniência externa ou interna associados a um doente, classificados por episódio e tipo de documento.

Processo Clínico Eletrónico

O Processo Clínico Eletrónico (SISCLI) propõe uma abordagem completamente "*paper-free*" e, neste sentido, o SISCLI tem informatização clínica e de enfermagem nas áreas de internamento, Urgência, Consulta Externa, Hospital de dia, Bloco Operatório, o que inclui o resultado de meios complementares de diagnóstico. Este subsistema permite configurar perfis e permissões, de acordo com os postos de trabalho da instituição hospitalar, sendo estes, habitualmente, os de Médico e Enfermeiro.

Subsistema de Farmácia Hospitalar

O Subsistema de Farmácia Hospitalar (SISFARM) é constituído pelos módulos básicos de aquisições e gestão de stocks, orientados para as especificidades de cada artigo farmacêutico.

O médico, ao prescrever medicamentos ao doente internado, em Dose Unitária, envia esse pedido para a Farmácia, que envia a medicação para o serviço, onde a enfermagem executa a administração. A informação relativa à terapêutica constitui a história clínica do doente e o SISFARM disponibiliza os custos dos fármacos administrados durante o episódio de internamento.

Subsistema de Aprovisionamento Hospitalar

O Subsistema de Aprovisionamento Hospitalar (SISPRO) é constituído pelos módulos de aquisições e gestão de stocks, estando preparado para funcionar em termos de logística hospitalar.

Subsistema de Recursos Humanos Hospitalares

O Subsistema de Recursos Humanos Hospitalares (SIGERH) é o subsistema que permite efetuar a gestão dos profissionais que exercem atividade na Instituição.

Subsistema de Serviços Financeiros Hospitalares

O Subsistema de Serviços Financeiros Hospitalares (SISCONT) está integrado com todos os outros e está estruturado em vários módulos, configuráveis. Apresenta o Módulo de gestão de tesouraria, o módulo de gestão de terceiros, o Módulo de Orçamentação e o Módulo de Contabilidade Geral e Analítica.

3.1.3 DrChrono

A DrChrono[17], empresa sediada nos Estados Unidos da América, com software de igual nome, é utilizado por dezenas de milhares de médicos e serve mais de 17 milhões de pacientes.

De acordo com os planos disponibilizados, o custo e a solução oferecida são diferentes, podendo incluir algumas ou a totalidade das seguintes funcionalidades:

- Registo Médico Eletrónico (EMR)
- Telesaúde
- Gestão de Ciclo de Receitas (RCM)
- Gestão de Prática Médica
- Faturação
- Plataforma Mobile

Registo Médico Eletrónico

O módulo de Registo Médico Eletrónico (EMR) está presente numa plataforma totalmente integrada para atender às necessidades do médico, de acordo com a sua especialidade. Este oferece mais valias para garantir a eficiência, através de formulários personalizados, modelos de especialidade pré-feitos e atalhos para consulta de grafismos.

Telesaúde

Permite agendar e realizar consultas virtuais diretamente no software. Este cumpre as normas Health Insurance Portability and Accountability Act (HIPAA)[18], uma regulamentação que deve ser seguida pelas diferentes entidades de forma a proteger e garantir a segurança dos dados de saúde dos envolvidos.

[17] Retirado em julho de 2022 de https://www.drchrono.com/about/
[18] Retirado em julho de 2022 de https://www.varonis.com/blog/hipaa-compliance#definition

Gestão de Ciclo de Receitas

Em termos gerais, o módulo de Gestão de Ciclo de Receitas (RCM) corresponde ao processo em que os sistemas de saúde cobram pelos seus serviços e pelo qual geram receitas[19].

Em relação ao software em questão, o RCM é uma solução para este processo, que tem as seguintes funções e inclusões[20]:

- Relatórios de desempenho financeiro
- Declarações dos pacientes
- Painel controlo do estado das reclamações
- Gestão de contas a receber
- Gestão de faturação

Gestão de Prática Médica

Plataforma de ligação do médico ao paciente, com responsabilidades de agendamento, lembretes de consultas e check-in do paciente. É personalizável para cada profissional de saúde.

Faturação

Este módulo, integrado na plataforma, permite criar perfis de faturação personalizados e o preenchimento automático parcial da ficha de cada paciente. Oferece uma visão completa do estado de reclamações e permite analisar os dados financeiros do médico.

Plataforma Mobile

Esta é uma aplicação móvel que integra as funcionalidades do *Practice Management*, agendamentos, fluxo de trabalho clínico e faturação médica. Esta aplicação é compatível com o sistema operativo iOS, no entanto não é compatível com Android.

3.1.4 athenaOne

A athenaHealth [21], empresa também sediada nos Estados Unidos da América, detém o AthenaOne, um software médico abrangente, com os seguintes módulos[22]:

- athenaClinicals
- athenaCollector
- athenaCommunicator

[19] Retirado em julho de 2022 de https://www.athelas.com/insights/what-is-revenue-cycle-management
[20] Retirado em julho de 2022 de https://www.businessnewsdaily.com/16245-drchrono-medical-software.html
[21] Retirado em julho de 2022 de https://www.athenahealth.com/about/who-we-are
[22] Retirado em julho de 2022 de https://www.businessnewsdaily.com/16246-athenahealth-medical-software.html

athenaClinicals

Este módulo consiste num sistema *Electronic Health Record* (EHR). Entre outros, permite ao médico realizar ordens eletrónicas para análises e prescrições e, em simultâneo, adicionar diagnósticos. Permite também ter uma visão geral sobre alergias, condições médicas e histórico de medicação de um paciente, assim como o seu estado na sala de espera, histórico de visitas e sinais vitais.

As notas clínicas são outro fator a destacar neste módulo, onde é possível registar na ficha do paciente novas notas, em simultâneo com a consulta de informações prévias do paciente. Além da possibilidade de escrever as notas digitalmente, é permitido ainda registar notas através da voz.

Disponibiliza *templates*, ou seja, notas pré-construídas e personalizáveis para cada utilizador, de forma a permitir a melhor organização da informação relevante do paciente, por parte do profissional de saúde.

Os pedidos eletrónicos são outra valência do módulo, permitindo ao médico fazer requisições de laboratório, adicionar diagnósticos e prescrever medicação eletronicamente.

athenaCollector

O módulo athenaCollector é o módulo de ferramentas de faturação. Permite analisar as contas médicas enviadas para os seguros de saúde e gerir as recusas das mesmas.

athenaCommunicator

Este módulo consiste num portal para o paciente, onde este pode marcar novas consultas e requisitar prescrições que constem no seu histórico. Permite ainda ao paciente preencher formulários de admissão para as suas consultas e atualizar as suas informações médicas (sujeitas a aprovação de profissionais de saúde).

O paciente pode ser notificado, através deste portal, de marcações de consultas e de pagamentos em atraso.

O portal inclui ainda uma ferramenta de comunicação, com certificação HIPAA, permitindo ao paciente contactar, a qualquer hora, um profissional de saúde.

3.1.5 Comparação das soluções hospitalares

A Tabela 2 faz uma comparação entre as soluções hospitalares estudadas, com critérios considerados relevantes.

Os critérios definidos foram: a capacidade da solução ser interoperável, o preço a que é disponibilizada, a presença de sistema de faturação, de teleconsultas e a existência de aplicação mobile associada.

Os dois primeiros critérios (Interoperável e Preço) foram considerados relevantes devido ao impacto que terão na solução *CPOE* a ser definida. A solução terá de ser capaz de ser integrada em softwares hospitalares e o custo é sempre um fator a considerar, tornando estes critérios fundamentais para a comparação.

Em relação aos restantes critérios (Sistema de faturação, Sistema de Teleconsultas e Aplicação Mobile), foram também considerados fatores relevantes visto que tornam o software hospitalar mais completo.

Não foi possível obter informações relativas a certos critérios e soluções, estando devidamente assinaladas na Tabela 2.

Tabela 2 - Comparativo softwares hospitalares [12,15,17,20,22]

Solução	Interoperável	Preço	Sistema de faturação	Sistema de Teleconsultas	Aplicação Mobile
Globalcare	✓	N/A	✓	✓	✓
SIGEHP	N/A	N/A	✓	N/A	×
DrChrono	✓	$249 até $1200[23]	✓	✓	✓
AthenaOne	✓	Mínimo de $140 [23,24]	✓	✓	✓

3.2 Soluções CPOE - Trabalhos Relacionados

Nesta secção são analisados vários softwares com funcionalidades semelhantes às pretendidas para solucionar o problema deste documento. Assim, todos os softwares estudados nas próximas subsecções são ou têm competências associadas a sistemas CPOE.

Entre vários sistemas analisados pela *altexsoft*[2], foram selecionados para análise o PatientKeeper CPOE e o Practice Fusion, dado serem os únicos com responsabilidades CPOE.

Adicionalmente, também é estudado o software Power Orders. Este software é objeto de análise dado que foi o principal alvo de investigação pela equipa de produto responsável pela conceptualização da nova solução CPOE, que o considerou um modelo a seguir.

[23] Valor por profissional de saúde, por mês

[24] Retirado em julho de 2022 de https://www.forbes.com/advisor/business/software/practice-fusion-review/

3.2.1 PatientKeeper CPOE

O *Patientkeeper* CPOE afasta-se das tradicionais soluções CPOE, desenhadas para suportar auxílio ao *workflow* dos departamentos dos hospitais, oferecendo uma utilização orientada para o médico.

Este CPOE, com mais de 75000 utilizadores ativos[25], permite um acesso fácil e personalizável a protocolos e pedidos únicos.

Seguem-se alguns fundamentos desta solução[26]:

- **Pedidos** - permite fazer pedidos únicos e compostos, possibilitando ao hospital começar com acesso a pedidos standard já existentes, pedidos de terceiros, pedidos desenhados individualmente para cada médico ou uma combinação de todos estes. Os pedidos podem ser posteriormente alterados pelos médicos.
  É também possível fazer pedidos compostos de medicação através de outros sistemas auxiliares, pelo que torna possível aos médicos ter acesso, desde o primeiro dia de uso deste CPOE, aos pedidos que já lhes eram familiares.

- **Funcionalidades:**
  - Suporte à Decisão Clínica – realiza automaticamente verificações no momento da realização de pedidos.
  - Alertas configuráveis - são apresentados ao médico no ecrã, permitindo a classificação de criticidade. O nível de criticidade do alerta pode ser configurável de forma a tomar ações como interromper o *workflow* do pedido ou apenas apresentar a informação relevante, deixando à consideração do médico possíveis ações a ser tomadas.
- **Portabilidade** – a solução pode ser utilizada em através de *web* ou *mobile*, permitindo o acesso em diferentes plataformas e dispositivos.
- **Flexibilidade na implementação –** independentemente do nível de digitalização de informação do hospital, desde uma política completamente baseada no papel até a uma política completamente eletrónica, o sistema permite rapidez na implementação. Isto torna-se possível porque a solução é uma camada de software que pode existir acoplada aos sistemas existentes, permitindo ao hospital a escolha de como ou onde implementar o CPOE, sem obrigar à substituição dos sistemas.
- **Vantagem económica para os clientes** – pelas razões expostas no ponto anterior (Flexibilidade na implementação), os hospitais podem utilizar esta solução sem necessitar de eliminar ou substituir os seus sistemas, podendo manter as suas infraestruturas informáticas. Esta abordagem não disruptiva, permite ao hospital poupar tempo, esforço e dinheiro.

[25] Retirado em fevereiro de 2022 de https://www.patientkeeper.com/about-us/news/press-release.html?id=22747
[26] Retirado em fevereiro de 2022 de https://www.patientkeeper.com/product-brochures/patientkeeper_cpoe_product_sheet-feb12.pdf

Na Figura 7 está representado este software em funcionamento, em versão demo. No exemplo, apresenta-se um ecrã com os pedidos médicos já feitos e uma lista de novos pedidos, relativos a um doente.

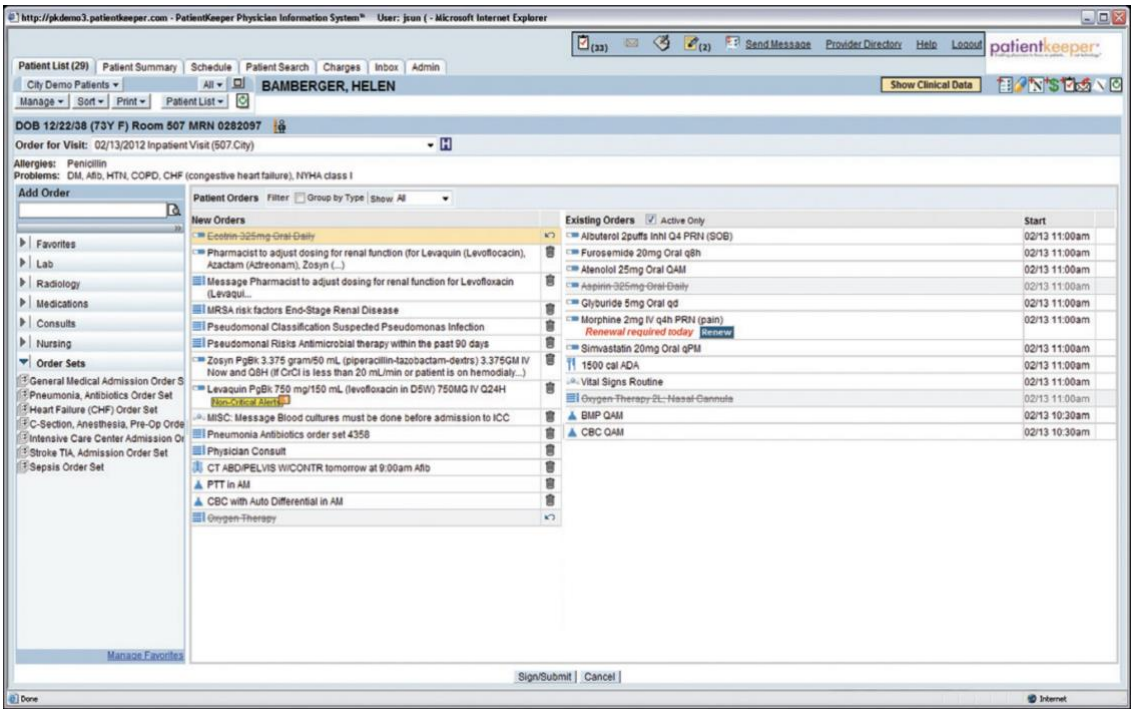

Figura 7 – *Printscreen* do PatientKeeper CPOE[26]

3.2.2 Practice Fusion

O Practice Fusion[27] é uma solução EHR em *cloud* que liga médicos, doentes e informação na mesma plataforma. O produto ajuda a racionalizar os fluxos de trabalho clínicos e a assegurar a prestação de serviços sem papel.

Entre os vários módulos que o constituem, um deles é o módulo de prescrição eletrónica (e-prescribing), responsável por permitir aos médicos o envio de receitas eletrónicas diretamente a partir das fichas dos seus pacientes[28]. Este módulo está capacitado com um sistema CPOE para prescrições de medicação.

[27] Disponível em fevereiro de 2022 em https://www.practicefusion.com/
[28] Retirado em fevereiro de 2022 de https://www.getapp.com/healthcare-pharmaceuticals-software/a/practice-fusion/)

Algumas funcionalidades relevantes deste CPOE incluem[29]:

- Agregação de informação relevante do paciente, como medicação ativa e lista de alergias.
- Verificação automática de interações entre medicamentos e de medicamento com alergias.
- Verificações automáticas de formulários de medicamentos.
- Acesso ao histórico de prescrições do paciente – ver Figura 8 - e conciliação de cada nova prescrição com a lista já existente de medicamentos.
- Acesso mais económico dos pacientes à medicação prescrita com *ecoupons* – descontos que, quando aplicáveis, são sugeridos ao médico, através de uma notificação do sistema. Este permite também que o médico envie esta informação para a farmácia, sendo a mesma processada antes do paciente a ir levantar.

Na Figura 8 pode ver-se um *printscreen* do software em utilização. Este é um ecrã que permite uma seleção de filtragem prévia ao acesso à funcionalidade de histórico de prescrições dos pacientes.

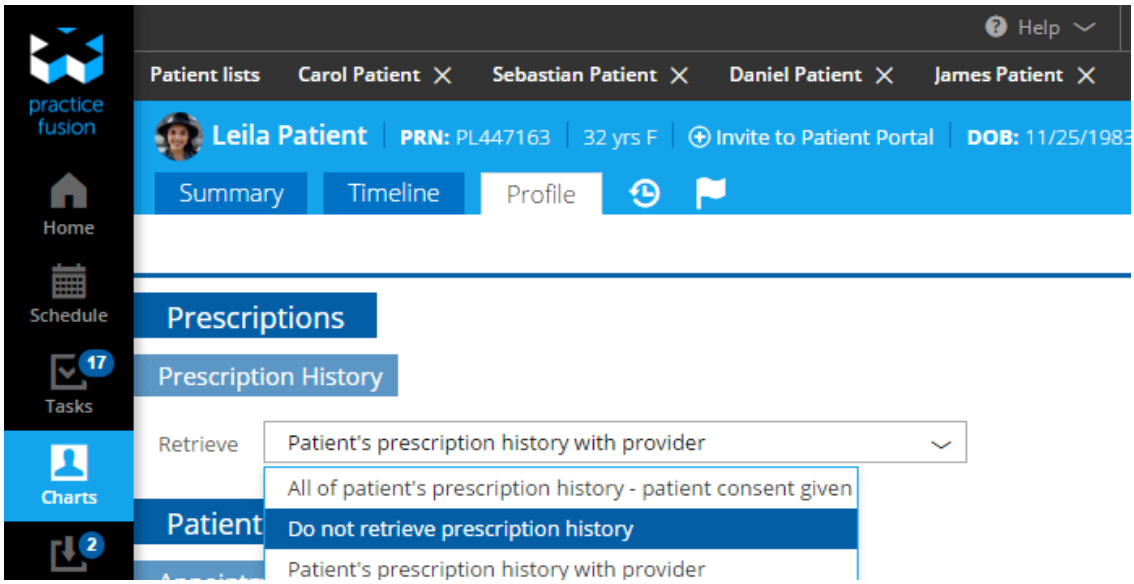

Figura 8 - Practice Fusion - Histórico de prescrição do paciente[28]

[29] Retirado em fevereiro de 2022 de https://www.practicefusion.com/e-prescribing/cpoe/

3.2.3 PowerOrders

O PowerOrders, propriedade da Cerner[30], é um software de prescrição médica baseado num sistema CPOE.[31]

De seguida, são apresentadas algumas das capacidades deste sistema, acoplado a diferentes soluções da Cerner[32]:

- **Medicação** – capacidade de registar, alterar e aceder eletronicamente a prescrições de medicação.
- **Laboratório** - capacidade de registar, alterar e aceder eletronicamente a prescrições de laboratório.
- **Diagnóstico de imagiologia** - capacidade de registar, alterar e aceder eletronicamente a diagnósticos de imagiologia.
- **Interações entre medicamentos / medicamento e alergias** – deteta e alerta os utilizadores finais de interações entre medicamentos e medicamentos e alergias, aquando do momento de prescrição. Também permite gerir o nível de criticidade em que os alertas de interação são acionados.
- **Formulário de medicamentos –** permite a pesquisa automática, pela existência de formulários de medicamentos, para um determinado paciente ou medicação.
- **Lista de medicamentos preferidos –** permite a pesquisa automática, pela lista de medicamentos preferidos, para um determinado paciente ou medicação.

Na Figura 9, pode ver-se a representação deste software. No caso, trata-se de um conjunto de ordens de admissão de um acidente vascular cerebral isquémico.

Os círculos azuis com um "x" branco representam campos que o médico tem de preencher antes de assinar e ativar todo o conjunto de ordens (Khanna and Yen, 2014).

[30] Disponível em fevereiro de 2022 em https://www.cerner.com/

[31] Retirado em fevereiro de 2022 dm https://healthmanagement.org/products/view/prescription-software-medical-powerorders-r-cerner

[32] Retirado em fevereiro de 2022 de https://www.cerner.com/-/media/cerner-media-united-states/certified-health-it/cerner-2015-edition-certified-health-it-costs--limitations-disclosures-v32521revised.pdf?vs=1&hash=3520C2110046B4BBEC2BB4CBBC5D968F

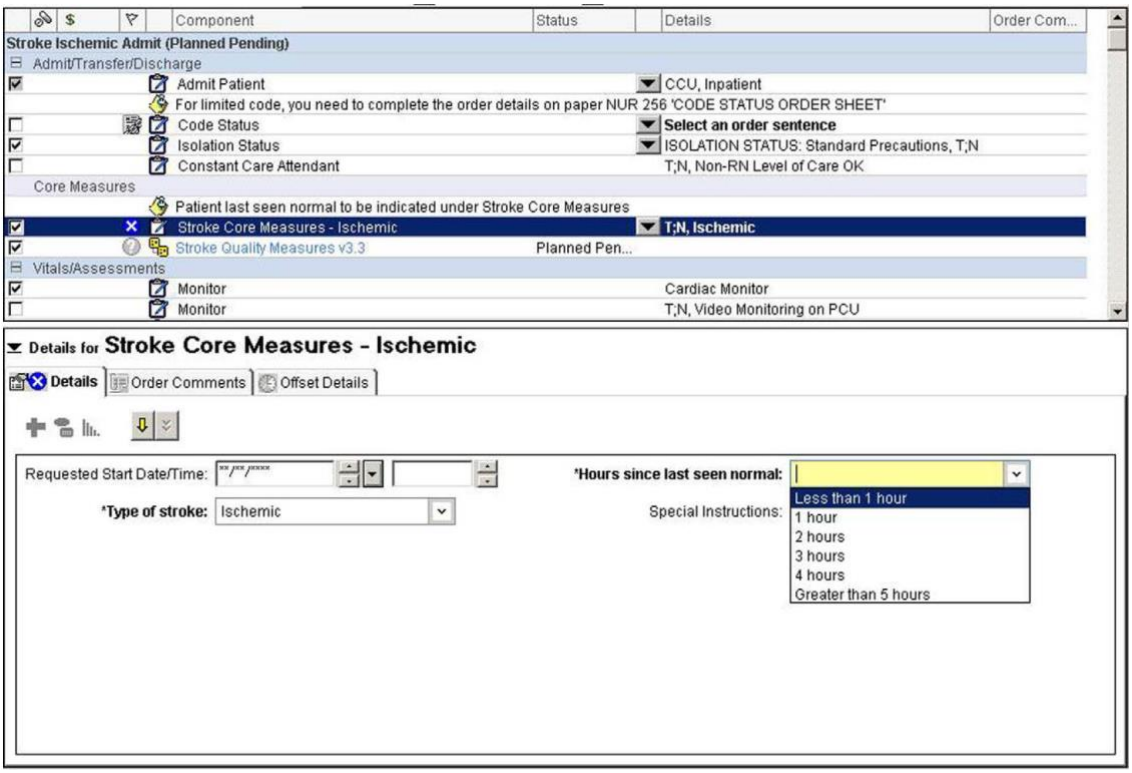

Figura 9 - Cerner CPOE - Caso de Uso (Khanna and Yen, 2014)

3.2.4 Comparação dos Trabalhos Relacionados

Após o estudo de três diferentes softwares relacionados com a nova solução que motiva este documento, na Subsecção 3.2, é pertinente comparar as diferentes soluções, de forma a identificar a ferramenta mais próxima da que se pretende desenvolver e as suas caraterísticas.

Após a fase de pesquisa e estudo, verificou-se a escassa existência de informação sobre estas soluções, disponibilizada pelos proprietários ou mesmo terceiros. Possivelmente por razões estratégicas, toda a informação que foi possível recolher, situa-se numa vertente mais comercial e menos tecnológica e arquitetural.

3.2.4.1 Análise Comparativa

Entre as três soluções, a Patientkeeper CPOE é única que se apresenta como uma solução independente.

As outras soluções estudadas são sistemas com responsabilidades não exclusivas de um sistema CPOE ou que dependem de sistemas terceiros, não sendo possível o seu desacoplamento. Ou seja, apresentam caraterísticas que as afastam da solução que se procura.

De seguida, a Tabela 3 confronta as soluções segundo diversos critérios relevantes para a solução que se procura neste projeto. Não foi possível obter informações relativas a certos critérios de duas soluções, estando devidamente assinaladas.

Tabela 3 - Comparativo de soluções CPOE[2,33]

Solução	Solução Independente	Compatibilidade com Sistemas externos	Verificações medicamento-medicamento / medicamento - alergia	Preço
PatientKeeper CPOE	✓	✓	✓	N/A
PracticeFusion	×	×	✓	$149/ mês[24]
PowerOrders	×	×	✓	N/A

3.3 Contextualização Tecnológica

Nesta secção são estudadas tecnologias e ferramentas para solucionar o problema. Para isto é feita uma investigação de diferentes tecnologias para responder aos problemas associados à implementação da solução.

Dado que o trabalho responde a vários problemas em diversas áreas de desenvolvimento, por uma questão de simplicidade e para não estender demasiado o documento, o autor tomou a decisão de fazer o estudo relativo a *frameworks* de *frontend*, sistemas de comunicação para *Application Programming Interfaces* (APIs) e padrões de comunicação de saúde, considerando-os mais relevantes para o projeto.

3.3.1 Tecnologias de frontend

Nesta subsecção são avaliadas três tecnologias de *frontend* com potencial para responder, em parte, ao problema. No final, é feita uma análise comparativa, permitindo compreender os prós e contras de cada uma.

React.js

O React [34] é uma biblioteca JavaScript para o desenvolvimento de interfaces de utilizador. Baseia-se numa lógica de componentes reutilizáveis que gerem o seu próprio estado e, em conjunto, formam o todo da interface do utilizador. Outra caraterística fundamental do React é o uso de *Virtual DOM*, que pode ser renderizado no lado do cliente ou no lado do servidor,

[33] Retirado em fevereiro de 2022 de https://healthmanagement.org/products/view/prescription-software-medical-powerorders-r-cerner
[34] Retirado em fevereiro de 2022 de https://reactjs.org/

comunicando de forma bidirecional. O *Virtual Data Object Model (DOM)* renderiza sub-árvores de nós baseado em mudanças de estados. O *DOM* é, assim, objeto do mínimo de manipulação possível para que os componentes estejam sempre atualizados (Kumar and Singh, 2016).

Através desta renderização inteligente, o React não interage com o *DOM* gerado pelo browser, mas sim com o *DOM* guardado em memória. Isto permite um maior e mais robusto desempenho das aplicações.

As comparações entre o *Virtual DOM* e o *DOM* são feitas através de um algoritmo que é capaz de compreender os nós que se alteram, atualizando apenas o estritamente necessário.

Tudo isto leva à capacidade de desenvolvimento de aplicações web complexas, onde os dados podem mudar, sem que isso implique a renderização total do *DOM* e, portanto, sendo benéfico para o desempenho. (Aggarwal, 2018).

Angular

O Angular[35] é uma *framework* de desenho de aplicações e uma plataforma de desenvolvimento, indicada para a criação de aplicações *single-page* eficientes. É construída em TypeScript[36] e inclui uma *framework* para construir aplicações *web* escaláveis, tem uma coleção de bibliotecas que envolvem capacidades como *routing*, gestão de formulários, comunicação cliente-servidor, entre outros. Está também capacitada de um conjunto de ferramentas que facilitam o desenvolvimento, construção, testagem e atualização do código[37].

A injeção de dependências, que permite o carregamento automático de novos módulos que possam ser necessários e a sincronização automática dos dados entre as camadas de *model* e *view*, do padrão arquitetural Model View Controller (MVC), são algumas das caraterísticas que conferem distinção ao Angular (Kumar and Singh, 2016).

O Angular tem também um sistema de ligação de dados bidirecional, mantendo as camadas de *view* e *model* em constante sincronização, o que permite que as alterações na camada de *model* sejam automaticamente refletidas na camada de *view*[38].

Vue.js

O Vue[39] é uma *framework* de JavaScript para a construção de interfaces gráficas. Tem um modelo de programação declarativo e baseado em componentes e é reativo às alterações de estados do JavaScript, atualizando o *DOM* quando estas acontecem.

[35] Retirado em fevereiro de 2022 de https://angular.io/docs
[36] Disponível em fevereiro de 2022 em https://www.typescriptlang.org/
[37] Retirado em fevereiro de 2022 de https://angular.io/guide/what-is-angular
[38] Retirado em fevereiro de 2022 de https://www.clariontech.com/blog/angular-features-what-it-brings-to-us
[39] Retirado em fevereiro de 2022 de https://vuejs.org/guide/introduction.html

Além de permitir o desenvolvimento de interfaces *web* e aplicações *single-page*, também permite o desenvolvimento de aplicações *desktop* e *mobile*.

O Vue faz uso do *DOM* virtual, ao invés de atualizar todo o *DOM* sempre que alguma alteração se verifica, permitindo uma renderização mais rápida do ecrã e aumentando o desempenho da aplicação[40].

Relativamente a bibliotecas e ferramentas, esta Framework disponibiliza[40]:

- **Vue CLI** – interface de linha de comandos para o desenvolvimento e instalação das bibliotecas Vue, assim como *plugins* de terceiros.
- **Ferramentas de desenvolvimento** – ferramentas para o *debug* da aplicação construída com Vue.
- **Vue Loader –** responsável por carregar *webpacks*.
- **Vue Router –** responsável pelo roteamento de mapeamento dos componentes.

Storybook

O *Storybook* [41] é uma ferramenta *open-source* para desenvolver componentes isolados de interface de utilizador para o *React, Vue, Angular,* entre outros. Esta ferramenta ajuda a documentar componentes para a sua reutilização e permite o teste visual dos componentes para precaver possíveis *bugs.*

O *Storybook* permite às equipas de desenvolvimento ter um espaço para construir componentes, que podem ser baseados em bibliotecas de *front-end*, com diferentes variações. Isto torna-se útil especialmente quando se prevê que o componente pode ser utilizado mais que uma vez na solução a desenvolver. Assim, existe a garantia de coerência no uso múltiplo do componente.

Figura 10 - Pertinência do *Storybook* [41]

[40] Retirado em fevereiro de 2922 de https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/
[41] Retirado em fevereiro de 2022 de https://storybook.js.org/

Esta ferramenta disponibiliza uma interface gráfica que permite consultar todos os componentes desenvolvidos e como os utilizar na solução.

Como se pode interpretar a partir da análise da Figura 10, o Storybook e a biblioteca de componentes trabalham em conjunto de forma a servir a aplicação.

3.3.1.1 Análise Comparativa

Dado que o Storybook é uma ferramenta que acrescenta valor quando aliada ao uso de qualquer uma das outras *frameworks* apresentadas, não é considerada nesta análise. Esta será usada, independentemente da escolha final, de forma a fornecer os componentes de interface gráfica necessários.

Relativamente às *frameworks* restantes – React, Angular e Vue – existe semelhança no propósito que pretendem servir. Isto torna-as mutuamente exclusivas aquando da escolha final para a resposta ao problema.

Assim, segue-se uma análise entre estas *frameworks*, na Tabela 4, através de diversos critérios pertinentes para a tomada de decisão da opção mais acertada para o projeto.

Antes do exercício de escolha da tecnologia a utilizar, fica uma nota relativa ao Critério Desempenho: a diferença que coloca o Angular num patamar inferior em relação às outras duas tecnologias prende-se com o *DOM*. Ao contrário destas, o Angular não utiliza o *DOM* virtual. Ao invés, faz uso do *DOM* regular, implicando uma re-renderização a cada alteração do código, por menor que seja[42].

Comparando os resultados, a tecnologia *React* é aquela que apresenta melhores resultados na globalidade dos critérios. A sua escalabilidade e alto desempenho serão fundamentais para a nova solução CPOE, dado que é expectável o seu crescimento para diversas áreas além da de monitorização de sinais vitais. A possibilidade de criação de componentes independentes e reutilizáveis é também de grande importância, garantindo coerência e poupando retrabalho desnecessário.

Assim, a escolha para o desenvolvimento da camada de *frontend* da solução recai na tecnologia *React*. Esta decisão vai de encontro aos resultados obtidos através do uso do modelo *Analytic Hierarchy Process* (AHP) para comparação destas três tecnologias, disponível no Apêndice A.1.3.3, relativo à Análise de valor do Apêndice A.

Tabela 4 - Análise Comparativa: React, Angular e Vue[42]

Critério	React	Angular	Vue
Componentes independentes e reutilizáveis	✓	✓	✓
Escalabilidade	Alta	Alta	Baixa
Desempenho	Alto	Médio	Alto
Popularidade (Contribuidores[43])	+1500	+1500	+400

3.3.2 Sistemas de comunicação para API's

Nesta subsecção são avaliados três sistemas de comunicação, o *GraphQL*, o *gRPC* e o *REST*. No final, é feita uma análise comparativa, permitindo compreender os prós e contras de cada um destes e os cenários em que cada tecnologia é mais adequada.

GraphQL

O GraphQL é uma linguagem *query* e um ambiente de execução para APIs.

Esta linguagem fornece uma descrição do modelo de dados, permitindo fazer *requests* onde são descritos os dados exatos pretendidos na resposta. É independente de qualquer base de dados, podendo ser implementada em qualquer contexto em que uma API é utilizada (Seabra, Nazário and Pinto, 2019) .

Originalmente desenvolvida pelo *Facebook*, está disponível ao público desde 2015. O GraphQL permite fazer pedidos aos *data services* de forma personalizada diretamente pelo cliente, permitindo precisão sobre os dados que devem ser enviados de volta na resposta do servidor. O modelo de interação torna-se assim flexível e eficiente, precavendo desta forma situações em que possivelmente se estaria a sobrecarregar a largura de banda com dados desnecessários para o cliente (Diaz, Olmedo and Tanter, 2020) .

Google Remote Procedure Call (gRPC)

O *gRPC*[44] é uma *framework* de Remote Procedure Call (RPC), ou seja, de um software de protocolo de comunicação, que pode correr em qualquer ambiente.

[42] Retirado em fevereiro de 2022 de https://relevant.software/blog/angular-vs-react-vs-vue-js-choosing-a-javascript-framework-for-your-project/#Scalability
[43] Retirado em outubro de 2022 de https://www.codeinwp.com/blog/angular-vs-vue-vs-react/
[44] Retirado em outubro de 2022 de https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC

Esta *framework* é *open-source*, com um sistema baseado em contratos de dados e um protocolo de comunicação multiplataforma que permite simplificar e gerir a comunicação entre serviços.

Os *protocol buffers* são o mecanismo que esta tecnologia usa, por defeito, para a serialização de estruturas de dados, à semelhança do uso do *JSON* por outras tecnologias como o *Representational State Transfer* (REST). Os contratos de dados são assim definidos em ficheiros de *protocol buffers*, através de uma linguagem própria para o efeito, o *proto3*, que permite vários tipos numéricos, de datas, listas, entre outros, de forma a definir as mensagens de *input* e *output*.[45]

Representational State Transfer (REST)

O REST[46] é um estilo de arquitetura que define uma série de restrições padrão entre sistemas na *web*, facilitando a sua comunicação entre eles.

Estes sistemas, designados de *RESTful*, são caraterizados pela forma como são independentes e separam as responsabilidades de cliente e servidor. Assim, a implementação cliente e a implementação servidor são modulares e não existe conhecimento mútuo.

Ao separar as questões relacionadas com a interface do utilizador e as questões relacionadas com o armazenamento de dados, a flexibilidade da interface em diferentes plataformas é melhorada, assim como a escalabilidade de dados.

Nesta arquitetura, o cliente envia pedidos para receber ou alterar recursos e o servidor responde a esses pedidos.

3.3.2.1 Análise Comparativa

Na Tabela 5 é feita uma análise comparativa através de diversos critérios.

De acordo com a comparação feita, segundo critérios relevantes para as necessidades da solução que motiva este documento, facilmente se coloca de parte a opção REST. Esta seria uma opção que iria sacrificar o desempenho e a rapidez de desenvolvimento em grande escala, quando comparada com as outras alternativas. Entre as restantes, existe um equilíbrio maior e têm diferentes características relevantes para a solução.

Dado que se prevê situações em que o *GraphQL* será a tecnologia mais adequada e outras em que o *gRPC* serve melhor a solução, tomou-se a decisão de fazer uso das duas tecnologias.

Para explicar esta decisão, é necessário desde já antecipar que a estrutura da solução será microaplicacional (esta será detalhada no decorrer do documento, especificamente no Capítulo Desenho da Solução). Assim, para o *backend*, prevê-se que vá existir uma grande quantidade de dados e, portanto, a prioridade recai sobre o desempenho, onde a rapidez de comunicação é o mais relevante, pelo que será usado *gRPC*. Já para o *frontend*, é necessária flexibilidade na

[45] Retirado em outubro de 2022 de https://www.toptal.com/grpc/grpc-vs-rest-api
[46] Retirado em fevereiro de 2022 de https://www.codecademy.com/article/what-is-rest

especificação dos dados a receber em todos os pedidos, de forma a filtrar apenas pelo estritamente necessário, pelo que será usado *GraphQL*.

Tabela 5 - Análise Comparativa: REST, gRPC e GraphQL[47]

Critério	GraphQL	REST	gRPC
Desempenho	Rápido	Mais lento	Rápido
Rapidez de desenvolvimento	Rápido	Mais lento	Rápido
Estabilidade	Sujeito a menos erros e com validações automáticas	*Queries* complexas	Alta
Casos de Uso mais apropriados	Flexibilidade na especificação dos dados necessários	Aplicações a ser usadas por muitos clientes	Rapidez prioritária na comunicação entre serviços
Pedidos com filtragem personalizável	✓	×	×

3.3.3 Padrões de comunicação de saúde

A interoperabilidade[48] é a capacidade de diferentes aplicações, dispositivos e fontes de informação, acederem, trocarem, integrarem e fazerem uso dos mesmos dados de uma forma cooperativa e coordenada.

A Health Information Exchange [48] (HIE) permite a troca de informação clínica entre estes sistemas de informação de saúde, mantendo a integridade total da informação. O objetivo desta troca de informação é o acesso e recolha de dados clínicos, de forma que o doente seja afetado positivamente com cuidados seguros, oportunos, eficientes, eficazes e equitativos.

De seguida, são apresentados diferentes padrões de comunicação de saúde.

Fast Healthcare Interoperability Resources (FHIR)

O FHIR [49] é um padrão desenvolvido pela Health Level Seven[50] (HL7) para o âmbito da saúde. As soluções do FHIR são construídas por um conjunto de componentes modulares denominados

[47] Retirado em fevereiro de 2022 de https://www.mobilelive.ca/blog/graphql-vs-rest-what-you-didnt-know/
[48] Retirado em fevereiro de 2022 de https://www.himss.org/resources/interoperability-healthcare
[49] Retirado em fevereiro de 2022 de https://www.hl7.org/fhir/summary.html
[50] Disponível em fevereiro de 2022 em https://www.hl7.org/

de recursos. Estes recursos podem ser facilmente reunidos em sistemas que resolvem problemas clínicos e administrativos.

O FHIR é baseado em *standards* utilizados em grande escala fora da área da saúde. Um exemplo disso é a arquitetura REST, que torna possível a troca de informação necessária.

O desenvolvimento deste padrão começou em 2012 em resposta a um mercado que necessitava de melhores métodos para troca de dados de saúde, numa altura em que a quantidade destes dados estava a crescer rapidamente.

Além de permitir esta resposta benéfica para a troca de dados, o FHIR assegura as seguintes vantagens para os programadores[51]:

- **Foco na implementação rápida e fácil** - desenvolvedores reportaram a implementação de interfaces simples no espaço de um dia.
- **Uso gratuito -** sem restrições.
- **Suporta sistemas de diversos fornecedores –** inclui sistemas da Apple, Microsoft, Google, Epic, Cerner e outros sistemas EHR da maior parte de outros fornecedores.
- **Inclui ferramentas online, gratuitas e descarregáveis –** inclui bibliotecas de implementação e servidores para referência.
- **Disponibilidade de exemplos de casos de uso –** públicos para ajuda no início do desenvolvimento de novas aplicações
- **Especificações online –** facilitam a aprendizagem para o desenvolvimento.
- **Comunidade global –** torna o suporte mais robusto.
- **Base forte em vários padrões web –** inclui XML, JSON, HTTP e OAuth.

Consolidated Clinical Document Architecture (C-CDA)

O C-CDA[52] é um padrão de *markup* exclusivo XML. Este padrão define como os documentos são codificados e, apesar de permitir a sua exportação, não define como deve ser feita a troca de informação.

Cada episódio de um doente no sistema de saúde pode ser representado por um único documento do tipo C-CDA[53]. Estes documentos consistem em dois componentes, um legível e com acesso através de um navegador de internet, e um outro codificado e orientado para o tratamento automático de dados[54].

Quanto à sua utilização, muitos desenvolvedores demonstraram dificuldade em trabalhar com o C-CDA. Este foi desenhado para facilitar trocas de dados e não para o armazenamento dos

[51] Retirado em fevereiro de 2022 de https://www.healthit.gov/sites/default/files/2019-08/ONCFHIRFSWhatIsFHIR.pdf
[52] Retirado em fevereiro de 2022 de https://www.particlehealth.com/blog/what-is-fhir
[53] Retirado em fevereiro de 2022 de https://www.particlehealth.com/blog/what-is-ccda-consolidated-clinical-document-architecture
[54] Retirado em fevereiro de 2022 de https://www.ihs.gov/rpms/PackageDocs/BCCD/bccd020u.pdf

mesmos, o que leva a que muitas vezes estes documentos sejam extensos, com mais de 200 páginas, tornando complicado o processo de pesquisa dentro do documento. Exemplificando com um caso prático, não é possível fazer um pedido de informação da medicação de um doente, mas apenas um pedido com todos os registos[53].

3.3.3.1 Análise Comparativa

Apesar do FHIR e do C-CDA serem alternativas para a exportação de registos médicos de pacientes, diferem numa série de fatores, onde o C-CDA se mostra mais limitado dado não passar de um padrão, enquanto o FHIR é uma API[52].

Segue-se uma análise entre estas *tecnologias* na Tabela 6 , através de critérios relevantes para a solução.

Como se pode ver, o FHIR é uma solução muito mais flexível, permite a especificação da informação, apresenta uma vasta quantidade de mensagens de *markup* compatíveis, tem uma comunidade muito superior ao C-CDA e revela ser uma alternativa com uma facilidade de implementação muito maior. Por todas estas razões o FHIR é indubitavelmente a escolha mais adequada, pelo que será a tecnologia usada pela solução.

Tabela 6 - Análise Comparativa: FHIR e C-CDA

Critério	FHIR	C-CDA
Pedido de informação filtrada específica	✓	×
Linguagens *markup* compatíveis	XML, JSON, HTTP e OAuth	XML
Grande comunidade de desenvolvedores	✓	×
Tipo de Solução	API	Padrão de *markup* XML
Facilidade na implementação	✓	×

4 Análise

O presente capítulo tem o propósito de estudar toda a envolvente com implicações no desenvolvimento do projeto.

Começa por detalhar o cenário no qual o desenvolvimento vai ser incluído, sendo feita uma análise técnica e arquitetural a este sistema. De seguida, é feita uma análise ao domínio do problema, onde são abordados conceitos de negócio relevantes e é estudado o Modelo de Domínio. Por último, é feito o levantamento de requisitos, tendo em conta a literatura revista no âmbito dos capítulos Contextualização e Estado da Arte e de acordo com os interesses estratégicos da organização onde o autor é colaborador.

4.1 Cenário Viewer

Apesar do projeto visar uma solução interoperável e preparada para ser utilizada por diferentes softwares, no cenário atual será integrada na aplicação hospitalar *Viewer*.

O *Viewer* é uma aplicação com uma estrutura microaplicacional, permitindo a sua comercialização por funcionalidade e por peças de software. O paradigma arquitetural associado possibilita assim montar uma instância da aplicação focada unicamente nas funcionalidades do CPOE.

Desta forma, a aplicação *Viewer* consumirá o MVP do CPOE, que terá exclusivamente funcionalidades associadas ao projeto *WoW*.

Para este efeito, na próxima subsecção é feita uma análise arquitetural e técnica ao *Viewer*.

4.1.1 Aplicação Viewer

O Viewer é uma aplicação *web* hospitalar que, à data da escrita deste documento, se encontra numa fase de desenvolvimento. Apesar de se propor a responder às várias necessidades hospitalares, atualmente apresenta apenas funcionalidades a serem usadas pelos profissionais de saúde. Por estas razões, o Viewer não foi incluído no estudo feito na Secção 3.1.

De forma simplificada e direcionada para o âmbito e fluxo do desenvolvimento do projeto CPOE, identificam-se as seguintes peças de software do Viewer (Figura 11):

- **Cockpit** – Representa a peça de *frontend* que orquestra e encapsula os *widgets*. Permite diferentes configurações de ecrã com diferentes *widgets*, disposições, tamanhos e fluxos de negócio. Além disto, inclui um módulo de autenticação para o profissional de

saúde se ligar, assim como um menu de utilizador, um menu de contexto profissional, uma barra de pesquisa e um centro de notificações globais.

- **Widgets** – Cada peça deste tipo representa um *microfrontend*. É responsável por representar visualmente um determinado cenário através de componentes de interface gráfica. Comunica unicamente com peças do tipo *middleware*, através da tecnologia *GraphQL*, e não tem conhecimento do domínio de negócio.
- **Middlewares** – Trata-se de um tipo de peça que serve como intermediário entre a camada de negócio e os *widgets*. É responsável por responder a pedidos de leitura e escrita de um ou mais *widgets*, gerindo esta comunicação através do *GraphQL*. Para isso, comunica com a camada de negócio – uma peça do tipo *composite* – através de pedidos *gRPC*. Recebe e trata linguagem FHIR dos *composites*, mapeando-a em cada caso para uma estrutura mais simples e direta que envia para o *widget*, numa dinâmica Single *Reponsibility Principle (SRP).*
- **Composites** – Correspondem a um tipo de peça onde é trabalhado o negócio. Comunicam com *microservices*, através de *gRPC*, enviando-lhes instruções de pedidos a realizar à base de dados. A resposta é devolvida ao *middleware*.
- **Microservices** – É um tipo de peça que realiza pedidos *CRUD* à base de dados. Recebe as instruções necessárias do *composite*, como o tipo de operação, o/s recurso/s FHIR envolvido/s e os filtros a aplicar.

  A base de dados está dividida em coleções e entidades, que representam sempre um recurso FHIR.

  A resposta é devolvida ao *composite*.

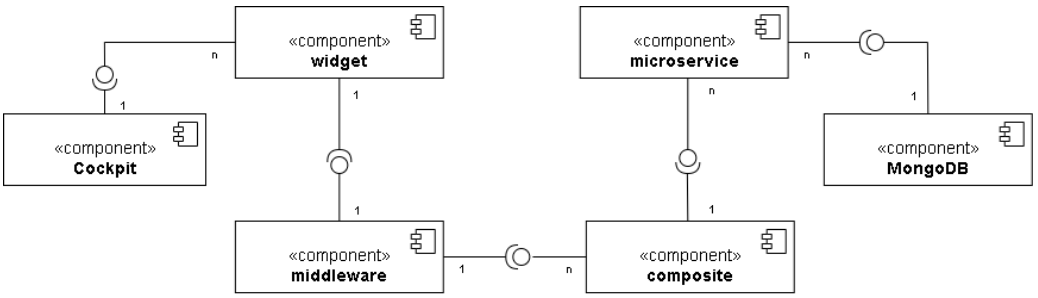

Figura 11 - Representação simplificada da comunicação e tipo de peças de software para novos desenvolvimentos no Viewer

Todas estas peças fazem parte do processo de desenvolvimento de novas funcionalidades para aplicação *Viewer*. A separação de responsabilidades por estas microaplicações é prática indispensável na metodologia da equipa. No entanto, apesar de as bases e os princípios arquiteturais não deverem ser alterados, não existe qualquer impedimento técnico para que, em novos desenvolvimentos, o processo seja diferente ou tenha alterações na arquitetura.

4.2 Domínio e Requisitos

Seguindo boas práticas de engenharia de software e tendo em vista o desenvolvimento de uma solução MVP coerente com as expetativas, ao longo desta secção é estudado o domínio e são definidos os requisitos funcionais e não funcionais da solução.

Visto que o projeto do autor é focado no desenvolvimento do software *WoW Services*, poderá ser pertinente ao leitor recordar a arquitetura global do projeto *WoW,* documentada na Subsecção 2.3.4 e, mais especificamente, representada na Figura 5.

Para a nova solução, são necessárias funcionalidades relacionadas com pedidos médicos, tal como em qualquer CPOE. Estes pedidos estão, concretamente neste MVP, relacionados com o paciente e os dispositivos de monitorização de sinais vitais.

4.2.1 Modelo de Domínio

O modelo de domínio é um mecanismo para documentar e definir entidades e conceitos de negócio que são identificados durante a análise de requisitos. Permite a definição única de conceitos e suas relações, de forma compreensível e acessível para todos os envolvidos[55].

No contexto do projeto e para a elaboração deste modelo, definem-se os seguintes conceitos de domínio:

- **Paciente –** Representa o indivíduo ao qual são realizadas as monitorizações e atos médicos.
- **Profissional de saúde –** Representa o ator do sistema, que fará consultas de dados e ordenará intervenções. Pode ser um médico ou enfermeiro.
- **Sinal Vital –** Representa o alvo da monitorização. Inclui frequência cardíaca, temperatura corporal, frequência respiratória, SpO2 e pressão arterial.
- **Observação** – Representa o conjunto de dados recolhidos, através da monitorização, de um determinado sinal vital, num determinado momento.
- **Intervenção –** Representa uma requisição feita por um profissional de saúde. Esta origina regras de recorrência para monitorizações ou atos médicos.
- **Monitorização –** Representa a medição feita pelos sensores, a um ou mais sinais vitais.
- **Recorrência –** Representa uma regra que define periodicidades para a realização de um ato médico ou monitorização.
- **Ato médico –** Representa determinada ação física, realizada pelo profissional de saúde, relacionada com o paciente e os sensores. Poderá, por exemplo, ser a colocação ou remoção de um sensor, a troca de um dispositivo ou uma limpeza na zona do corpo do paciente onde este é colocado.

[55] Retirado em agosto de 2022 de https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/create_a_domain_model.html

- **Smartbox –** Representa o dispositivo que gere um conjunto de sensores e que pode, em cada momento, estar associado a um e só um paciente.
- **Sensor –** Representa um dispositivo de monitorização associado a uma *smartbox*. É responsável por monitorizar um determinado sinal vital.

Tendo os conceitos de domínio descritos, é fundamental a elaboração do modelo de domínio. Com recuso à notação Unified Modeling Language[56] (UML), apresenta-se o modelo de domínio na Figura 12.

Este modelo retrata as relações entre os vários conceitos, o que inclui as entidades Profissional de Saúde e Paciente, que representam utilizadores do sistema e dos equipamentos de monitorização, respetivamente.

As entidades Monitorização, Intervenção, Recorrência, Ato Médico, Observação, Sinal Vital, *Smartbox* e Sensor completam o ecossistema. Estas entidades conceptuais, permitem todo o processo de realização e consulta dos sinais vitais do paciente. Adicionalmente, têm também a responsabilidade da recolha e disponibilização de dados monitorizados.

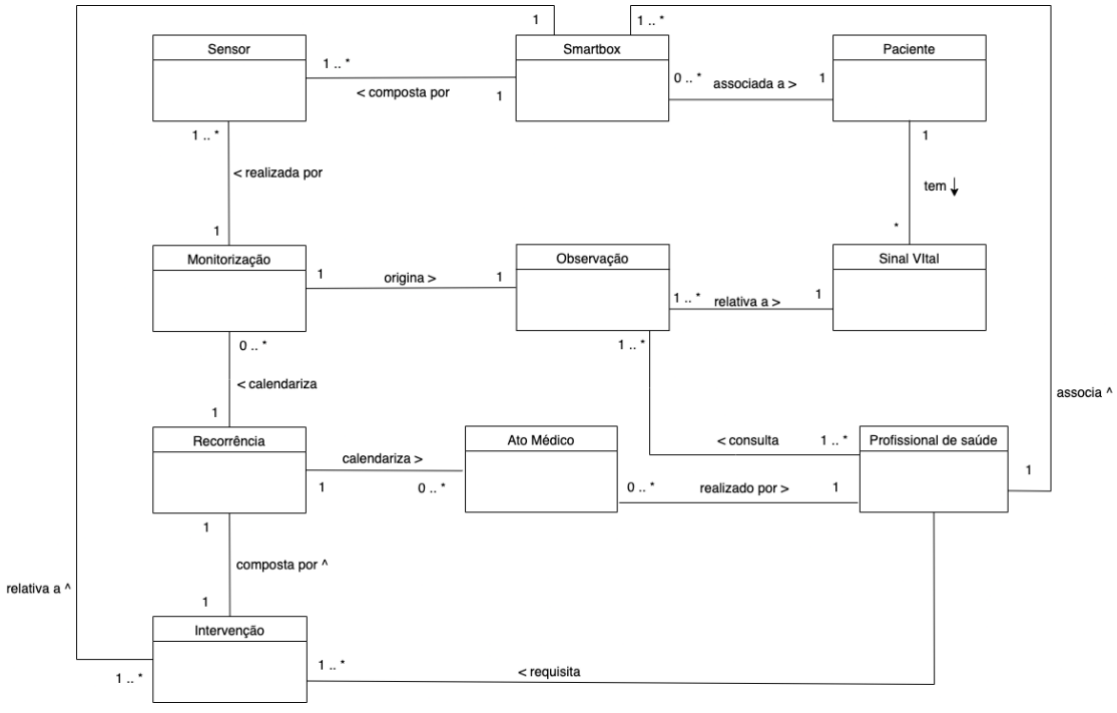

Figura 12 - Modelo de Domínio

[56] Retirado em agosto de 2022 de https://support.microsoft.com/pt-pt/office/diagramas-uml-no-visio-ca4e3ae9-d413-4c94-8a7a-38dac30cbed6

4.2.2 Requisitos

Para desenvolver um sistema completo de funcionalidades, este tem de cumprir dois tipos de requisitos (Danylenko and Löwe, 2012) :

- **Requisitos funcionais (RF) –** requisitos que definem o que é suposto o sistema fazer.
- **Requisitos não funcionais (RNF) –** requisitos que especificam como é que o sistema se deve comportar.

4.2.2.1 Requisitos funcionais

Os requisitos funcionais são funcionalidades e caraterísticas do produto que são implementadas de forma a responder às necessidades dos clientes. Geralmente, descrevem o comportamento do sistema sob determinadas condições e devem ser claros para os *stakeholders* e equipa de desenvolvimento[57].

Os requisitos funcionais identificados para a solução foram:

- **RF1** - O sistema deve ser capaz de associar novas *smartboxes* com sensores ao paciente.
- **RF2** - O sistema deve ser capaz de listar as *smartboxes* e sensores associados ao paciente.
- **RF3** – O sistema deve permitir a adição de novas intervenções, com a possibilidade de definir uma recorrência para estas.
- **RF4** - O sistema deve ser capaz de listar o histórico de dados recolhidos nas monitorizações ao paciente.

A Figura 13 representa a interação entre os utilizadores do sistema e as funcionalidades a ser desenvolvidas. O utilizador será o profissional de saúde, que poderá ser médico ou enfermeiro, no entanto não há impacto nesta diferenciação, visto que todas as funcionalidades serão disponibilizadas para ambos, sem restrições.

De uma forma breve, descrevem-se os casos de uso:

- **Associar Dispositivos (RF1)** – O profissional de saúde navega até ao *widget* de associação de dispositivos e o sistema apresenta todos as *smartboxes* (e respetivos sensores) disponíveis para associação imediata ao paciente. O profissional de saúde seleciona os dispositivos pretendidos e confirma os dados, enviando para o sistema. O sistema informa do sucesso da operação.
- **Consultar Dispositivos (RF2)** - O profissional de saúde navega até ao *widget* de listagem de dispositivos e o sistema apresenta todos as *smartboxes* (e respetivos sensores) disponíveis associados ao paciente.
- **Adicionar Intervenções (RF3)** - O profissional de saúde seleciona uma *smartbox* do paciente. O sistema redireciona-o para o *widget* de intervenções. O sistema solicita as

[57] Retirado em fevereiro de 2022 de https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/

condições nas quais deve ser definida a intervenção – recorrência/ periodicidade da intervenção, executante (pode ser o dispositivo, no caso de monitorização, ou profissional de saúde, no caso de Ato Médico). O profissional de saúde introduz e confirma os dados solicitados. O sistema guarda a nova Intervenção e informa o utilizador do sucesso da operação.

- **Consultar Observações (RF4)** - O profissional de saúde navega até ao *widget* de consulta dos resultados de monitorização do paciente. O sistema apresenta os resultados para cada sinal vital que sofreu monitorização.

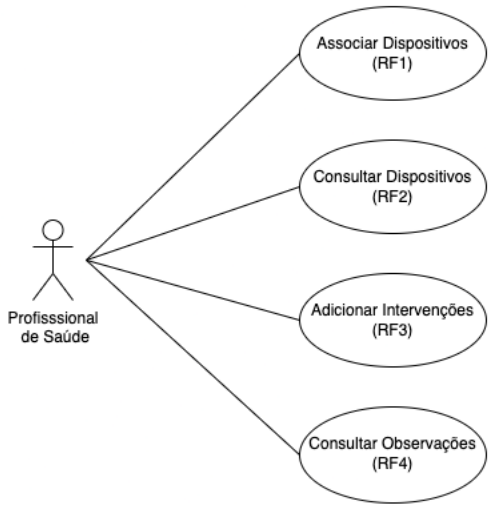

Figura 13 - Diagrama de Casos de Uso

Definidos os casos de uso, segue-se o estudo do comportamento do estado de usos Associar Dispositivos (RF1), conforme representado no Diagrama de sequência de sistema da Figura 14, em notação UML:

Ator Principal:

- Profissional de saúde (Médico ou Enfermeiro).

Partes interessadas e seus interesses:

- Profissional de Saúde – tem como interesse ver os dispositivos disponíveis para associação, de forma a associar ao seu paciente, prestando os melhores cuidados ao mesmo.
- Paciente – tem como interesse que lhe sejam associados dispositivos de monitorização, de forma a acompanhar o seu estado de saúde através da medição dos seus sinais vitais.

Pré-Condições:

- O sistema deve conter dispositivos disponíveis para associação ao paciente.
- O profissional de saúde deve estar autenticado na aplicação *Viewer*, no contexto de um paciente.

Fluxo principal:

1. O profissional de saúde acede ao *widget* de Associação de Dispositivos.
2. O Sistema apresenta todos os dispositivos (*smartboxes* e respetivos sensores) disponíveis para a associação.
3. O profissional de saúde filtra os resultados por nome da *smartbox*.
4. O sistema devolve os resultados filtrados de dispositivos.
5. O profissional de saúde seleciona as *smartboxes* pretendidas e envia os dados para o sistema.
6. O sistema processa os dados e informa o profissional de saúde do sucesso da operação.

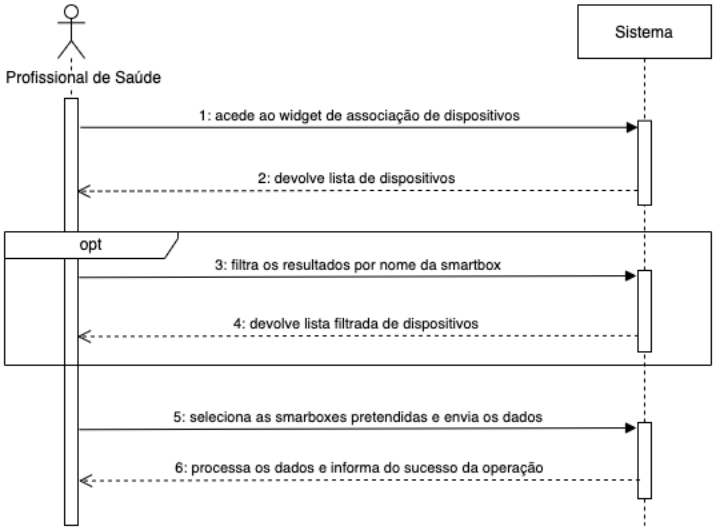

Figura 14 - Diagrama de Sequência do Caso de Uso: Associar Dispositivos

4.2.2.2 Requisitos não funcionais

Os requisitos não funcionais, ou atributos de qualidade[58], são requisitos não especificados pelos *stakeholders*, mas que estão previstos para a execução das funcionalidades do sistema (Rahman *et al.*, 2019). Estes são necessários para garantir a qualidade do software e são também muitas vezes responsáveis pela satisfação dos clientes e pela otimização do desempenho da solução. Apresentam frequentemente um nível de criticidade superior aos requisitos funcionais, dado que o seu incumprimento pode implicar falhas transversais a todo o sistema (Alencar *et al.*, 2019).

[58] Retirado em fevereiro de 2022 de https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/

Os requisitos não funcionais identificados para a solução foram:

- **RNF1 –** O sistema deve ser capaz de listar resultados de monitorizações provenientes de diferentes fontes de informação.
- **RNF2 –** O sistema deve notificar o utilizador que a associação de dispositivos e os pedidos de intervenção foram feitos corretamente.
- **RNF3** - O sistema deve respeitar políticas de privacidade, de acordo com o Regulamento Geral sobre a Proteção de Dados[59] (RGPD) em vigor.
- **RNF4** – O sistema deve iniciar no contexto de um paciente.
- **RNF5 –** O sistema deve apresentar a informação dos dispositivos e monitorizações de forma clara e facilmente acessível.
- **RNF6 –** O sistema deve ser capaz de armazenar resultados de monitorização e permitir novas intervenções, sem que eventuais falhas levem a perdas de informação.
- **RNF7 –** O sistema deve garantir a integridade das intervenções quando estas mudam de estado e são enviadas para a aprovação de terceiros.
- **RNF8 –** O sistema deve permitir 1000 pedidos de consulta de dispositivos em simultâneo, sem que implique atrasos ou perdas de informação.
- **RNF9 –** O sistema deve ser implementado de forma a permitir facilmente futuras melhorias e desenvolvimento de novas funcionalidades.
- **RNF10 –** O sistema deve permitir comunicação com uma *gateway* através de interoperabilidade, sem afetar outros sistemas.
- **RNF11 –** O sistema deve ter uma arquitetura baseada em microaplicações.
- **RNF12 –** O sistema deve usar a biblioteca *React* para o desenvolvimento da camada de *frontend*.
- **RNF13 –** O sistema deve suportar múltiplos navegadores *web*, incluindo Microsoft Edge, Mozilla Firefox, Opera, Safari, Google Chrome e outros baseados neste último.
- **RNF14 –** O sistema deve usar FHIR como *standard* para troca de dados de saúde.

Dadas as caraterísticas singulares deste tipo de requisitos, o autor decidiu usar o modelo FURPS+[60], que permite identificar e categorizar RNF.

Através da sigla FURPS+, são identificadas as categorias que esta metodologia preconiza[60]:

- **F**uncionalidade – representa as funcionalidades principais do produto, de acordo com o domínio de negócio da solução a desenvolver. Outras funcionalidades mais técnicas também podem ser integradas nesta categoria, o que inclui, entre outros, auditoria, licenciamento, localização, suporte online, segurança e gestão de sistema.
- **U**sabilidade – identifica requisitos relacionados com a interface do utilizador, incluindo requisitos de acessibilidade, estética e consistência.

[59] Disponível em fevereiro de 2022 em https://www.portugal.gov.pt/pt/gc22/politica-de-privacidade
[60] Retirado em fevereiro de 2022 de https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/

- Confiabilidade (***R**eliability*) – inclui requisitos associados à disponibilidade, exatidão e capacidade de recuperação de dados em caso de falhas do sistema.
- Desempenho (***P**erformance*) – representa requisitos envolvidos em transferência de informação através do sistema, tempos de resposta (relacionados também com *usabilidade*), tempos de recuperação e tempos de arranque.
- **S**uportabilidade – engloba requisitos como testabilidade, adaptabilidade, manutenibilidade, compatibilidade, configurabilidade, escalabilidade, entre outros.
- **+**
    - Restrições de desenho – limitam o desenho do sistema, implicando o uso, ou impossibilidade de uso, de certos recursos.
    - Restrições de implementação – limitam o uso de *standards*, plataformas ou linguagens de programação, na fase de implementação.
    - Restrições de interface – requerimentos que envolvem comunicação com sistemas externos, ou seja, interoperabilidade.
    - Restrições físicas – relacionadas com equipamentos de *hardware* e as suas caraterísticas físicas.

De acordo com estas categorias, a Tabela 7 identifica a natureza de cada um dos requisitos não funcionais previamente identificados.

Tabela 7 - Categorização segundo FURPS+ dos requisitos não funcionais do sistema

Categoria	**Requerimentos**
Funcionalidade	RNF1, RNF2, RNF3, RNF4
Usabilidade	RNF5
Confiabilidade	RNF6, RNF7
Desempenho	RNF8
Suportabilidade	RNF9, RNF13
+	RNF10, RNF11, RNF12, RNF14

Os requisitos não funcionais da categoria representada pelo símbolo “+” na Tabela 7, estão subcategorizados na Tabela 8.

Tabela 8 – Subcategorização dos requisitos do sistema da categoria "+" de FURPS+

Subcategoria	Requerimentos
Restrições de desenho	RNF11
Restrições de implementação	RNF12, RNF14
Restrições de interface	RNF10
Restrições físicas	Não identificados

5 Desenho da Solução

O planeamento da solução a implementar é fundamental. Realizar este trabalho *à priori* torna o processo mais eficiente, visto que antecipa diversos problemas, evitando retrocessos na fase implementação.

O desenho da solução é a fase de planeamento onde são produzidos artefactos de Engenharia de software, respondendo aos problemas arquiteturais da solução.

Ao longo deste capítulo são feitas análises arquiteturais através de várias vistas, com diferentes granularidades. Para cada vista são definidas algumas alternativas, sendo selecionada a mais vantajosa.

5.1 Arquitetura

Nesta secção são estudadas algumas arquiteturas com potencial para servir a solução a desenvolver.

O modelo de vista 4+1 representa uma forma de expor a arquitetura de um software através de 5 vistas diferentes. A quinta vista, correspondente ao "+1" no nome, ilustra e valida todas as outras, que têm a função de capturar as decisões de conceção.

Este modelo separa responsabilidades através das diferentes vistas. Estas abordam questões distintas e dividem-se entre (Kruchten, 1995):

- **Vista Lógica** - Vista dedicada aos requisitos funcionais, ou seja, às funcionalidades que servem o utilizador. A estrutura lógica do sistema é assim representada através de uma organização em larga escala, através de classes e suas relações lógicas, retiradas da interpretação do domínio do problema.
- **Vista de Processos -** Vista dedicada às partes dinâmicas do sistema. Explica os processos e as suas comunicações, focando-se no comportamento do sistema.
- **Vista Física -** Vista que mapeia a arquitetura do sistema, através de nós, criando relações entre o hardware e software do sistema. Foca a organização da estrutura física em que o software está implantado.
- **Vista de Desenvolvimento –** Vista que organiza os módulos de software do sistema em ambiente de desenvolvimento.
- **Cenários -** Vista que ajuda no estudo e criação conceptual de elementos arquiteturais durante a fase de desenho, servindo também para validar e ilustrar a mesma.

A Vista Lógica está ilustrada na Secção 4.2.1, através do Modelo de Domínio da Figura 12, ao passo que os Cenários estão retratados na Figura 13 da Secção 4.2.2, através do diagrama de Casos de Uso. Por esta razão, não são estudadas neste capítulo.

Nas próximas secções são estudadas as Vistas de Processos, Desenvolvimento e Física. Para as duas últimas são avaliadas e comparadas diferentes alternativas, enriquecendo e sustentando as decisões arquiteturais. De acordo com as escolhas feitas, são apresentadas as vistas de processos.

5.1.1 Vista de Desenvolvimento

Através desta vista representam-se os vários componentes do sistema e as suas interações, analisando a arquitetura de software. Neste âmbito, as alternativas apresentadas procuram diferir na distribuição de responsabilidades e nas peças de software envolvidas.

Alternativa A

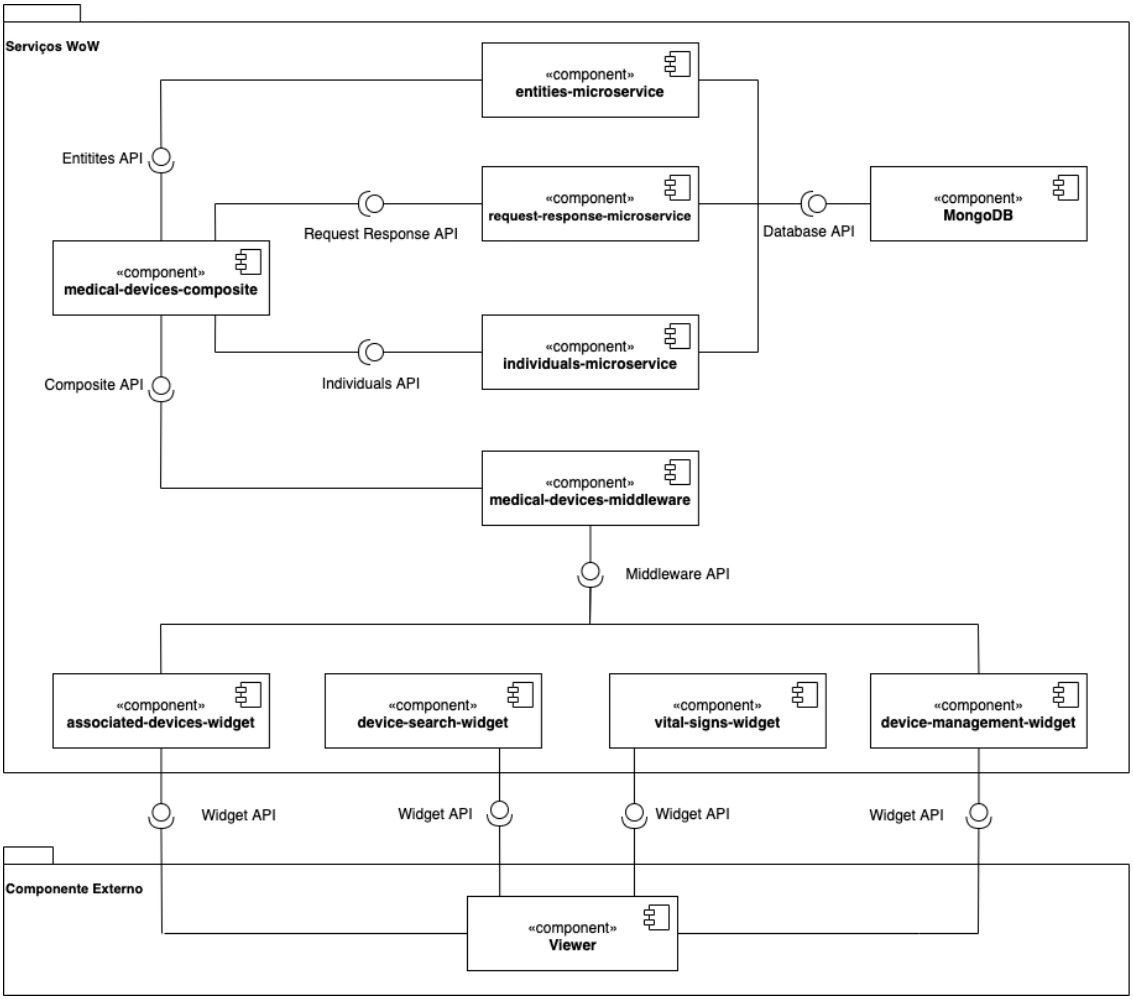

Figura 15 - Diagrama de Componentes - Alternativa A

O diagrama da Figura 15 representa uma alternativa que reparte as responsabilidades do sistema nos seguintes componentes:

- **Viewer** – Este componente é o único componente externo à solução. O componente representa a aplicação hospitalar do mesmo nome.
  Uma das funções deste componente é ser um orquestrador de *microfrontends*. Assim, seja no contexto deste projeto, ou em qualquer desenvolvimento, o *Viewer* consome *microfrontends*, organizando-os e exibindo-os na interface gráfica da aplicação.
- **associated-devices-widget** – Esta peça *widget* é um *microfrontend*, ou seja, constitui uma interface gráfica para o utilizador. Neste caso, é a camada de *frontend* relativa ao RF2 – Consultar Dispositivos.
- **device-search-widget –** Tem o mesmo propósito do componente acima, mas com responsabilidades relativas ao RF1 – Associar Dispositivos.
- **vital-signs-widget** – É também uma peça *widget*, com responsabilidades relativas ao RF4 – Consultar Observações.
- **device-management-widget** – Última das 4 peças *widget*. Esta tem responsabilidades relativas ao RF3 – Adicionar intervenções.
- **medical-devices-middleware** – Este componente tem a função de mediar a comunicação entre as peças de *frontend* (*widgets*) e a peça de camada de negócio (composite).
  Define contratos de dados, filtra e mapeia informação, proveniente do *composite* e com a estrutura de recursos *FHIR*, da forma mais direta possível para os *widgets*, permitindo que estes não conheçam qualquer lógica de negócio.
- **medical-devices-composite** – Componente de camada de negócio responsável por construir e chamar estruturas de *queries* aos vários *microservices*. Mantém contratos de dados com o *medical-devices-middleware* e com os *microservices*, de forma a trocar informação.
- **individuals-microservice** – Peça do tipo *microservice*, tal como outras com esta terminação, pelo que realiza operações *Create Read Update Delete* (CRUD*)* à base de dados, de acordo com a informação que recebe do *composite*, e devolve-lhes a resposta.
  No caso, este microservice faz pedidos associados a recursos *FHIR Practitioner* e *PractitionerRole*.
- **request-response-microservice** – Realiza pedidos associados a recursos *FHIR*, como *ServiceRequest*, *DeviceRequest* e *Observation*.
- **entities-microservice** – Realiza pedidos associados ao recurso *FHIR Device*.
- **MongoDB** – Base de dados não relacional, responsável por armazenar toda a informação de acordo com o *standard* FHIR.

Alternativa B

A segunda alternativa, representada na Figura 16, apresenta uma arquitetura com menos componentes. Esta alternativa, apesar de manter uma distribuição de responsabilidades, acaba por causar o aumento das mesmas em algumas peças.

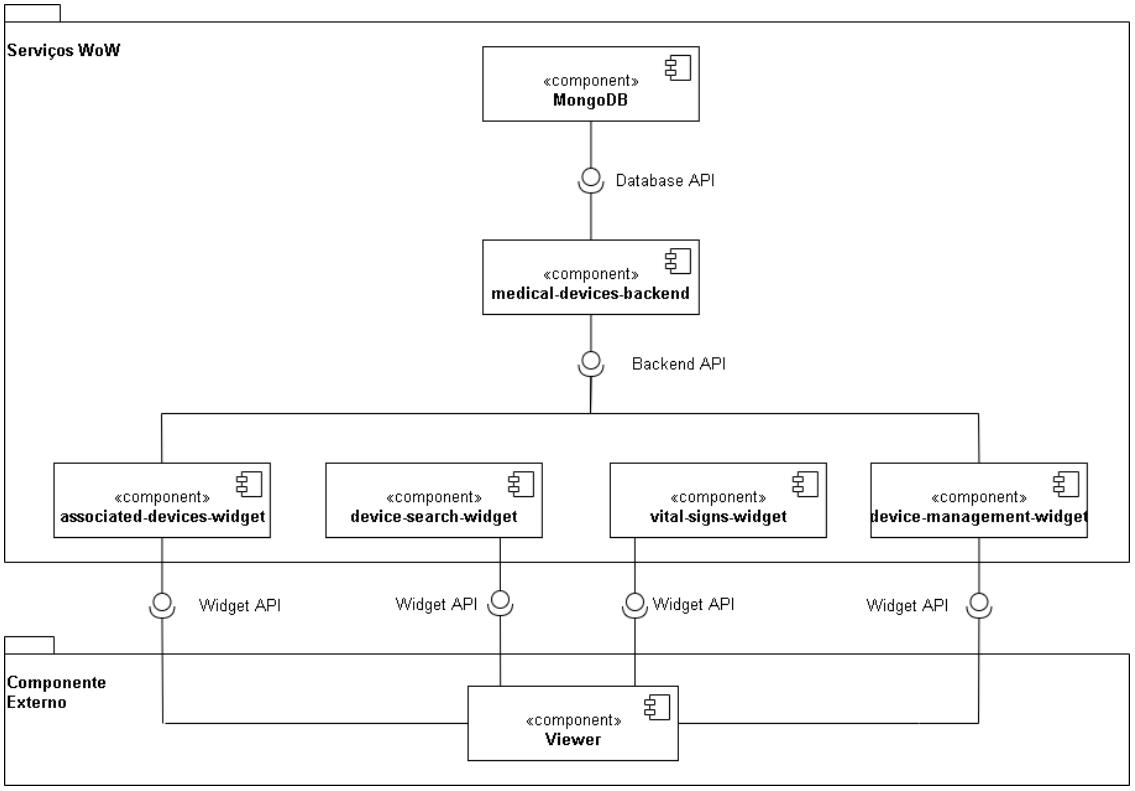

Figura 16 - Diagrama de Componentes - Alternativa B

Todos os componentes deste diagrama, à exceção do medical-devices-backend, estavam já presentes na Alternativa A, pelo que nesta alternativa mantêm as caraterísticas e responsabilidades.

Os *widgets* – interfaces gráficas independentes, representados no diagrama pelos componentes com o sufixo "*widget*" – consomem a peça medical-devices-backend.

O **medical-devices-backend** tem as seguintes caraterísticas:

- Define contratos de dados com as peças *Widget*, trocando informação com estas em conformidade com os mesmos.
- Trata informação proveniente dos *widgets*, para realizar operações na base de dados.
- Constrói e executa pedidos à base de dados.
- Filtra a informação dos resultados das *queries* às bases de dados. Traduz também esta informação de standard FHIR, para uma estrutura mais simples para os *widgets* interpretarem, evitando que estes tenham responsabilidades de negócio.

Em suma, esta peça agrega as caraterísticas e responsabilidades presentes nas peças do tipo *composite*, *middleware* e *microservice*, presentes na Alternativa A.

Alternativa C

A terceira alternativa estudada é, das três, a que apresenta uma vista de desenvolvimento de estrutura mais simples. Como se pode ver na Figura 17, apresenta apenas quatro componentes, novamente com divisão de responsabilidades. No entanto, comparando com as outras alternativas, existe um congestionamento maior das responsabilidades entre os componentes.

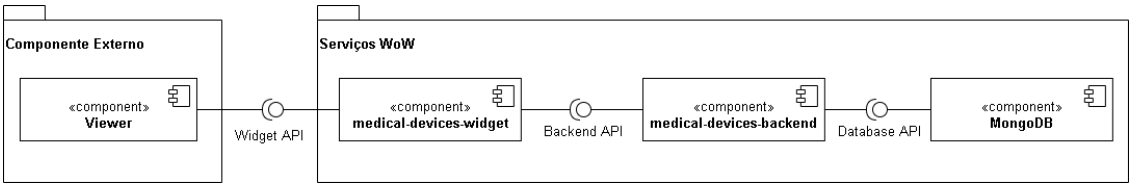

Figura 17 - Diagrama de Componentes - Alternativa C

Tal como acontece com a Alternativa B, as peças deste diagrama presentes nas outras alternativas mantêm as caraterísticas e responsabilidades.

O único componente que diferencia esta alternativa é o **medical-devices-widget.** Nesta proposta esta é a única peça do tipo *widget*. Agrega as responsabilidades de *frontend* relativas a todos os requisitos funcionais (RF1, RF2, RF3 e RF4), apresentado uma única interface gráfica.

Apesar de alterar a experiência do utilizador, implicando navegações dentro do mesmo *widget* para o acesso às várias funcionalidades da solução, não compromete o funcionamento das mesmas.

Seleção de Alternativa

Além das alternativas referidas, foi brevemente considerada a hipótese de uma arquitetura monolítica, ou seja, com apenas uma peça de software. No entanto, esta hipótese foi rapidamente descartada devido ao requisito não funcional RNF11 - O sistema deve ter uma arquitetura baseada em microaplicações.

Através da análise das restantes alternativas, em discussão com colaboradores da *Glintt-HS*, tomou-se a decisão de optar pela **Alternativa A** (Figura 15).

Esta é alternativa com a distribuição de responsabilidades mais adequada, promovendo a boa prática do padrão *Single Responsibility Principle* (SRP) e uma alta coesão.

Por outro lado, esta arquitetura facilita a manutenção do sistema e permite a reusabilidade dos vários componentes. Promove também o baixo acoplamento, dado que a falha de uma peça de software não compromete o correto funcionamento de outras, salvaguardando também casos de falha total do sistema.

5.1.2 Vista Física

Esta vista retrata a topologia das peças de software no ambiente físico em que são inseridas. Adicionalmente, identifica as suas ligações e os protocolos de comunicação envolvidos.

As alternativas para esta vista, estão construídas de acordo com a arquitetura selecionada para a Vista de Desenvolvimento do sistema, analisada na Subsecção 5.1.1.

Alternativa A

A primeira alternativa está representada na Figura 18 através de um diagrama de implantação.

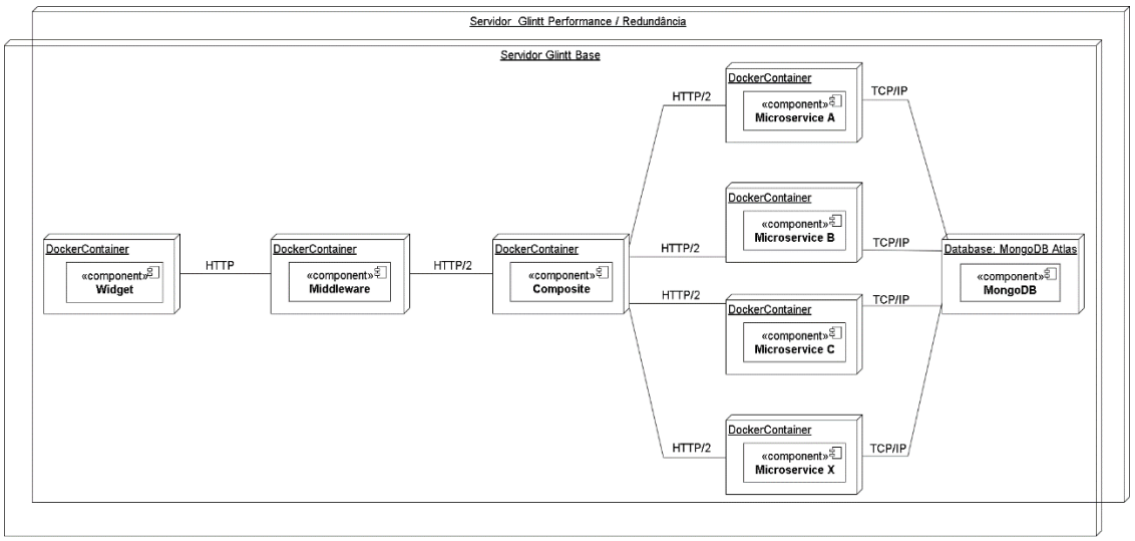

Figura 18 - Diagrama de Implantação - Alternativa A

Note-se que, neste diagrama, para alojar os diferentes componentes do sistema, existem dois servidores distintos, o **Servidor *Glintt* Performance/ Redundância** e o **Servidor *Glintt* Base**, sendo ambos máquinas virtuais.

O **Servidor Glintt Base** é o servidor principal. Este aloja os vários ambientes de execução virtuais, os *Docker Containers*[61], como se pode ver na Figura 18.

O **Servidor Glintt Performance/ Redundância** é um servidor secundário, alojando também os Docker containers. Apesar de ter a mesma função, tem um propósito diferente, graças ao *Docker Swarm*[62].

O *Docker Swarm* faz a implantação das várias peças e escalona os containers entre os dois servidores. Funciona como um orquestrador, executando este escalonamento dos Docker Containers de acordo com a disponibilidade de cada servidor e gerindo a carga de ambos. Assim, a existência do segundo servidor garante um maior desempenho (**performance)** do sistema,

[61] Retirado em outubro de 2022 de https://www.docker.com/resources/what-container/
[62] Retirado em outubro de 2022 de https://www.sumologic.com/glossary/docker-swarm/

através deste escalonamento inteligente. Por outro lado, garante **redundância** / alta disponibilidade, sendo tolerável à falha de um servidor.

Relativamente aos *Docker containers*, como se pode ver na imagem, existe um por peça de software. Este é também um fator que contribui para a alta disponibilidade, dado que a falha de um Container apenas compromete a disponibilidade de uma peça de software.

Em relação aos protocolos de comunicação, as peças *Widget* comunicam com a peça *Middleware* através de *GraphQL*, que utiliza o protocolo *HTTP*. Já a comunicação entre *Middleware – Composite* e *Composite – Microservice* é feita através de *gRPC*, que utiliza o protocolo HTTP/2. Por fim, os *microservices* comunicam com a base de dados *MongoDB* por meio do protocolo *TCP/IP*.

Alternativa B

O diagrama de implantação desta segunda alternativa está representado na Figura 19.

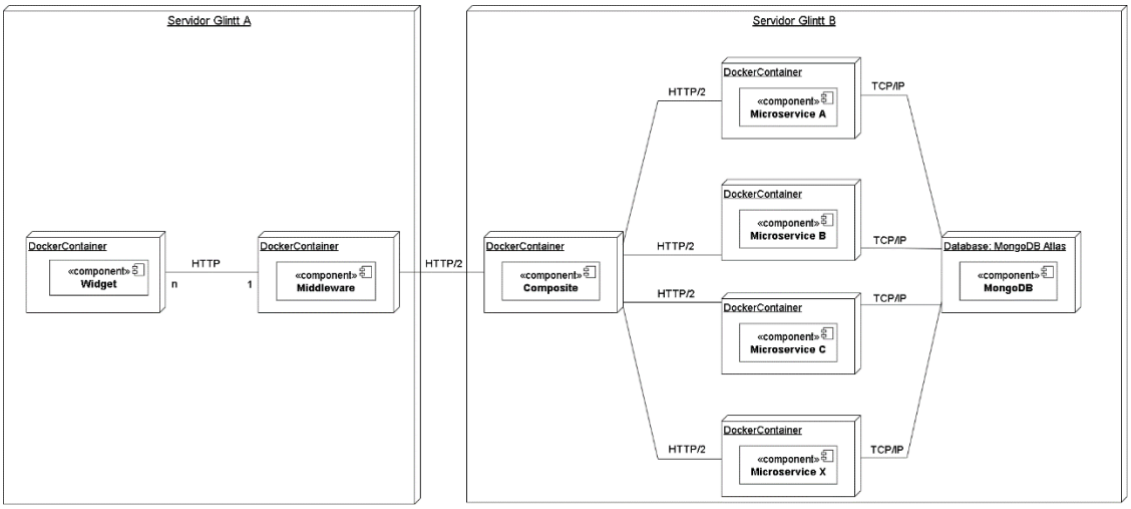

Figura 19 - Diagrama de Implantação - Alternativa B

Relativamente a protocolos de comunicação e à instalação das peças de software em containers, esta proposta não apresenta diferenças em relação à Alternativa A.

No entanto, nesta alternativa não existe nenhum orquestrador a implantar e escalonar os Docker containers pelos servidores. Neste caso, como se pode observar na Figura 19, existem dois servidores distintos, com funções diferentes.

O **Servidor Glintt A** é responsável por alojar os containers das peças *Widget* e da peça *Middleware*, enquanto o **Servidor Glintt B** aloja todos os outros containers.

Embora esta alternativa, na ocorrência de falha de um servidor, não comprometa a disponibilidade de todas as peças do sistema, não apresenta nenhum grau de redundância.

Seleção de Alternativa

Além da Alternativa A e B, foi brevemente considerada uma alternativa onde existiria um servidor para cada Docker Container. No entanto, seria uma alternativa irrealista no que concerne à disponibilização de infraestruturas necessárias para o efeito. Adicionalmente, era uma alternativa que não dava garantias de redundância. Assim, a alternativa foi descartada.

Sendo a alta disponibilidade, redundância e tolerância à falha fatores relevantes para a escolha, a **Alternativa A** foi a selecionada. A utilização do *Docker Swarm* fortalece-a, beneficiando o sistema com o dinamismo de escalonamento dos *Docker containers* entre servidores, resultando num maior desempenho e em alta disponibilidade.

5.1.3 Vista de Processos

De forma a ser compreendido o fluxo que envolve todos os requisitos funcionais, apresenta-se um diagrama de Granularidade grossa, na Figura 20. O diagrama representa os processos entre os serviços *WoW* (software desenvolvido no âmbito deste documento) e os componentes externos.

Neste diagrama encontram-se destacados os requisitos funcionais RF1, RF2, RF3 e RF4, através dos quatro retângulos etiquetados.

O **RF1: Associar Dispositivos** implica uma comunicação entre apenas duas entidades. Os Serviços Wow realizam um pedido à *GateWay* através dos dispositivos disponíveis, isto é, dispositivos não associados a um doente e em estado *active*. A *GateWay* devolve os dispositivos que cumprem estas condições.

Um ou mais dispositivos podem ser então adicionados para associação. Os serviços WoW enviam os dados de associação para a *GateWay*, que a consuma.

O **RF2: Consultar Dispositivos** é relativo aos dispositivos do paciente. A *GateWay* retorna as *smartboxes* e respetivos sensores associados ao paciente.

O **RF3: Adicionar intervenções** permite adicionar, para dispositivos associados ao doente, pedidos de intervenção médica aos profissionais de saúde, como monitorizações, aos sensores, ou atos médicos. Neste caso, analisou-se a hipótese mais complexa, onde a intervenção é um pedido de monitorização, de forma a responder ao caso de uso.

Os Serviços WoW fazem o pedido à *GateWay*, que por sua vez comunica com a *smartbox*. Esta comunica com os sensores, enviando informação que espoleta uma ou mais monitorizações, dependendo da recorrência pedida.

Para a devolução dos valores obtidos na monitorização, os sensores enviam esta informação à *smartbox*. Esta entidade envia a informação para a *GateWay*, que a armazena.

O **RF4: Consultar Observações** permite a consulta dos resultados dos sinais vitais monitorizados. Os Serviços WoW, através da escolha do dispositivo ao qual se irá fazer a leitura, fazem o pedido de informações à *Gateway*, que devolve os resultados das monitorizações.

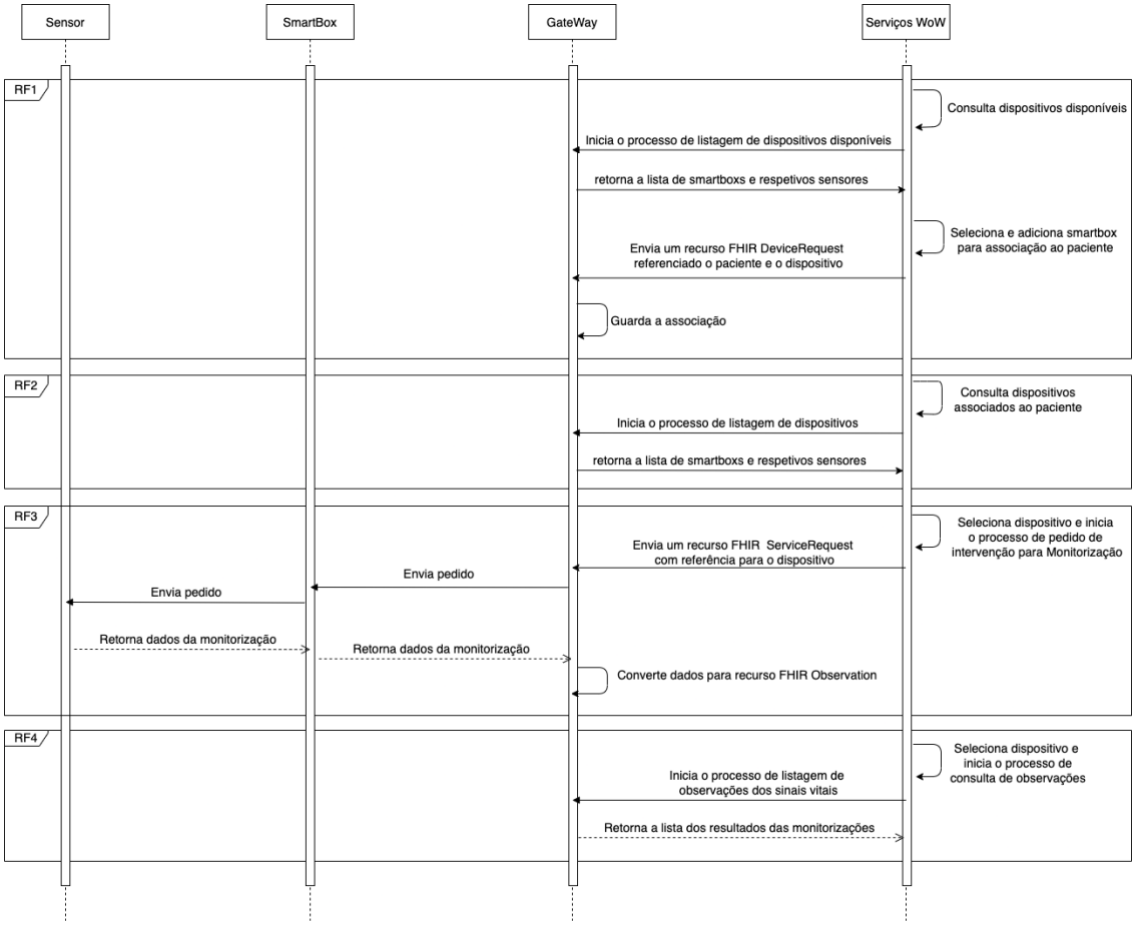

Figura 20 - Diagrama de Sequência Geral de Granularidade Grossa

Nas próximas subsecções encontram-se vistas de uma granularidade mais fina, onde se estuda o comportamento e processos entre componentes dos serviços WOW, desenvolvidos no âmbito do projeto do documento.

Note-se que, neste sentido, a alternativa selecionada para a vista de desenvolvimento, na Secção 5.1.1 , tem impacto direto nas próximas vistas.

5.1.3.1 Caso de Uso: Associar Dispositivos (RF1)

Na Figura 21 apresenta-se o diagrama de sequência relativo ao Caso de Uso: Associar Dispositivos.

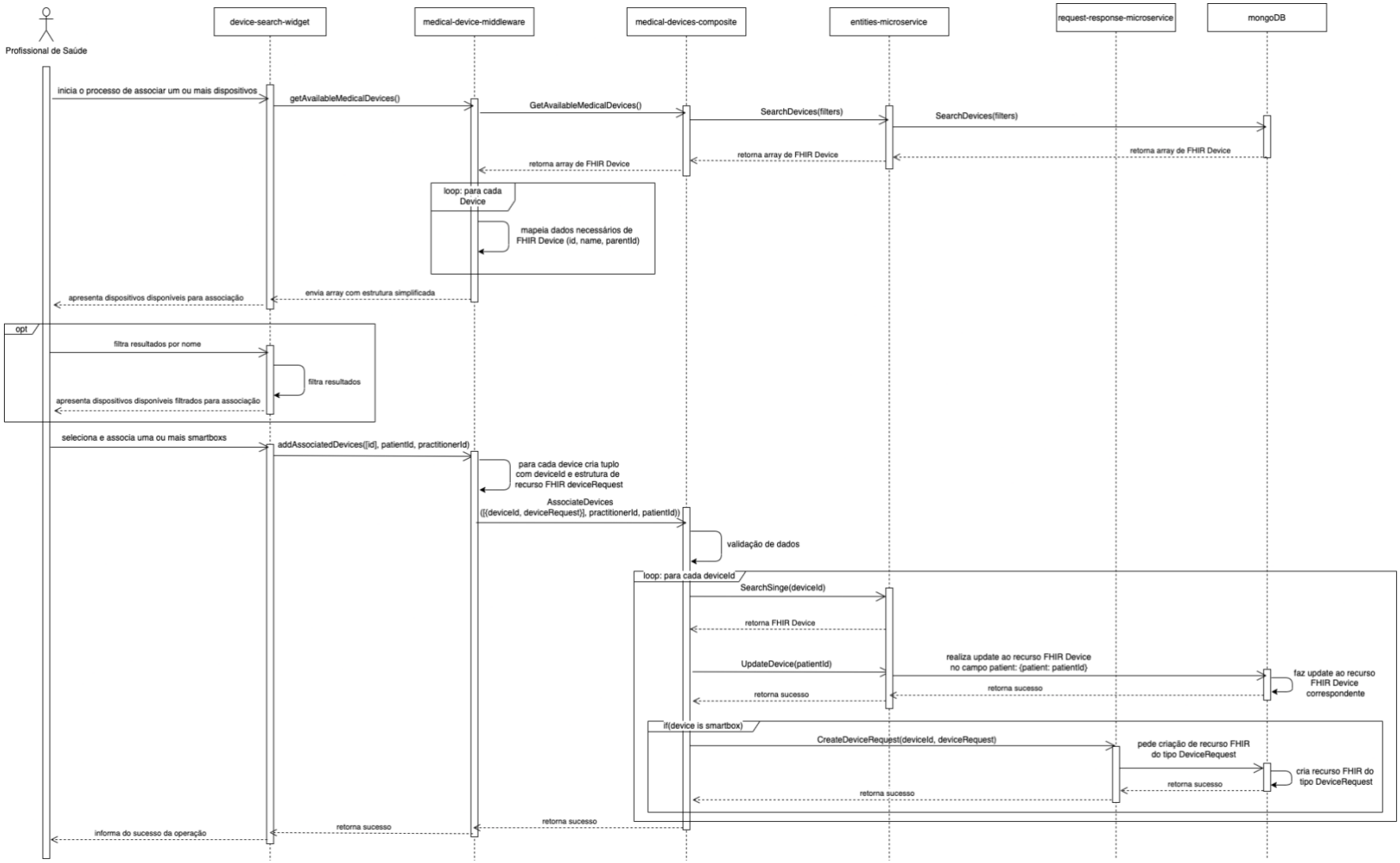

Figura 21 - Diagrama de Sequência: Associar Dispositivos

Analisando o diagrama da Figura 21, é possível verificar que o *devices-search-widgets* é a interface do utilizador com que este interage para procurar e associar dispositivos disponíveis para a monitorização a um paciente.

Este *widget* começa por fazer um pedido, de forma a **receber a informação de todos os dispositivos que se encontram livres**, ou seja, sem associação a qualquer paciente. Este pedido é feito ao *middleware* que o comunica ao *composite*, que por sua vez constrói o pedido.

O pedido visa procurar na base de dados por recursos *FHIR* com as seguintes condições:

1. O recurso deve ser FHIR *Device*[63] (recurso que representa dispositivo *smartbox* ou sensor).
2. O campo status do recurso deve estar preenchido com a *string* "*active*".
3. O campo *patient*, que representa uma referência para o recurso *FHIR Patient*, não deve estar preenchido, indicador de que o dispositivo não está associado a nenhum paciente.

O pedido é enviado para o *entities-microservice*, que realiza a *query* à base de dados. Esta responde com o conjunto de recursos *FHIR* existentes, que obedecem às condições especificadas. A informação segue o percurso inverso, sendo esta informação tratada na peça *middleware*.

Para compreender este tratamento de dados é necessário ter em conta a informação da qual o *widget* necessita para mostrar ao utilizador cada dispositivo disponível:

- **name** – Designação do dispositivo. Representado no campo "*name*" do FHIR Device.
- **deviceId** – Identificador único do dispositivo[64]. É gerado pela base de dados.
- **parentId** – Hierarquia do dispositivo, isto é, referência para o dispositivo pai. No caso de o dispositivo ser uma *smartbox*, o campo nunca está preenchido. Sendo um sensor, é sempre uma referência para a *smartbox* a que está associado, através do *deviceId* da mesma.
  É representado no campo "*parent*" do *FHIR Device*.

Dado que esta é uma fração da informação presente em cada recurso do tipo *FHIR Device*, o tratamento de dados do *middleware* consiste precisamente em filtrar e mapear cada um destes recursos. Assim, é criada uma estrutura que apresenta apenas o *name*, device*Id* e *parentId*, descartando toda a restante informação, irrelevante para o utilizador e, consequentemente, para o *widget*.

[63] Disponível em setembro de 2022 em https://www.hl7.org/fhir/device.html

[64] Embora o device*Id* não seja necessário para o utilizador consultar os dispositivos, desempenha um papel fundamental no posterior processo de associação, sendo indispensável a obtenção desta informação.

O *widget* obtém esta informação, apresentando ao utilizador uma lista de *smartboxes* selecionáveis, com os respetivos sensores.

O ecrã permite ainda uma filtragem opcional por nome de *smartboxes*. Para o efeito, são filtrados os dados recebidos pelo *middleware*, através do *name*.

Posteriormente, o utilizador **seleciona as *smartboxes* pretendidas** (o que inclui forçosamente os sensores correspondentes) e **prossegue com a associação**. O *widget* envia assim ao *middleware* vários *deviceId* (relativos a cada dispositivo a associar), um identificador do paciente (*patientId*) e do profissional de saúde (*practitionerId*).

O recurso FHIR utilizado para representar uma associação de dispositivo *smartbox* é o *DeviceRequest*[65]. Desta forma, o *middleware* mapeia os dados recebidos para uma estrutura que obedece ao standard FHIR deste recurso.

```
{
   authoredOn: date,
   reference: deviceId
   patient: patientId
   requester:  practitionerId,
   status: 'ACTIVE'
}
```

Figura 22 - Estrutura com campos de FHIR DeviceRequest (pseudocódigo)

Assim, para cada *deviceId* recebido (referente a uma *smartbox*), é criada uma estrutura como a da Figura 22:

- **status** – Indica o estado da associação. É preenchido com a *string* "*active*".
- **codeReference** – Indica qual é a *smartbox* desta associação, através de uma referência. É preenchido com o *deviceId* recebido do *widget*.
- **subject** – Indica o paciente desta associação, através de referência. É preenchido com o *patientId* recebido do *widget*.
- **requester** - Indica o profissional de saúde que pediu esta associação, através de referência. É preenchido com o *practitionerId* recebido do *widget*.
- **authoredOn** – Preenchido com a data e hora do momento, calculada diretamente no *middleware*.

Para os *deviceId* de dispositivos sensores, a estrutura acima não é criada, dado não se aplicar a criação do recurso FHIR *DeviceRequest* para estes dispositivos.

Após esta operação, o *middleware* comunica com o *composite*, enviado-lhe tuplos com *deviceId* e a estrutura construída correspondente, além do *patientId* e *practitionerId*.

[65] Disponível em setembro de 2022 em https://www.hl7.org/fhir/devicerequest.html

Para cada tuplo, o *composite* realiza os seguintes pedidos, por esta ordem:

1. Pedido ao *entities-microservice* para obter o recurso completo *FHIR Device*, através do *deviceId*.
2. Pedido ao *entities-microservice* para alterar o campo "*patient*" do recurso *FHIR Device* obtido no passo anterior. O *patientId* recebido é injetado neste campo, garantido assim que o dispositivo não volta a ser listado em futuras associações.
3. Pedido ao *request-reponse-microservice* para adicionar um recurso do tipo FHIR DeviceRequest. Note-se que isto só acontece quando o tuplo contém a estrutura deviceRequest, ou seja, quando o deviceId do tuplo corresponde a uma smartbox.

As peças *microservice* executam todos estes pedidos à base de dados, devolvendo ao *composite* a informação recebida.

Com os recursos *FHIR Device* atualizados e os recursos *FHIR DeviceRequest* criados para cada uma das *smartboxes*, a associação está finalizada. Assim, a informação do sucesso da operação passa pelo *middleware* para o *widget*, sendo o utilizador finalmente notificado que os dispositivos foram associados.

5.1.3.2 Caso de Uso: Consultar Dispositivos (RF2)

Na Figura 23 apresenta-se o diagrama de sequência relativo ao Caso de Uso: Consultar Dispositivos.

Como se pode ver na Figura 23, o *associated-devices-widgets* é a interface do utilizador com que este interage para consultar os dispositivos associados a um determinado paciente.

O *widget* começa por fazer um pedido, de forma a **receber a informação de todos os dispositivos associados ao paciente**.

O pedido visa procurar na base de dados recursos *FHIR* com as seguintes condições:

1. Recursos do tipo *DeviceRequest* (representam uma associação previamente realizados de um dispositivo do tipo *smartbox* a um paciente), com:
    a. Campo *status* preenchido com a *string* "*active*".
    b. Campo *subject.reference* com o identificador do paciente em questão.
2. Recursos do tipo Device, com:
    a. Campo *status* preenchido com a *string* "*active*".
    b. Campo *patient.reference* com o identificador do paciente em questão.
3. Recursos do tipo *Practitioner*[66] que sejam referenciados nos *DeviceResquest* (no campo *requester* ou *performer*) da condição 1.
4. Recursos do tipo *PractitionerRole*[67] (representam uma função de profissional de saúde) que no campo *practitioner.reference* tenham a referência para qualquer um dos recursos *Practitioner* da condição 3.

[66] Disponível em setembro de 2022 em https://www.hl7.org/fhir/practitioner.html
[67] Disponível em setembro de 2022 em https://www.hl7.org/fhir/practitionerrole.html

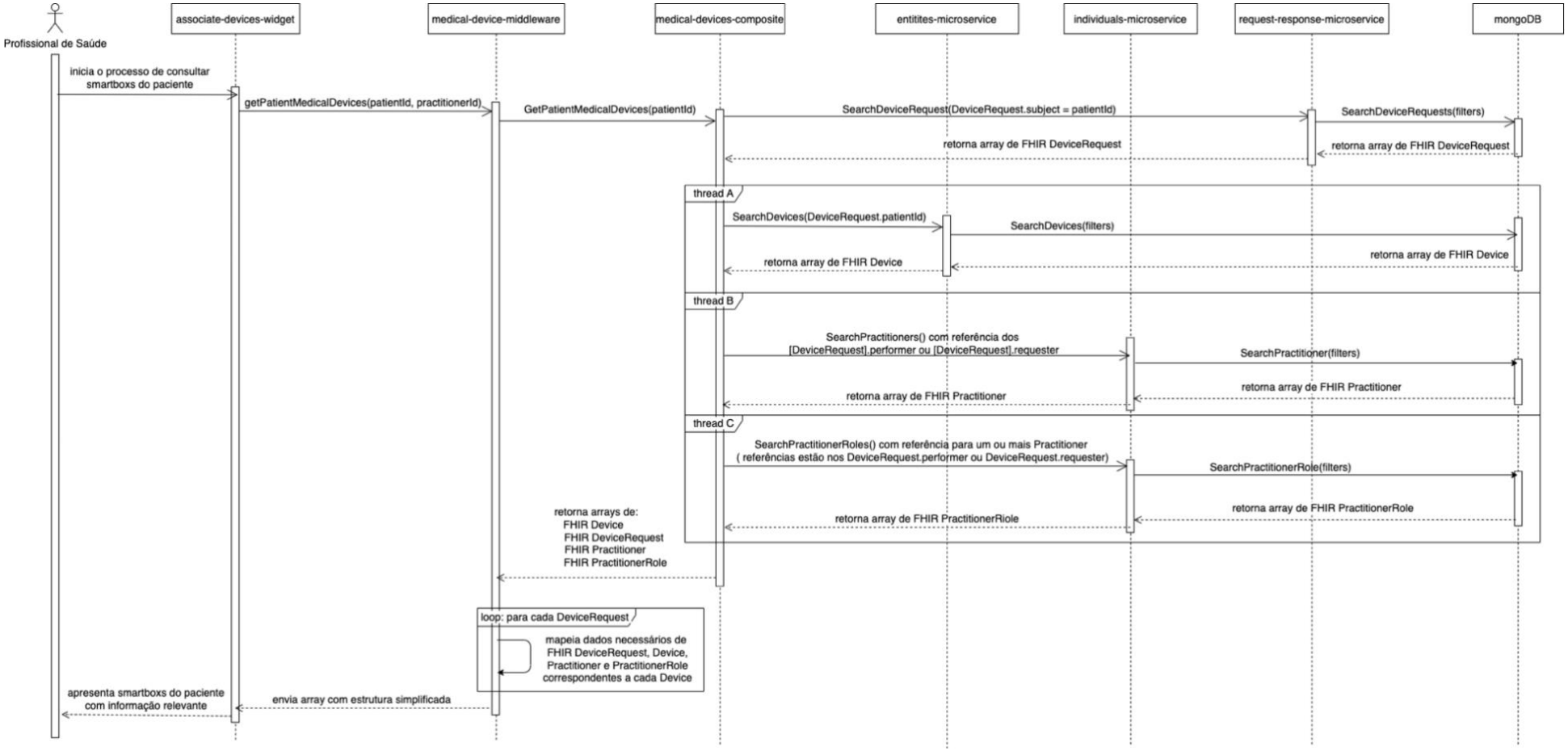

Figura 23 - Diagrama de Sequência: Consultar Dispositivo

O pedido é enviado ao *middleware*, que comunica com o *composite*. Este comunica com vários *microservices*, para realizar *queries* à base de dados, com as condições acima especificadas.

Os recursos pelos quais cada *microservice* é responsável (operações *CRUD*) para este caso de uso estão descritos na Tabela 9.

Tabela 9 - Recursos FHIR por microservice

Microservice	Recurso
entities-microservice	FHIR Device
Individuals-microservice	FHIR Practitioner e FHIR PractitionerRole
request-response-microservice	FHIR DeviceRequest

Cada *microservice* realiza os pedidos de obtenção à base de dados, devolvendo ao *composite* os recursos *FHIR* existentes que obedecem às condições especificadas. A informação segue o percurso inverso, sendo a informação da lista tratada na peça *middleware*.

O *middleware* trata todos os dados recebidos, mapeando para uma lista campos dos vários tipos de recursos recebidos. Esta lista é composta por vários elementos, onde cada um deles representa informações de uma *smartbox* e respetivos sensores associados ao paciente.

Estes dados são enviados ao *widget* que os apresenta no ecrã, onde o utilizador pode consultar toda esta informação.

5.1.3.3 Caso de Uso: Adicionar Intervenções (RF3)

Na Figura 25 apresenta-se o diagrama de sequência relativo ao Caso de Uso: Adicionar Intervenções.

O *device-management-widget* é a interface do utilizador, com o qual este interage para consultar e adicionar intervenções para o dispositivo *Smartbox*.

O *widget* é assim iniciado no contexto de uma *smartbox*, começando por fazer um pedido, de forma a **receber a informação de todas as intervenções associadas ao dispositivo *Smartbox***.

O pedido visa procurar, na base de dados, recursos *FHIR* do tipo *ServiceRequest*[68] (representa uma intervenção a um dispositivo do tipo *smartbox*) que referenciem o identificador da *smartbox* à qual se está a realizar a consulta.

Assim, o *composite* indica ao *request-response-microservice* que pretende esta informação, recebendo-a posteriormente. Os recursos FHIR *ServiceRequest* são devolvidos ao *middleware*,

[68] Disponível em setembro de 2022 em https://www.hl7.org/fhir/servicerequest.html

que filtra a informação, enviando para o *widget* uma estrutura de dados provenientes de FHIR semelhante à da Figura 24:

- **authoredOn** – Preenchido com a data e hora do momento, calculada diretamente no *middleware*.
- **subject** – Indica a que entidade vai ser realizada a intervenção, através de referência. Neste caso é a *smartbox*, sendo este campo preenchido com o *deviced* recebido do *widget*.
- **occurrenceTiming** – Agrega toda a informação relativa à recorrência temporal da monitorização.
- **requester** - Indica qual o profissional de saúde que pediu esta associação, através de referência. É preenchido com o *practitionerId* recebido do *widget*.
- **note** – Representa uma nota médica associada à intervenção.
- **status** – Indica o estado da intervenção. Para este caso de uso, é preenchido com a *string* "*draft*", dado representar uma monitorização agendada para um momento futuro.

```
{
   authoredOn: date,
   subject: deviceId
   occurrenceTiming: reccurrence
   requester:  practitionerId,
   note: annotation
   status: 'DRAFT'
}
```

Figura 24 - Estrutura com campos de FHIR ServiceRequest (pseudocódigo)

Através desta informação, o *widget* apresenta ao utilizador todas as intervenções existentes para a *smartbox*.

Posteriormente, o utilizador pode adicionar, editar ou remover intervenções. Por questões de simplicidade, segue-se apenas a explicação do caso de **adicionar intervenções**.

Para este cenário, o utilizador preenche no *widget* novas intervenções, definindo as suas recorrências temporais e notas médicas associadas, gravando posteriormente cada uma das novas intervenções.

Para cada intervenção adicionada, o *middleware* cria uma estrutura que obedece ao recurso FHIR *ServiceRequest*, com a mesma informação da Figura 24, e comunica-a ao *composite*. Por sua vez, este comunica com o *request-response-microservice*, indicando-lhe que deve criar o respetivo recurso na base de dados. Após estas operações, é retornada uma mensagem de sucesso e o *widget* informa ao utilizador que as intervenções foram adicionadas.

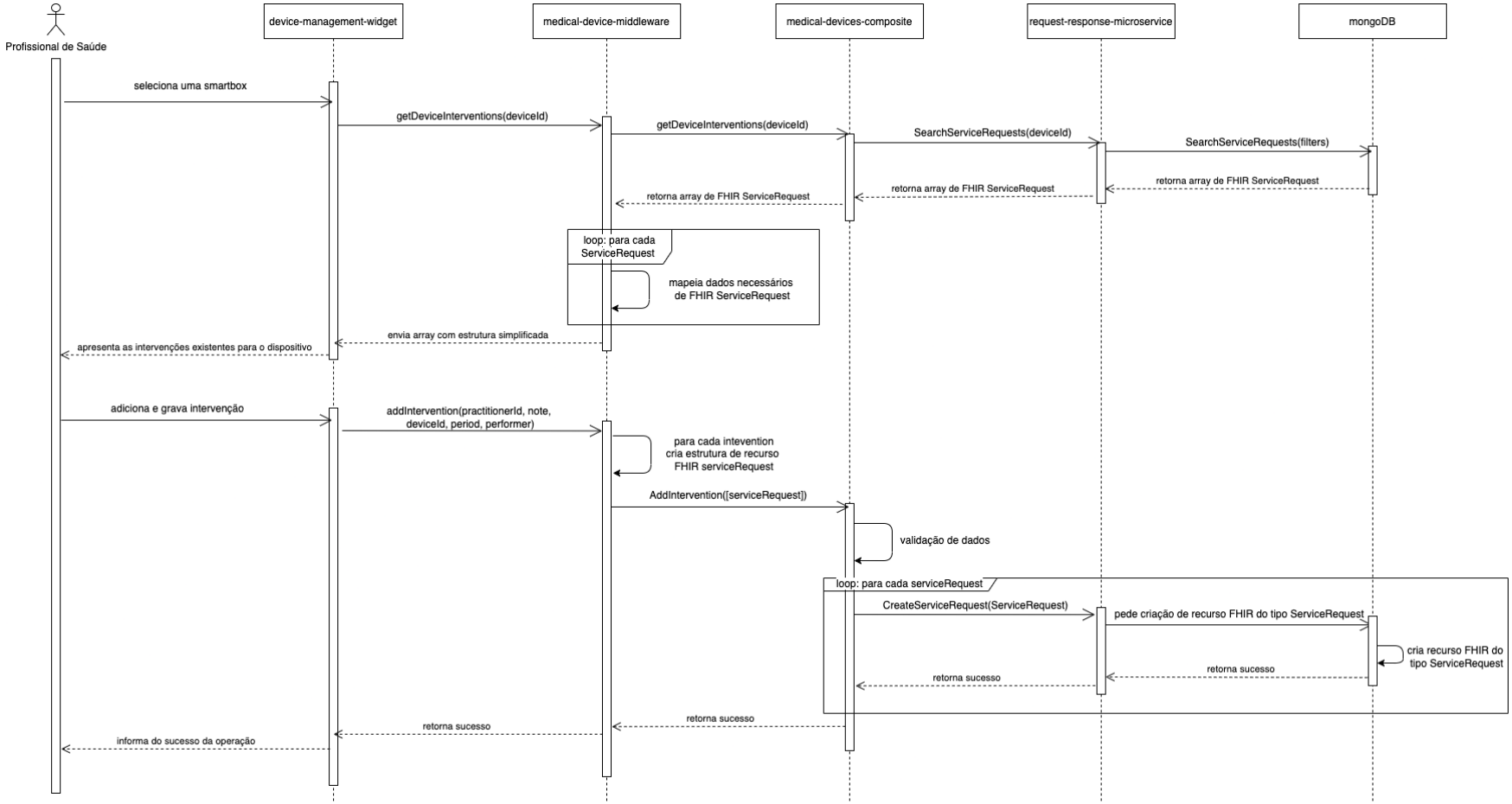

Figura 25 - Diagrama de Sequência: Adicionar Intervenção

5.1.3.4 Caso de Uso: Consultar Observações (RF4)

Na Figura 26 apresenta-se o diagrama de sequência relativo ao Caso de Uso: Consultar Observações.

O *vital-signs-widget* é a interface do utilizador, com o qual este interage para consultar os resultados das monitorizações feitas ao paciente.

O *widget* é assim iniciado no contexto de um paciente, começando por fazer um pedido, de forma a **receber a informação das observações associadas ao paciente**. Dado que o paciente pode ter vários sensores e até vários dispositivos *smartbox*, todos os resultados de intervenções devem ser apresentados através deste *widget*.

O pedido visa procurar, na base de dados, recursos *FHIR* do tipo *Observation*[69] (no caso representa um conjunto de dados relativos aos resultados de uma intervenção / monitorização a sinal vital), que referenciem o identificador do paciente cuja consulta se está a realizar.

Assim, por cada tipo de sinal vital, o *middleware* faz um pedido ao *composite*, com essa indicação.

Por sua vez, o *composite* indica ao *request-response-microservice* que pretende essa informação, recebendo-a posteriormente. Os recursos FHIR *Observation* são devolvidos ao *middleware*, que filtra a referida informação, enviando para o *widget* uma estrutura com dados de proveniência FHIR *Observation*, representativos da informação de cada monitorização, relativa a cada sinal vital:

- **effectiveDateTime** – Preenchido com a data e hora da monitorização / resultados.
- **subject** – Indica o paciente dos resultados da monitorização, através de referência para um recurso FHIR *Practitioner*.
- **value** – Representa o valor do resultado da monitorização a um sinal vital.
- **code** – Estrutura com o nome e um código único do sinal vital ao qual foi realizada a monitorização.

Desta forma, o *widget* apresenta ao utilizador os dados de cada monitorização a cada sinal vital.

[69] Disponível em setembro de 2022 em https://www.hl7.org/fhir/observation.html

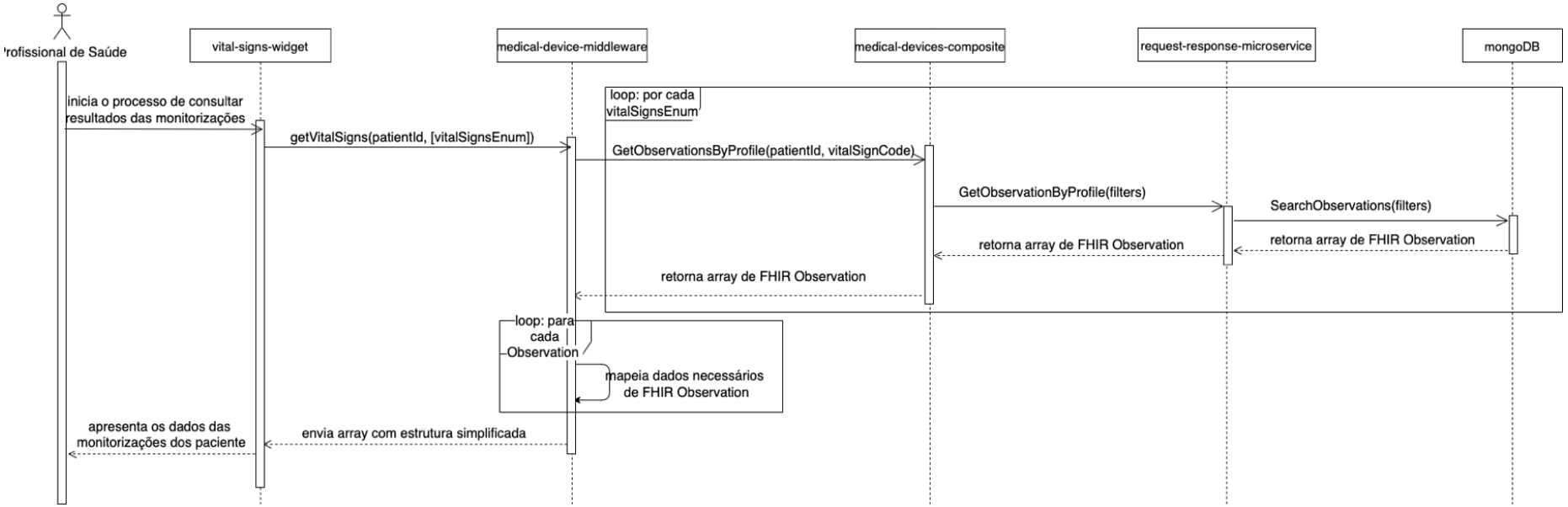

Figura 26 - Diagrama de Sequência: Consultar Observações

6 Implementação da Solução

Definida a arquitetura do sistema no Capítulo Desenho da Solução, segue-se a fase de implementação.

A implementação da solução deve seguir uma metodologia, sendo esta descrita numa etapa inicial do capítulo. De seguida, é apresentada a estrutura de cada tipo de peça do sistema. Finalmente, é documentada de forma exaustiva a envolvente técnica do desenvolvimento para cada caso de uso.

6.1 Metodologia de Desenvolvimento

Uma metodologia é um conjunto de convenções que uma equipa se compromete a seguir. Isto implica que cada equipa tenha uma metodologia diferente, embora possam ser semelhantes entre elas.

Desta forma, as metodologias Agile são as convenções escolhidas por cada equipa, que seguem os valores e princípios Agile[70].

No caso concreto deste projeto, o desenvolvimento foi feito apenas pelo autor. No entanto, os valores e princípios Agile não foram negligenciados. Alguns pontos a assinalar são o foco em entregar valor ao cliente, o desenvolvimento incremental e o *Code Review* efetuado por membros da equipa em que o autor se insere.

Dado que o projeto envolve vários componentes, seguem-se os pontos transversais no processo de desenvolvimento de cada componente:

- Git[71] é utilizado como ferramenta de versionamento.
- Cada componente do sistema tem o seu repositório no *Bitbucket*[72].
- Cada repositório segue a estratégia de *Feature Branch*[73], sendo criado um ramo (*branch*) novo para cada desenvolvimento.
- Cada desenvolvimento, em cada peça, origina um *Pull Request*, que tem de ser aceite pelos membros da equipa, após *Code Review*.
- Um *Pull Request*, quando aceite pelos membros da equipa, passa por um processo de validação de código. A ferramenta *Jenkins*[74] executa este processo, através de uma *pipeline*, assegurando a qualidade e correta testagem do desenvolvimento.

[70] Retirado em setembro de 2022 de https://www.agilealliance.org/agile101/
[71] Retirado em setembro de 2022 de https://git-scm.com/
[72] Retirado em setembro de 2022 de https://bitbucket.org/
[73] Retirado em setembro de 2022 https://www.optimizely.com/optimization-glossary/feature-branch/
[74] Retirado em setembro de 2022 https://www.jenkins.io/

6.2 Estrutura dos componentes

A estrutura de um componente representa a sua hierarquia de pastas e diretórios no projeto.

Nas próximas subsecções é estudada a estrutura base de cada tipo de peça trabalhada ao longo do desenvolvimento: *widget*, *middleware*, *composite* e *microservice*.

6.2.1 Widget

De acordo com a vista de desenvolvimento selecionada (Figura 15), a solução tem 4 *widgets* envolvidos: *device-management-widget*, *device-search-widge*t, *associated-devices-widget* e *vital-signs-widget*.

Por razões de manutenção, coerência e organização, todas estas peças de software são desenvolvidas com recurso ao *React*, utilizam o *GraphQL* para comunicar com a peça *middleware* e utilizam componentes e *templates* de *frontend* disponibilizados por um *design system* do qual o autor é responsável.

A sua estrutura, representada na Figura 27, é composta pelos seguintes elementos:

- **node_modules** – guarda todos os pacotes das dependências que o projeto necessita para sua a execução, após instalação através do comando *yarn install*.
- **public** – inclui o ficheiro *manifest.json* onde são definidos os tamanhos que o widget pode ter.
- **src**
    - **containers –** contém ficheiros com vistas completas relativas a um cenário de interface gráfica.
    - **componentes -** contém ficheiros que representam pequenas peças de interface gráfica, reutilizáveis e independentes.
    - **pages/home** – inclui um único ficheiro, responsável por ser o ponto de entrada do *widget*. Este chama os componentes, containers e ainda as *queries* e *mutations*.
    - **state**
        - **queries –** contém a declaração de todos os pedidos para receção de dados proveniente do *middleware*, incluindo os *inputs* e *outputs*, através de *GraphQL*.
        - **mutations -** contém a declaração de todos os pedidos para manipulação de dados proveniente do *middleware*, incluindo os valores que deve receber e retornar, através de *GraphQL*.
    - **utils** – tem todos os ficheiros de utilitários que possam ser necessários.
- **tests**
    - **acceptance** – contém todos os testes de aceitação realizados.
    - **unit** – contém todos os testes unitários realizados.

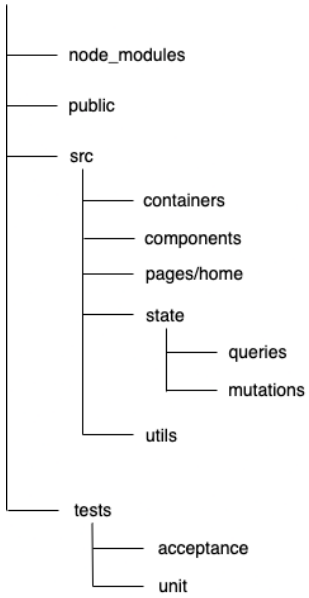

Figura 27 - Estrutura de diretórios: Widget

6.2.2 Middleware

Observando o diagrama de componentes (Figura 15), a solução apresenta apenas uma peça *middleware*: *medical-devices-middleware*.

Esta peça é desenvolvida em *node.js*, utiliza o *GraphQL* para comunicar com as peça ***widget*** e *gRPC* para comunicar com as peças *microservice*.

Na Figura 28 está representada a estrutura deste componente, que é constituída pelos seguintes diretórios:

- **src**
  - **definitions** - declara as *queries* e *mutations*, definindo o contrato de dados com o widget.
  - **resolvers** – faz a gestão de cada *query* e *mutation* declarada em *definitions*. Habitualmente chama os métodos do *composite* e manipula estruturas de recursos FHIR, de forma a obter ou acrescentar dados.
  - **services** – responsável por chamar métodos do *composite*.
- **tests**
  - **resolvers** – contém testes relativos aos *resolvers*.
  - **services** – contém testes relativos aos *services*.

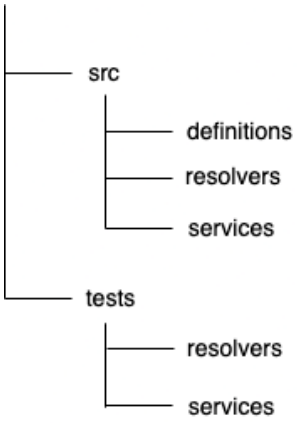

Figura 28 - Estrutura de diretórios: Middleware

6.2.3 Composite

Como está patente no diagrama de componentes (Figura 15), a solução apresenta apenas uma peça *composite*: *medical-devices-composite*.

Esta peça é desenvolvida em *.NET Core*, utiliza *gRPC* para comunicar com as peça *microservice* e trata da lógica de negócio. Para isto, o *gRPC* utiliza ficheiros onde são definidos os contratos de dados entre esta peças, os *proto-buffs*.

Na Figura 29 está representada a estrutura deste componente, que é constituída pelos seguintes diretórios:

- **Grpc**
  - **Controllers** – funciona como uma ponte entre os pedidos que chegam do middleware e os serviços.
  - **Services** – tratam cada um dos pedidos que chegam do middleware. Aplica lógica de negócio e comunica com os microservices.
  - **Validators** – responsável por pequenos métodos utilitários que validam um conjunto de dados.
- **tests**
  - **Controllers** – contém testes relativos aos controllers.
  - **Services** – contém testes relativos aos services.

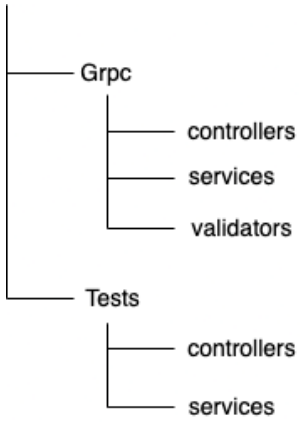

Figura 29 - Estrutura de diretórios: Composite

6.2.4 Microservices

Como se pode observar na vista de desenvolvimento selecionada (Figura 15), a solução tem 3 *microservices* envolvidos: *entities-microservice*, *request-response-microservice* e *individuals-microservice.*

Estas peças de software são desenvolvidas em *.NET Core*, utilizam o *Grpc* para comunicar com a peça *composite* e comunicam também diretamente com a base de dados *MongoDB*. Os *microservices* são peças essencialmente com funções de pedidos CRUD à base de dados, pelo que o seu desenvolvimento tem um grau de automatização elevado.

A estrutura das peças *microservice* é igual à das peças *composite*, podendo ser consultada na Figura 29:

- **Grpc**
  - **Controllers** – funciona como uma ponte entre os pedidos que chegam da peça composite e do diretório services.
  - **Services** – tratam cada um dos pedidos que chegam do composite, fazendo o respetivo pedido à base de dados
  - **Valiadators** – responsável por pequenos métodos utilitários que validam conjuntos de dados.
- **tests**
  - **Controllers** – contém testes relativos aos controllers.
  - **Services** – contém testes relativos aos services.

6.3 Implementação dos Casos de Uso

Ao longo desta secção é detalhado o desenvolvimento dos casos de uso (Figura 13) definidos para a solução. Dado o desenvolvimento ser transversal a todos os tipos de componentes estudados na Secção 6.2, o trabalho executado em cada peça de software é detalhado separadamente, exceto para as peças *microservice*. Estas são abordadas nas fases de detalhe das peças *composite*, pelo facto de apenas executarem operações *CRUD*.

6.3.1 Casos de Uso: Associar Dispositivos

De forma a satisfazer o requisito RF1, o sistema deve ser capaz de mostrar todos os dispositivos disponíveis para associar ao paciente e realizar a associação daqueles que o utilizador escolha.

Desta forma, e de acordo com o diagrama de sequência da Figura 14, o componente *devices-search-widget* será responsável por toda a interface gráfica que envolve o caso de uso, bem como por espoletar a associação de dispositivos.

6.3.1.1 Widget: associated-devices-widget

De forma a apresentar ao utilizador uma lista de dispositivos, permitindo-o filtrar por nome e fazer uma posterior seleção dos mesmos para associação, o *widget* tem de ser capaz de fazer dois tipos de pedidos ao *medical-devices-middleware*.

Dada a utilização do *GraphQL* para comunicação com o *middleware*, é necessária uma *query*, para o acesso a todos os dispositivos disponíveis e uma *mutation*, para a associação dos novos dispositivos.

Na Figura 30 está a declaração da *query* ***getAvailableMedicalDevices***. Como se pode interpretar pelo código, o pedido não envia nenhum tipo de *input* e pretende receber de volta uma lista com o conjunto de dados { *id*, *identifier*, *name*, *parentId* }, relativos a cada dispositivo (*smartbox* ou sensor).

```
query getAvailableMedicalDevices {
  getAvailableMedicalDevices {
    id
    identifier
    name
    parentId
  }
}
```

Figura 30 - Declaração da query getAvailableMedicalDevices

Relativamente à *mutation*, para a adição múltipla de *smartboxes* (e respetivos sensores), pode ver-se a sua declaração na Figura 31.

```
mutation addAssociatedDevices($devices: [DeviceToAssociate]) {
  addAssociatedDevices(devices: $devices)
}
```

Figura 31 - Declaração de Mutation *addAssociatedDevices*

Nesta *mutation*, por cada *smartbox* a associar, é enviada uma estrutura com os dados da mesma. Este pedido não especifica qual o retorno que espera, no entanto, por defeito é um *boolean*, sendo a resposta *true* em caso de sucesso e *false* em caso contrário.

Os pedidos são inicializados e a *query* é realizada na renderização do ecrã. Consoante a resposta, a informação recebida é mapeada para o ecrã. Para isto são utilizados os *React Hooks useState* e *useEffect* como se pode ver na Figura 32.

```
const [devices, setDevices] = useState([])
const [devicesSelected, setDevicesSelected] = useState([])
const [getAllDevices, { data, error, loading }] = useLazyQuery(GET_DEVICES)
const [associateDevices, { loading: loadingAssociate }] = useMutation(ASSOCIATE_DEVICES)

useEffect(() => {
  if (data && data.getAvailableMedicalDevices) {
    const newDevices = data.getAvailableMedicalDevices
      .filter((i) => isEmpty(i.parentId))
      .map((parent) => ({
        children: data.getAvailableMedicalDevices
          .filter((other) => other.parentId === parent.id)
          .map((descendent) => ({
            id: descendent.id,
            key: `${descendent.id}-${descendent.name}`,
            name: descendent.name,
          })),
        id: parent.id,
        key: `${parent.id}-${parent.name}`,
        name: `${parent.identifier} | ${parent.name}`,
      }))
    setDevices(devices.concat(newDevices))
  }
}, [data])
```

Figura 32 - Uso de useState e useEffect para guardar e mapear dados recebidos

O *useEffect* permite realizar operações periodicamente, enquanto o *useState* é usado para guardar dados numa variável, permitindo alterações posteriores, de forma assíncrona.

Assim, os *useEffect* são usados para que o *widget* realize o pedido *getAllDevices* de obtenção de dispositivos. A informação recebida é armazenada através do *useState devices* e é realizado

um mapeamento destes dados para uma estrutura hierárquica, onde se distinguem as *smartboxes* e os seus sensores.

Quando o utilizador guarda estas seleções, a *mutation* é executada com os dados dos dispositivos selecionados, incluindo sensores, através do uso do u*seState selectedDevices*, que armazena, em cada momento, os dispositivos selecionados.

Para a interface gráfica do utilizador utilizou-se um *template* React de *frontend*.

Este *template*, denominado ***Search Template*** (Figura 33), foi desenvolvido no *Design system* (que utiliza a *framework Storybook*) da equipa onde o autor trabalha. Dadas as caraterísticas de listagem, filtragem e seleção, será relevante em muitas outras áreas de negócio do CPOE, quando este se expandir para além da monitorização de sinais vitais. De ressalvar que esta utilização transversal acrescenta coerência ao CPOE, facilitando a experiência do utilizador.

Vejam-se as responsabilidades de algumas das *inputs* mais relevantes do *SearchTemplate* (Figura 33):

- **tableProps** – para apresentar as *smartboxes* o *template* usa, internamente, um componente tabela. É necessário assim definir as colunas que apresentará. No caso, será uma coluna para o nome da *smartbox* e uma coluna com uma seta clicável, permitindo a expansão da linha. Aquando da sua expansão, é possível ver os sensores associados à *smartbox*.
- **onSelectedItemsChange** – responsável por realizar uma ação quando é feita uma seleção. No caso, altera o *useState selectedDevices*.
- **bodyTitle** – define um título a apresentar na zona superior
- **footerTitle** – define um título na zona inferior, para apresentar o conteúdo que indica os nomes dos itens selecionados a cada momento.

```
<Search
  dataTest="associated-diagnostic-search"
  bodyTitle={`${translate('clinical_terms.RESULTS')} (${ dataFiltered.length })`}
  footerTitle={`${translate('expression.SELECTION_TO_ADD')} (${dataSelectedLength})`}
  tableProps={{
    columns: [
      {
        key: 'device',
        render: (_, item) => (
          <Highlighter
            text={`${item.name}`}
            highlight={isEmpty(item?.parentId) ? searchWord : ''}
          />
        ),
        title: <Text weight="bold" label={translate('expression.NAME')} />
      }
    ],
    dataSource: dataFiltered,
    expandable: {
      expandedRowRender: record => (
          {record?.children.map(sensor => (
            <Row key={sensor.key}>
              <Col offset={1}>{sensor.name}</Col>
            </Row>
          ))}          ),
      rowExpandable: record => !isEmpty(record?.children)
    }
  }}
  onSelectedItemsChange={items => setSelectedDevices(items)}
/>
```

Figura 33 - Search Template

6.3.1.2 Middleware: medical-devices-middleware

De forma a servir corretamente o *widget*, o *middleware* deve implementar a *query* e *mutation*, relativas à Figura 30 e Figura 31, respetivamente.

Estas foram inicialmente declaradas de acordo com a mesma lógica definida no *widget*, como se pode ver na Figura 34.

```
type AvailableDevice {
  id: String
  name: String
  parentId: String
}

type Query {
  getAvailableMedicalDevices(): [AvailableDevice]
}
type Mutation {
  addAssociatedDevices(deviceId: String): Boolean
}
```

Figura 34 - Definição do contrato de dados GraphQL

Para a comunicação posterior com o *composite*, é também necessário definir o contrato de dados entre este e o *middleware*, através de um *protobuf* (Figura 35).

```
[…]
message DeviceIdAndDeviceRequestItem {
    google.fhir.r4.core.DeviceRequest deviceRequest = 1;
    string deviceId = 2;
}
message AssociateDevicesRequestMsg {
    repeated DeviceIdAndDeviceRequestItem deviceItems = 1;
}
message AssociateDevicesResponseMsg {
    bool success = 1;
    repeated cockpit.grpc.shared.ErrorMsg errors = 2;
}
message GetAvailableMedicalDevicesResponseMsg {
    bool success = 1;
    repeated google.fhir.r4.core.Device devices = 2;
    repeated cockpit.grpc.shared.ErrorMsg errors = 3;
}

service ServerMedicalDeviceComposite {
    rpc GetAvailableMedicalDevices(google.protobuf.Empty)
     returns (GetAvailableMedicalDevicesResponseMsg) {}
    rpc AssociateDevices(AssociateDevicesRequestMsg) returns (AssociateDevicesResponseMsg) {}
[…]
}
```

Figura 35 - Protobuf para pedido GetAvailableMedicalDevices e AssociateDevices

Para responder à *query* ***getAvailableMedicalDevices***, recebida através do *widget* para obtenção dos dispositivos disponíveis, o *middleware* faz o pedido *gRPC* **GetAvailableMedicalDevices()** ao *medical-devices-composite* (Figura 36).

```
this.medicalDeviceClient.GetAvailableMedicalDevices(filters, meta, (err, response) => {
    [...]
    ctx.logger.info(
        ` Requested Composite Medical Device with metadata ${JSON.stringify(
            filters
        )}`,
        metadata
    )
    return resolve(response)
})
```

Figura 36 - Pedido gRPC GetAvailableMedicalDevices ao Composite

A resposta recebida do *composite* deve ser de acordo com a estrutura do *protobuf* da Figura 35, pelo que vai conter um conjunto de objetos FHIR do recurso *Device* (palavra-chave *repeated*), um *boolean* a indicar o sucesso ou insucesso da operação e uma estrutura que poderá conter informação sobre eventuais erros, se aplicável.

Através dos recursos FHIR *Device* recebidos, são extraídos os dados necessários (*id*, *name*, *parentId*) de cada um, para enviar para o *widget* (Figura 37).

```
return response.devices?.map(device => {
    return {
        id: getIdentifier(device),
        name: getName(device),
        parentId: getParentId(device)
    }
})
```

Figura 37 - Resposta ao Widget (getAvailableMedicalDevices)

Para realizar a extração, são usados métodos auxiliares que obtêm cada dado necessário, como se pode ver no exemplo para extrair o *name* do *Device*, na Figura 38.

```
export const getCodingValueBase = coding => {
    const value = get(coding, 'code.value') || ''
    return value
}

export const getName = device => {
    const coding = get(device, 'type.coding[0]') || {}
    return getDisplayValueBase(coding)
}
```

Figura 38 - Extração de campo de FHIR Device referente ao nome

Relativamente ao pedido ***addAssociatedDevices*()**, o objetivo é criar a associação entre o paciente e os dispositivos recebidos através da *mutation* (Figura 34) proveniente do *widget*.

O *middleware* recebe os identificadores de todos os dispositivos a associar (*deviceId*) e o identificador do paciente a que vão ser associados (*patientId*). Este último faz parte de um contexto que inclui, entre outros, o identificador do profissional de saúde, e, por defeito, é sempre enviado.

Para realizar esta associação, o *composite* tem de fazer alterações aos FHIR *Device* envolvidos, além de criar um FHIR *DeviceRequest* por cada *smartbox* a associar.

Desta forma, por cada dispositivo a associar, o *middleware* envia ao *composite* o seu identificador (*deviceId*) e a estrutura mínima necessária para a criação de um FHIR *DeviceRequest* (Figura 39).

```
const authoredOn = convertToProtoDateTime(moment())
const deviceIdAndDeviceRequests = []
const createDeviceRequestStructure = deviceId => {
  return {
    authored_on: authoredOn,
    code: {
      choice: 'reference',
      reference: {
        device_id: { value: deviceId },
        reference: 'device_id'
      }
    },
    patient: {
      patient_id: { value: ctx.metadata.patientId },
      reference: 'patient_id'
    },
    requester: {
      practitioner_id: { value: ctx.metadata.practitionerId },
      reference: 'practitioner_id'
    },
    status: { value: 'ACTIVE' }
  }
}
devices.map(device => {
  deviceIdAndDeviceRequests.push({
    deviceId: device.deviceId,
    deviceRequest: device.isParent ? createDeviceRequestStructure(device.deviceId) : null
  })
})
```

Figura 39 - Criação de estrutura para FHIR DeviceRequest

Note-se que, a estrutura *DeviceRequest* só é criada quando o *device* é um dispositivo pai, ou seja, uma *smartbox*.

Ainda relativamente a esta estrutura, como se pode ver na Figura 39, os campos existentes respeitam o *standard FHIR* para o recurso *DeviceRequest*, onde são injetados dados, tais como o identificador do dispositivo *smartbox*, do profissional de saúde e do paciente.

Posteriormente, esta informação é enviada para o *composite* com o pedido *AssociateDevices*() e de acordo com o *protobuf* da Figura 36.

6.3.1.3 Composite: medical-devices-composite

Para este caso de uso, o composite é responsável pelas seguintes operações:

- **GetAvailableMedicalDevices()** - pedido ao *entities-microservice* pelos *FHIR Device* em estado ativo e que não referenciam nenhum paciente.
- **AssociateDevices()** – pedido ao *request-response-microservice* para a criação de FHIR DeviceRequest.

Para estes pedidos, o contrato de dados existente no *middleware* é partilhado também pelo *composite*, disponibilizado na Figura 35.

Relativamente ao **GetAvailableMedicalDevices**(), para o pedido ser realizado ao *entities-microservice,* é criado um filtro a restringir os resultados recebidos. Este indica que o *FHIR Device* não tem referência para qualquer paciente e tem de estar em estado *active*.

Como se pode ver na Figura 40, o filtro é criado com recurso ao *BSON Document*, que cria uma representação binária de um documento *JSON*. Logo de seguida, pode ver-se também o pedido ao *microservice* e a devolução da resposta ao *middleware*.

Quanto ao pedido **AssociateDevices()**, quando é invocado através do *middleware*, recebe a lista *DeviceItems* com um ou mais identificadores de dispositivo e uma ou mais estruturas para a criação de *FHIR DeviceRequest* na base de dados.

Com esta informação, o *composite* faz pedidos para atualizar todos os *FHIR Device*, que passam assim a ter uma referência para o paciente, deixando de estar disponíveis para novas associações.

Na Figura 41 pode ver-se a alteração a um destes recursos, onde é feito um pedido para obter o *FHIR Device* com um determinado identificador. Posteriormente é injetada a referência do paciente no campo *patient*, sendo feito um pedido ao *microservice* para atualizar o recurso.

```
public async Task<GetAvailableMedicalDevicesResponseMsg> GetAvailableMedicalDevices
(GetAvailableMedicalDevicesRequestMsg request, RequestContextModel context)
{
    var jsonFilter = new BsonDocument();
    jsonFilter.AddRange(new BsonDocument("status", "active"));
    jsonFilter.AddRange(
    new BsonDocument("patient.reference", new BsonDocument("$in", new BsonArray
    {
        BsonNull.Value,
        ""
    })));

    var searchRequest = new JsonSearchRequestMsg
    {
        JsonFilter = jsonFilter.ToJson()
    };
    var response = new GetAvailableMedicalDevicesResponseMsg { Success = true };

    logger.LogInformation($"Call: Search Devices with ' Request = {JsonSerializer.Serialize(searchRequest)} '");
    var deviceResponse = deviceServiceClient.Search(searchRequest, context.RequestHeaders);

    await foreach (var device in deviceResponse.ResponseStream.ReadAllAsync())
    {
        response.Devices.Add(device);
    }
    logger.LogInformation("GetAvailableMedicalDevices Ended");
    return response;
}
```

Figura 40 - Chamada ao entities-microservice por FHIR Device e mapeamento da resposta ao middleware

Caso seja um dispositivo pai (*smartbox*), resta uma operação para terminar o processo de associação: um pedido para a criação do novo *FHIR DeviceRequest*, o que consuma a associação (Figura 42).

Em suma, as operação para alteração do *FHIR Device* (Figura 41) são realizadas para todos os dispositivos a associar, enquanto que a operação de criação de *FHIR ServiceRequest* apenas acontece quando é recebida a estrutura *item.DeviceRequest* para associação, ou seja, quando se trata de um dispositivo *smartbox* (Figura 42).

Recorde-se que a estrutura para a criação desde *FHIR DeviceRequest* é recebida pela *middleware* (Figura 39), onde foi construída.

```
var searchRequest = new JsonSearchRequestMsg
{
    JsonFilter = jsonFilter.ToJson()
};

logger.LogInformation($"CreateDeviceRequestAndUpdateDevice | Call: Get medical device by id with ' Request =
{JsonSerializer.Serialize(searchRequest)} '");
var device = await RequestUtilities.SearchSingle<GoogleFHIR::Device>(_ => deviceServiceClient.Search(_,
context.RequestHeaders), jsonFilter.ToJson());

if (device != null)
{
    logger.LogInformation($"CreateDeviceRequestAndUpdateDevice | Update Device | Started");

    device.Patient = patientReference;

    var deviceToUpdateResponse = await deviceServiceClient.UpdateAsync(device, context.RequestHeaders);

    if (deviceToUpdateResponse.Success)
    {
            logger.LogInformation($"CreateDeviceRequestAndUpdateDevice | Update Device | Finished");
    }
    else
    {
            logger.LogError($"CreateDeviceRequestAndUpdateDevice | Update Device.
            {JsonSerializer.Serialize(deviceToUpdateResponse.Errors)}");
       response.Errors.AddRange(deviceToUpdateResponse.Errors);
    }
[...]
}
```

Figura 41 - Obtenção de FHIR Device e alteração do seu campo *patient*

```
if (item.DeviceRequest != null)
{
    logger.LogInformation($"CreateDeviceRequestAndUpdateDevice | Create device request | Started");
    var createDeviceRequestResponse = await deviceRequestServiceClient.CreateAsync(item.DeviceRequest,
    context.RequestHeaders);

    if (createDeviceRequestResponse.Success)
    {
            logger.LogInformation($"CreateDeviceRequestAndUpdateDevice | Create device request | Finished");
    }
    […]
}
```

Figura 42 - Criação de recurso *FHIR DeviceRequest* para *smartbox*

6.3.2 Caso de Uso: Consultar Dispositivos

De forma a satisfazer o requisito RF2, o sistema deve ser capaz de mostrar todos os dispositivos associados ao paciente.

Desta forma, e de acordo com o diagrama de sequência da Figura 14, o componente *associated-devices-widget* será responsável por toda a interface gráfica que envolve o caso de uso.

6.3.2.1 Widget: associated-devices-widget

Para apresentação ao utilizador da lista de dispositivos associados ao paciente, o widget tem de realizar um pedido ao *medical-devices-middleware*.

Este pedido é feito através de uma *query*, que devolve informação sobre cada *smartbox* associada ao paciente, assim como informação dos seus sensores.

Na Figura 43 está a declaração da *query* ***getPatientMedicalDevices***. Apesar do pedido não especificar nenhum tipo de *input*, por defeito é sempre enviado o identificador do paciente e do profissional de saúde (utilizador). Como resposta, é definido que esta deve receber de volta uma lista com o conjunto de dados relativos à informação sobre cada conjunto *smartbox* – sensores.

```
query getPatientMedicalDevices {
  getPatientMedicalDevices {
    deviceId
    name
    sensors {
      id
      name
      status
    }
    date
    requester {
      name
      role
    }
    applier {
      name
      role
    }
    status
    deviceRequestId
  }
}
```

Figura 43 - Declaração da query getPatientMedicalDevices

O pedido é inicializado e a *query* é chamada na primeira renderização do ecrã.

Com recurso aos *React Hooks useState* e *useEffect*, os dados recebidos são armazenados e passados como *props* para o componente *React ExpandedDevices* (Figura 44).

```
useEffect(() => {
    if (data && data.getPatientMedicalDevices && data.getPatientMedicalDevices.length) {
        setDevices(data.getPatientMedicalDevices)
    } else setDevices([])
}, [data])

[...]

return isEmpty(devices) ? (
    renderEmpty()
) : (
    <ExpandedDevices data={devices} loading={loading} />
)
```

Figura 44 - useState e React props para envio de dados entre componentes React

Por sua vez, o componente *React ExpandedDevices* faz *import* do componente *frontend Table*, disponibilizado pelo *Design System*, para onde são mapeados os dados dos dispositivos. A implementação do *ExpandedDevices* pode ser vista na Figura 45.

Neste componente *ExpandedDevices* os dados de cada dispositivo são mapeados para cada linha da tabela apresentada ao utilizador. Aqui é definida uma estrutura hierárquica para a apresentação dos dispositivos, segundo as relações *smartbox*-sensores.

Esta tabela apresenta as seguintes colunas:

- **name** – nome do dispositivo.
- **deviceNumber** – número de sensores associados.
- **date** – data do pedido de associação do dispositivo.
- **requester** - profissional de saúde responsável pelo pedido de associação do dispositivo
- **applier** – profissional de saúde responsável pela aplicação do dispositivo.
- **status** – estado do dispositivo.

Para as linhas e colunas serem preenchidas, é utilizada a *prop dataSource* do componente *Table*, que invoca a função *getDataSource*. Esta função mapeia cada um dos dados dos dispositivos para as colunas referidas, de acordo com a Figura 46.

Note-se que, para as colunas de *requester* e *applier*, além do nome é mapeada a função do profissional de saúde (enfermeiro, médico, entre outros).

```
<Table
        dataTest="associated-devices-table"
        loading={loading}
        columns={[
            {
                key: 'name',
                render: (_, item) => <Text label={item.name} />,
                title: <Text level={3} strong label={'Descrição'} />,
                width: '25%'
            },
            {
                key: 'deviceNumber',
                render: (_, item) => <Text label={item.deviceNumber} />,
                title: <Text level={3} strong label={'Nº Dispositivos'} />
            },
            {
                key: 'date',
                render: (_, item) => <Text label={item.date} />,
                title: <Text level={3} strong label={'Data'} />
            },
            {
                key: 'requester',
                render: (_, item) => <Text label={item.requester} />
                 title: <Text level={3} strong label={'Pedido por'} />
            },
            {
                key: 'applier',
                render: (_, item) => <Text label={item.applier} />
                title: <Text level={3} strong label={'Aplicado por'} />
            },
            {
                key: 'status',
                render: (_, item) => <Text label={item.status} />,
                title: <Text level={3} strong label={'Estado'} />
            }
        ]}
        dataSource={getDataSource}
/>
```

Figura 45 - Componente React ExpandedDevices

```
const getDataSource = () => (
  data?.map(device => ({
      applier: `${device.applier.name} (${device.applier.role})`,
      children: device?.sensors.map(child => ({
        name: child.name,
        status: child.status
      })),
      date: device.date,
      deviceNumber: device.sensorsCount,
      name: device.name,
      requester: `${device.requester.name} (${device.requester.role})`,
      status: device.status
    })
  )
)
```

Figura 46 - Mapeamento dos dados dos dispositivos para a Table

6.3.2.2 Middleware: medical-devices-middleware

Para enviar ao *widget* os dados dos dispositivos do paciente, o *middleware* deve implementar a *query* relativa à Figura 43.

Estas *query* tem a mesma lógica definida no *widget*, como se pode ver na Figura 47.

Tal como anteriormente referido, a informação de input sobre o identificador do paciente é enviada por defeito, razão pela qual a *query* não recebe qualquer input.

Relativamente ao output da *query* (resposta devolvida ao *widget*), é organizado através de um *array* de objetos do tipo *PatientDevice*. Este contém vários dados relativos à *smartbox*, incluindo os sensores a ela associados e os dados dos profissionais de saúde envolvidos.

O *protobuf* é também definido (Figura 48) para a comunicação com a peça *composite*, de onde o *middleware* vai receber os recursos *FHIR*.

Após o pedido ao *composite*, este devolve uma lista para cada conjunto de recursos do mesmo tipo. No caso são listas de *FHIR DeviceRequest*, *FHIR Device*, *FHIR Practitioner* e *FHIR PractitionerRole*.

Estes recursos são recebidos na função *getPatientMedicalDevices()* ,que filtra a informação dos mesmos, constrói e devolve a estrutura adequada para responder ao *widget*.

```
type Sensor {
  id: String
  name: String
  status: String
}

type Practitioner {
  name: String
  role: String
}

type PatientDevice {
    deviceId: String
    deviceRequestId: String
    name: String
    sensorsCount: Int
    sensors: [Sensor]
    date: String
    requester: Practitioner
    applier: Practitioner
    status: String
}

type Query {
  getPatientMedicalDevices(): [PatientDevice]
}
```

Figura 47 - Definição da query getPatientMedicalDevices

```
[…]
message GetPatientMedicalDevicesResponseMsg {
    bool success = 1;
    repeated google.fhir.r4.core.DeviceRequest deviceRequests = 2;
    repeated google.fhir.r4.core.Device devices = 3;
    repeated google.fhir.r4.core.Practitioner practitioners = 4;
    repeated google.fhir.r4.core.PractitionerRole practitionersRoles = 5;
    repeated cockpit.grpc.shared.ErrorMsg errors = 6;
}
service ServerMedicalDeviceComposite {
    rpc GetPatientMedicalDevices(google.protobuf.Empty)
            returns (GetPatientMedicalDevicesResponseMsg) {}
}
```

Figura 48 - Protobuf para pedido GetPatientMedicalDevices

```
getPatientMedicalDevices: async (parent, input, { ctx, services }) => {
  const {
    devices = [],
    deviceRequests = [],
    practitioners = [],
    practitionersRoles = []
  } = await services.medicalDevicesService.getPatientMedicalDevices(ctx)

  return deviceRequests.map(devRequest => {
    const parentDevice = getParentDeviceFromDeviceRequest(devRequest, devices)
    const deviceChildren = getDeviceChildren(devRequest, parentDevice, devices)
    const performerPractitioner = getPerformerFromDeviceRequest(devRequest, practitioners)
    const requesterPractitioner = getRequesterFromDeviceRequest(devRequest, practitioners)

    return {
      applier: getPractitionerNameAndRole(performerPractitioner, practitionersRoles),
      date: getDeviceRequestAuthoredOn(devRequest).replace(' ', ' | '),
      deviceId: getId(parentDevice),
      deviceRequestId: getDeviceRequestId(devRequest),
      name: getName(parentDevice),
      requester: getPractitionerNameAndRole(requesterPractitioner, practitionersRoles),
      sensors: deviceChildren,
      sensorsCount: deviceChildren.length,
      status: getDeviceRequestStatus(devRequest)
    }
  })
```

Figura 49 – Método getPatientMedicalDevices

Os dados devolvidos ao *widget* (Figura 49) relativos a cada dispositivo são assim obtidos através da informação recebida do *composite*. Na Tabela 10 é esclarecido a proveniência *FHIR* de cada um destes dados.

Tabela 10 - Proveniência FHIR dos dados extraídos

Recurso	Elemento
FHIR Device	deviceId; name; sensors.id; sensors.name; sensorsCount;
FHIR DeviceRequest	date; deviceRequestId; status; sensors.status;
FHIR Practitioner	requester.name; applier.name;
FHIR PractitionerRole	requester.role; applier.role;

Para realizar o mapeamento destes dados, são utilizadas algumas funções auxiliares, como se pode verificar na Figura 49.

6.3.2.3 Composite: medical-devices-composite

De forma a cumprir o contrato de dados da Figura 48, o *composite* deve devolver listas de recursos *FHIR* (*DeviceRequest*, *Device*, *Practitioner* e *PractitionerRole*) relevante para a informação dos dispositivos associados ao paciente.

Assim, inicialmente é necessário efetuar o pedido ao request-response-microservice por recursos FHIR DeviceRequest que referenciem o paciente, através do campo *subject* (Figura 50).

```
[...]
var jsonFilter = new BsonDocument()
{
   {"status", "active"},
   {"subject.reference", $"Patient/{context.PatientId}"}
};

[...]
try
{
   var deviceRequests = await RequestUtilities.Search(_ =>    deviceRequestServiceClient.Search(searchRequest,
   context.RequestHeaders), jsonFilter.ToJson());
   response.DeviceRequests.Add(deviceRequests);
}
catch (Exception e)
{
   response.Errors.Add(new ErrorMsg { Message = e.Message });
}
[...]
```

Figura 50 - Obtenção de recursos FHIR DeviceRequest

É fundamental este ser o primeiro recurso a ser obtido, dado que contém informações necessárias para a obtenção dos restantes recursos necessários. Na Tabela 11 estão identificados os campos do FHIR *DeviceRequest* que fazem referência aos demais recursos FHIR exigidos neste âmbito.

Tabela 11 - Campos FHIR DeviceRequest que referenciam outros recursos FHIR

Recurso	Campos referência
FHIR Device	DeviceRequest.codeReference
FHIR Practitioner	DeviceRequest.requester; DeviceRequest.performer
FHIR PractitionerRole	DeviceRequest.requester; DeviceRequest.performer

Após esta operação, é então possível obter todos os outros recursos. Isto é feito de forma paralela, utilizando *threads*, através da classe *Task*.

Para simplificar, a Figura 51 apresenta apenas o pedido ao *entities-microservice* por recursos FHIR *Device*, dado ter uma lógica semelhante aos pedidos dos restantes recursos FHIR (*Practitioner* e *PractitionerRole*).

```
[...]
var tasks = new List<Task>();
tasks.Add(Task.Run(async () =>
   {
      var jsonDevicesFilter = new BsonDocument()
      {
         {"status", "active"},
         {"patient.reference", $"Patient/{context.PatientId}"}
      };

      var searchDevicesRequest = new JsonSearchRequestMsg
      {
         JsonFilter = jsonDevicesFilter.ToJson()
      };
      try
      {
         var devices = await RequestUtilities.Search(_ =>  deviceServiceClient.Search(searchDevicesRequest,
         context.RequestHeaders), jsonDevicesFilter.ToJson());
         response.Devices.Add(devices);
      }
      catch (Exception e)
      {
         response.Errors.Add(new ErrorMsg { Message = e.Message });
      }
   }));
[...]
```

Figura 51 - Obtenção de recursos FHIR Device

Após a obtenção de todos os recursos FHIR necessários, estes são devolvidos ao *middleware*.

6.3.3 Caso de Uso: Adicionar Intervenções

De forma a satisfazer o requisito RF3, o utilizador deve ser capaz de adicionar intervenções a um dispositivo. Isto é, deve ser possível definir novas recorrências para monitorizações aos sinais vitais.

Desta forma, e de acordo com o diagrama de sequência da Figura 25, a adição pressupõe também a consulta das intervenções existentes. Para isto, o componente *device-management-widget* será responsável por toda a interface gráfica que envolve o caso de uso.

Como mencionado anteriormente, apesar de não ter sido definido como requisito para o projeto, foram também implementados os cenários de editar e de eliminar intervenções. De

forma a não sobrecarregar o documento e respeitando os requisitos definidos, nas próximas subsecções será apenas detalhada a implementação do cenário de adição de intervenções, como o título desta secção indica.

6.3.3.1 Widget: device-management-widget

Para apresentação ao utilizador da lista de intervenções associadas ao dispositivo, o *widget* realiza um pedido ao *medical-devices-middleware*.

Este pedido é feito através de uma *query*, que devolve informação sobre cada intervenção associada ao dispositivo.

A Figura 52 apresenta a declaração da *query* ***getDeviceInterventions***. O pedido especifica que o *input* deve conter o identificador do dispositivo *smartbox* (*deviceId*), essencial para a obtenção das intervenções associadas apenas a esse dispositivo. Por sua vez, a resposta que a *query* espera é a de uma lista com o conjunto de dados relativos à informação sobre cada intervenção.

Como acontece noutros casos de uso, este pedido é executado na primeira renderização do ecrã.

```
query getDeviceInterventions($deviceId: String) {
  getDeviceInterventions(deviceId: $deviceId) {
    id
    name
    performerId
    status
    creationDate
    occurrence
    observations
  }
}
```

Figura 52 - Declaração da query getDeviceInterventions

Os dados recebidos são mapeados para o ecrã de forma a serem apresentados ao utilizador, com auxílio dos *React Hooks useState* e *useEffect*, de modo semelhante ao já foi exposto no caso de uso da Subsecção 6.5.1. Por cada intervenção recebida na resposta à *query*, é utilizado um painel com um pequeno formulário para definir dados tal como a recorrência da monitorização, o executante e notas sobre a mesma. Este painel é representado através do componente de *frontend Panel*, disponível no *Design System* e a sua utilização está patente na Figura 53.

Sendo iniciado o processo de criar intervenções, para cada uma é adicionado um outro *Panel* à lista prévia de painéis, sendo que o utilizador deve definir a recorrência e preencher todos os outros parâmetros. Quando estas intervenções são guardadas, a informação é enviada para a peça *middleware* através de uma *mutation*, declarada de acordo com a Figura 54.

```
<div key={`div-${intervention.id}-${idx}`}>
  <Panel>
    <Parameters
      onParametersChange={newData =>
        setInterventions(
          interventions.map((interv, intervIdx) =>
            idx === intervIdx
              ? { ...interv, ...newData }
              : interv
          )
        )
      }
      itemData={intervention}
    />
  </Panel>
</div>
```

Figura 53 - Mapeamento de cada intervenção para componentes de frontend

```
mutation addInterventions($deviceId: String, $interventionItems: [InterventionInput]) {
  addInterventions(deviceId: $deviceId, interventionItems: $interventionItems)
}
```

Figura 54 - Declaração da mutation AddInterventions

De forma a que as intervenções sejam corretamente criadas, são enviados os dados relativamente aos parâmetros definidos pelo utilizador (*interventionItems*), assim como o identificador do dispositivo *smartbox* ao qual se estão a adicionar as mesmas (*deviceId*).

6.3.3.2 Middleware: *medical-device*s-middleware

Para servir corretamente o *widget*, o *middleware* deve implementar a *query* e *mutation*, relativas à Figura 52 e Figura 54, respetivamente.

A mesma lógica definida no *widget* é respeitada na definição destes pedidos nesta peça, como se pode ver na Figura 55.

Para a comunicação posterior com o *composite*, é também necessário definir o contrato de dados entre estes, através de um *protobuf* (Figura 56).

Nesta fase é necessário perceber os recursos FHIR necessários. No caso, uma intervenção corresponde a um recurso FHIR *ServiceRequest*, o único necessário para o caso de uso.

```
type Intervention {
    id: String
    performerId: String
    creationDate: String
    status: String
    occurrence: String
    observations: String
}
input InterventionInput {
    id: String
    performer: String
    occurrence: String
    observations: String
}
type Query {
    getDeviceInterventions(deviceId: String): [Intervention]
}
type Mutation {
    addInterventions(deviceId: String, interventionItems: [InterventionInput]): Boolean
}
```

Figura 55 - Definição dos pedidos getDeviceInterventions e addInterventions

```
message GetDeviceServiceRequestsRequestMsg {
    string deviceId = 1;
}
message CreateDeviceServiceRequestsRequestMsg {
    repeated google.fhir.r4.core.ServiceRequest serviceRequests = 2;
}
message AssociateDevicesResponseMsg {
    bool success = 1;
    repeated cockpit.grpc.shared.ErrorMsg errors = 2;
}
message GetDeviceServiceRequestsResponseMsg {
    bool success = 1;
    repeated google.fhir.r4.core.ServiceRequest serviceRequests = 2;
    repeated cockpit.grpc.shared.ErrorMsg errors = 3;
}
message CreateDeviceServiceRequestsResponseMsg {
    bool success = 1;
    repeated cockpit.grpc.shared.ErrorMsg errors = 2;
}
service ServerMedicalDeviceComposite {
    rpc GetDeviceServiceRequests(GetDeviceServiceRequestsRequestMsg)
      returns (GetDeviceServiceRequestsResponseMsg) {}
    rpc CreateDeviceServiceRequests(CreateDeviceServiceRequestsRequestMsg)
      returns (CreateDeviceServiceRequestsResponseMsg) {}
}
```

Figura 56 - Protobuf para pedido GetDeviceServiceRequests e CreateDeviceServiceRequests

Relativamente à *query* **getDeviceInterventions()**, o *middleware* faz o pedido *gRPC* **GetDeviceServiceRequests()** ao *medical-devices-composite,* aplicando a mesma lógica anteriormente utilizada para o caso de uso da Subsecção 6.3.1 (Figura 36).

Como resposta, o *middleware* recebe da peça *composite* um objeto representativo de um FHIR *ServiceRequest* por cada Intervenção. De acordo com os dados que o *widget* necessita sobre cada intervenção, o *middleware* filtra a informação de cada FHIR *ServiceRequest*, capturando apenas os campos necessários e devolvendo-os ao *widget* (Figura 57).

```
getDeviceInterventions: async (parent, input, { ctx, services }) => {
    const { serviceRequests = [] } = await services.medicalDevicesService.getDeviceInterventions(ctx, input)
    return serviceRequests?.map(serviceRequest => {
        return {
            creationDate: getAuthoredOndate(serviceRequest),
            id: getServiceRequestId(serviceRequest),
            observations: getServiceRequestNote(serviceRequest),
            occurrence: getTimingOccurrenceDate(serviceRequest),
            performerId: getPerformerId(serviceRequest),
            status: getServiceRequestStatus(serviceRequest)
        }
    })
}
```

Figura 57 - Método getDeviceInterventions

```
const createServiceRequestStructure = intervention => {
    const { id = '', performer, occurrence, observations } = intervention
    const data = {
        authoredOn,
        code: {
            codeValue: '410188000',
            systemValue: 'http://snomed.info/sct'
        },
        note: observations,
        occurrence: {
            codeValue: occurrence,
            systemValue: 'https://www.rfc-editor.org/rfc/rfc5545.html'
        },
        performer,
        requester: ctx.metadata.practitionerId,
        serviceRequestId: id,
        status: 'DRAFT',
        subject: deviceId
    }
    return setServiceRequest(data)
}
```

Figura 58 - Criação de estrutura de FHIR ServiceRequest

Para a *mutation* **addInterventions()**, o *middleware* recebe do *widget* os dados necessários para a criação de estruturas que obedecem ao *standard* FHIR *ServiceRequest*. Para cada intervenção, é criada uma destas estruturas, de acordo com a Figura 58.

Posteriormente esta informação é enviada para o *composite*, que tem a responsabilidade de chamar o *microservice* adequado para a criação destes recursos FHIR na base de dados.

6.3.3.3 Composite: medical-devices-composite

De forma a cumprir o contrato de dados da Figura 56, o *composite* deve implementar os métodos **GetDeviceServiceRequests()** e **CreateDeviceServiceRequests()**.

Para ambos os casos, é necessário fazer pedidos *ao request-response-microservice*.

Relativamente ao **GetDeviceServiceRequests()**, é efetuado o pedido ao *microservice* por recursos FHIR *ServiceRequest* que referenciem o dispositivo *smartbox* (*deviceId*), através do campo *subject* e em que o seu *status* seja "*draft*", como se pode ver na Figura 59.

```
var jsonFilter = new BsonDocument()
{
   {"status", "draft"},
   {"subject.reference", $"Device/{request.DeviceId}"}
};
var searchRequest = new JsonSearchRequestMsg
{
   JsonFilter = jsonFilter.ToJson()
};
var response = new GetDeviceServiceRequestsResponseMsg();
try
{
   var serviceRequest = await RequestUtilities.Search(_ =>
   serviceRequestServiceClient.Search(searchRequest, context.RequestHeaders),
   jsonFilter.ToJson());
   response.ServiceRequests.Add(serviceRequest);
}
catch (Exception e)
{
   response.Errors.Add(new ErrorMsg { Message = e.Message });
}
```

Figura 59 - Obtenção de recursos FHIR ServiceRequest

No caso do **CreateDeviceServiceRequests()**, é recebido do *middleware* uma lista de estruturas que obedecem ao standard do recurso FHIR *ServiceRequest*. Para cada estrutura, o *composite* indica ao *microservice* que faça o pedido à base de dados para a criação do recurso.

```
foreach (var serviceRequest in request.ServiceRequests)
{
   try

      if (string.IsNullOrEmpty(serviceRequest?.Id?.Value))
      {
         var clientCreateResponse = await
         serviceRequestServiceClient.CreateAsync(serviceRequest,
         context.RequestHeaders);
      }
   }
   catch (Exception e)
   {
      response.Errors.Add(new ErrorMsg { Message = e.Message });
   }
}
```

Figura 60 - Pedido ao *microservice* para criação de um conjunto de FHIR *ServiceRequest*

6.3.4 Casos de Uso: Consultar Observações

Para satisfazer o requisito RF4, o sistema deve ser capaz de mostrar os resultados das monitorizações feitas a um paciente por um dispositivo.

É importante esclarecer que o âmbito desta dissertação não inclui o detalhe do processo de envio de informação dos dispositivos *smartbox* para esta solução. Para este caso de uso, deve assumir-se a pré-condição da existência de dados de monitorizações aos sinais vitais na base de dados.

De acordo com o diagrama de sequência da Figura 15, o componente *vital-signs-widget* será responsável por toda a interface gráfica que envolve o caso de uso.

6.3.4.1 Widget: *vital-signs*-widget

Para apresentação ao utilizador dos resultados das monitorizações do paciente, o *widget* realiza um pedido ao *medical-devices-middleware*.

Este pedido é feito através de uma *query*, que devolve informação sobre cada conjunto de dados relativo a cada sinal vital.

A Figura 61 apresenta a declaração da *query* ***getVitalSigns***. O pedido especifica que o *input* deve conter uma lista dos sinais vitais pretendidos. Além disto será também posteriormente necessário o identificador do paciente, que é enviado por defeito, sem que tenha de ser declarado no *input*.

Em relação ao output, a resposta que a *query* espera é a de uma lista com o conjunto de dados relativos à informação sobre cada observação relativa a um sinal vital, o que inclui a descrição do tipo de sinal vital, a unidade de medida e os resultados e datas de cada medição.

```
query getVitalSigns($vitalSigns: [VitalSignEnum]) {
  getVitalSigns(vitalSigns: $vitalSigns) {
    description
    evaluations {
      date
      values {
        value
      }
    }
    unit
  }
}
```

Figura 61 - Declaração da query getVitalSigns

Este pedido é feito imediatamente após a primeira renderização do ecrã e os dados recebidos são mapeados para o componente de *frontend Table*, num processo semelhante ao caso de uso detalhado na Secção 6.3.2, onde é também é usado este componente (Figura 46).

6.3.4.2 Middleware: *medical-device*s-middleware

Para dar resposta à *query* pedida pelo *widget*, o *middleware* declara-a de acordo com a Figura 62.

Como se pode interpretar através da definição dos tipos desta *query*, a informação proveniente do *widget* deve corresponder a um conjunto de um ou mais sinais vitais. Este deve respeitar os tipos aceites de acordo com o *enum VitalSignEnum*, onde estão representados os sinais vitais de pressão arterial, temperatura, frequência cardíaca, SpO2 e frequência respiratória.

Relativamente à comunicação com o *composite*, de onde são recebidos os recursos FHIR necessários, é também necessário definir o contrato de dados entre estes, através de um *protobuf* (Figura 56). Pode-se inferir através deste contrato que o único recurso FHIR necessário é o *Observation*.

```
type EvaluationValue {
  interpretation: String
  value: String
}
type VitalSignEvaluation {
  date: String
  values: [EvaluationValue]
}
type VitalSign {
  description: String
  evaluations: [VitalSignEvaluation]
  unit: String
}
enum VitalSignEnum {
  BLOOD_PRESSURE
  BODY_TEMPERATURE
  HEART_RATE
  OXYGEN_SATURATION
  RESPIRATORY_RATE
}
type Query {
  getVitalSigns(vitalSigns: [VitalSignEnum]) : [VitalSign]
}
```

Figura 62 - Definição da query getVitalSigns

```
message GetObservationsByProfileRequestMsg {
  string profile = 1;
  string code = 3;
  string codeSystem = 4;
}

message GetObservationsByProfileResponseMsg{
    bool success = 1;
    repeated cockpit.grpc.shared.ErrorMsg errors = 2;
    repeated google.fhir.r4.core.Observation observations = 3;
}

service ServerEarlyWarnScoreComposite {
    rpc GetObservationsByProfile(GetObservationsByProfileRequestMsg)
      returns (GetObservationsByProfileResponseMsg) {}
}
```

Figura 63 - Protobuf para pedido GetObservationsByProfile

Para cada sinal vital em que é necessário recolher dados, o *middleware* faz um pedido ao *composite*, como representado na Figura 64.

```
result.push({
    description: vitalSign.description,
    evaluations: this.buildObservationEvaluations(
        (await services.medicalDeviceService.getVitalSigns(ctx, input, vitalSign, code,
         codeSystem))
            .observations,
        vitalSign.decimals
    ),
    unit: vitalSign.unit
})
```

Figura 64 - Indicação de chamada ao composite por cada sinal vital

Para isto é usada uma função auxiliar responsável por adquirir as datas e valores das monitorizações (Figura 65). Posteriormente, a informação é enviada para o *widget*, através de uma única resposta.

```
buildObservationEvaluations(observations, decimals) {
    return observations.map(observation => ({
        date: getDate(observation),
        values: this.getEvaluationValue(observation, decimals)
    }))
}
```

Figura 65 - Mapeamento dos dados necessários provenientes de recurso FHIR Observation

6.3.4.3 Composite: medical-devices-composite

De forma a responder ao pedido do *middleware*, cumprindo o contrato de dados da Figura 63, o *composite* deve implementar o método **GetObservationsByProfile()**.

Para o caso, é necessário fazer pedidos ao *request-response-microservice*. O pedido é por recursos do tipo FHIR *Observation* que cumprem as seguintes condições: deve referenciar o paciente no campo *subject.reference* e corresponder ao tipo de sinal vital que se procura, em campos associados ao *code.coding* (Figura 66).

Assim, o *composite* indica ao *microservice* que faça o pedido à base de dados, devolvendo os recursos FHIR *Observation* que cumprem todas as condições.

```
var array = new BsonArray
   {
      new BsonDocument("subject.reference", $"Patient/{patientId}"),
      new BsonDocument("category", request.VitalSign),
      new BsonDocument("code.coding", new BsonDocument
      {
         { "$elemMatch", new BsonDocument
         {
            { "code", request.Code },
            { "system", request.CodeSystem }
         } }
      })
   };

var observationJsonFilter = new BsonDocument("$and", array);
var observationSearchRequest = new JsonSearchRequestMsg
{
   JsonFilter = observationJsonFilter.ToJson()
};
observationSearchRequest.JsonSorter = new BsonDocument("effectiveDateTime", -1).ToJson();

var observations = await RequestUtilities.Search(_ =>
 observationServiceClient.Search(observationSearchRequest, contextModel.RequestHeaders),
 jsonDevicesFilter.ToJson());
response.Devices.Add(observations);
```

Figura 66 - Obtenção de recursos FHIR Observation

6.4 Testes

De forma a assegurar a qualidade do desenvolvimento de todos os casos de uso, deve realizar-se a fase de testagem.

Assim, foram feitos vários tipos de testes, garantindo a qualidade do software de acordo com diferentes vertentes e granularidades.

Para isto o autor optou por seguir a metodologia de *Test-Driven Development* (TDD)[75]. No TDD os testes são escritos antes da implementação, pelo que foi seguido este processo para cada um dos casos de uso.

[75] Retirado em outubro de 2022 de https://www.browserstack.com/guide/what-is-test-driven-development

6.4.1 Testes Unitários

Este tipo de testes foca-se, como o nome indica, em testar unidades do código do sistema. Valida o correto funcionamento de uma pequena porção de código, de acordo com o que se espera que produza.

Estes testes foram implementados para todos os casos de uso e para todas as peças de software envolvidas. Na Figura 67 está um exemplo de um teste feito a uma peça *middleware*, onde se valida a correta criação de um campo de uma estrutura compatível com o recurso FHIR *ServiceRequest*.

```
describe('*** setOccurrence ***', () => {
    it('should corretly set occurrence', () => {
        const result = setOccurrence({}, { codeValue: 'SOME_CODE', systemValue: 'SOME_SYSTEM' })
        expect(result.occurrence.timing.code.coding[0]).toStrictEqual({
            code: { value: 'SOME_CODE' },
            system: { value: 'SOME_SYSTEM' }
        })
    })
    it('should corretly return empty object', () => {
        const result = setOccurrence({}, {})
        expect(result).toMatchObject({})
    })
})
```

Figura 67 - Teste Unitário para alteração do campo Occurrence de estrutura compatível com FHIR ServiceRequest

Na Figura 68 está representado um outro teste que valida a criação de toda a estrutura compatível com o recurso FHIR *ServiceRequest*.

```
describe('*** setServiceRequest ***', () => {
    it('should correctly return service request with new data', () => {
        const data = {
            authoredOn: 'SOME_AUTHORED_ON',
            code: 'SOME_CODE',
            serviceRequestId: '2',
            performer: 'SOME_REQUESTER',
            requester: 'SOME_PERFORMER',
            note: 'SOME_NOTE',
            occurrence: { codeValue: 'SOME_CODE', systemValue: 'SOME_SYSTEM' },
            subject: 'SOME_SUBJECT',
            status: 'SOME_STATUS'
        }
        const result = setServiceRequest(data)
        expect(result.id.value).toBe('2')
        expect(result.note[0].text.value).toBe('SOME_NOTE')
        expect(result.performer[0].device_id.value).toBe('SOME_REQUESTER')
        expect(result.requester.practitioner_id.value).toBe('SOME_PERFORMER')
        expect(result.status.value).toBe('SOME_STATUS')
        expect(result.subject.device_id.value).toBe('SOME_SUBJECT')
        expect(result.occurrence.timing.code.coding[0]).toStrictEqual({
            code: { value: 'SOME_CODE' },
            system: { value: 'SOME_SYSTEM' }
        })
    })
})
```

Figura 68 - Teste Unitário para criação de estrutura compatível com FHIR ServiceRequest

6.4.2 Testes de Aceitação

Os testes de aceitação[76] focam-se na capacidade do sistema cumprir os requisitos do cliente. Assim, descrevem o comportamento que o software deve ter em determinados cenários.

Estes testes foram feitos para todos os casos de uso, validando o comportamento dos mesmos. Isto inclui casos em que os pedidos são respondidos com sucesso, mas também casos em que possa existir alguma falha ou comportamento inesperado (*happy path – sad path*).

Na Figura 69 e na Figura 70 pode ver-se a implementação dos testes de aceitação para o Caso de Uso Consultar Observações, criados com recurso à *framework codeceptJS*[77] e a *mocks* de dados.

[76] Retirado em outubro de 2022 de https://www.agilealliance.org/glossary/acceptance/
[77] Retirado em outubro de 2022 de https://codecept.io/

Relativamente aos casos de insucesso, procurou-se que o sistema desse *feedback* ao utilizador neste sentido, informando-o que existiu um problema. O teste da Figura 69 valida este cenário, através da verificação de que é apresentada de uma mensagem de erro ao utilizador.

```
Given('the user is on the page widget vital signs', () => {
   I.mockGeVitalSignsFailure()
})

Given('the user makes a request API', () => {
   I.amOnPage('/')
})

Then('the user should see the message that it was not possible to make the information available', () => {
   I.seeElement("//div[@data-test='vital-signs-error-small:error']")
   I.stopMock()
})
```

Figura 69 - Teste de aceitação - Sad Path

Para o caso de sucesso (*happy path*), foi validado que as linhas da tabela, representativas dos valores dos resultados de cada sinal vital, eram apresentadas ao utilizador (Figura 70).

```
Given('the user is on the page Widget vital signs', () => {
   I.mockGetVitalSignsResults([
      'HEART_RATE',
      'RESPIRATORY_RATE',
      'OXYGEN_SATURATION',
      'BODY_TEMPERATURE',
      'BLOOD_PRESSURE'
   ])
   I.amOnPage('/')
})

Then('the user will see the results', () => {
   I.waitForElement("//span[@data-test='vital-signs-table-value-0:text']")
   I.seeElement("//span[@data-test='vital-signs-table-value-0:text']")
   I.seeElement("//span[@data-test='vital-signs-table-value-1:text']")
   I.seeElement("//span[@data-test='vital-signs-table-value-2:text']")
   I.seeElement("//span[@data-test='vital-signs-table-value-3:text']")
   I.seeElement("//span[@data-test='vital-signs-table-value-4:text']")
   I.stopMock()
})
```

Figura 70 - Teste de Aceitação - Happy Path

6.4.3 Testes de Integração

Os testes de integração [78] permitem verificar se os módulos independentes de software funcionam corretamente quando comunicam entre si.

Estes testes foram feitos para os vários cenários do projeto com recurso à ferramenta *codeceptJS*, tal como os testes de aceitação.

Um dos testes efetuados (Figura 71) valida que, quando um dispositivo é associado a um paciente no *widget device-search-widget* (Caso de Uso: Associar Dispositivos), aparece de seguida na tabela do *widget associated-devices-widget* (Caso de Uso: Consultar Dispositivos), comprovando que está associado ao paciente.

```
let deviceName = null

When('the user sees Device Search widget', () => {
  I.seeDeviceSearch()
})

When('the user selects and saves the association of a device', async () => {
  deviceName = await I.grabDeviceSearchDeviceName()
  I.click("//span[@data-test='device-search-table-value-1-checkbox:checkbox']")
  I.saveDeviceSearch()
})

When('the user sees a success message', () => {
  I.seeSuccessMessage()
})

When('the user goes to associated devices screen', () => {
  I.seeAssociatedDevices()
})

Then('the user will see the new associated device on the table', () => {
  I.seeDevice(deviceName, "//span[@data-test='vital-signs-table-name-0:text']")
})
```

Figura 71 - Teste de Integração

[78] Retirado em setembro de 2022 de https://martinfowler.com/bliki/IntegrationTest.html

6.4.4 Testes de Carga

Os testes de carga[79] procuram criar simulações de ambientes de produção. Permitem perceber qual o desempenho da aplicação sob diferentes circunstâncias, o que inclui situações em que se coloca uma carga considerável de stress na mesma.

No âmbito do projeto, estes testes foram criados através da ferramenta *Apache JMeter*[80], para simular várias ações de utilizador. Um destes testes simula a consulta de dispositivos disponíveis para associação, referente ao requisito funcional RF1, onde são realizados 1000 pedidos destes em simultâneo.

Inicialmente criou-se uma *Thread Group*, onde se definiram 1000 *threads*. Cada uma destas *threads* simula, em simultâneo, um determinado pedido à aplicação, tal como se fosse um utilizador normal do sistema. (Figura 72)

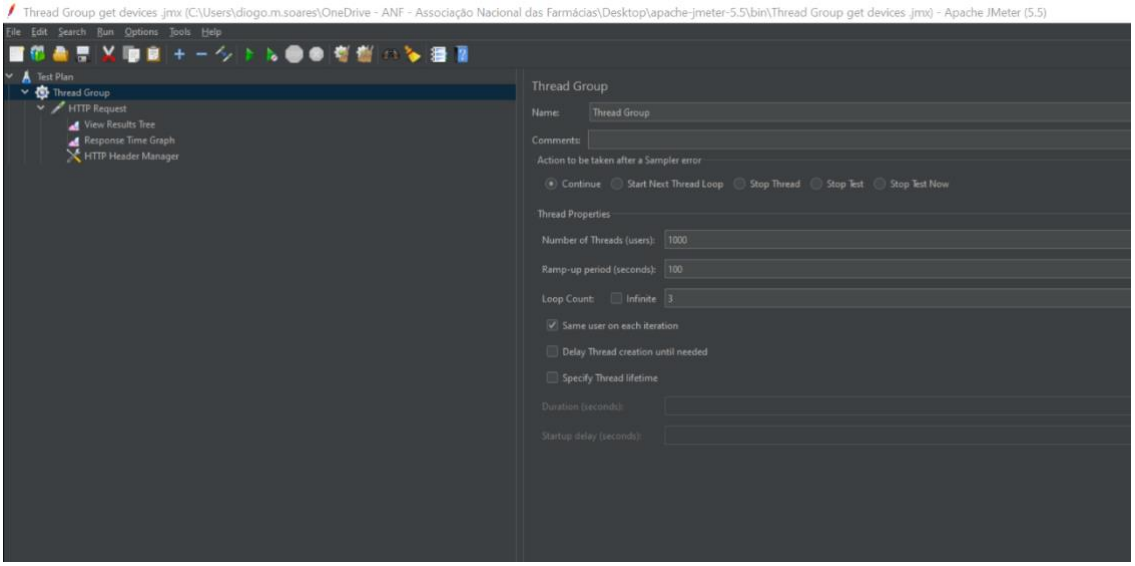

Figura 72 - Teste de Carga - Thread Group (Apache JMeter)

Logo após, definiu-se o pedido, incluindo o protocolo de comunicação, endereço do servidor, corpo do pedido, entre outros. Finalmente, executou-se o teste e obtiveram-se os resultados dos 1000 pedidos executados, com várias informações como latência, tempo de resposta, número de erros e resposta do servidor. Na Figura 73 podem ver-se os dados obtidos para um destes pedidos, escolhido de forma aleatória.

Esta ferramenta permitiu ainda gerar um gráfico relativo à globalidade dos pedidos, mais concretamente aos tempos de resposta dos mesmos no decorrer do tempo. Os resultados indicam que o servidor apresentou as primeiras respostas com um tempo de 600 milissegundos,

[79] Retirado em setembro de 2022 de https://loadninja.com/load-testing/
[80] Retirado em setembro de 2022 de https://jmeter.apache.org/

no entanto este valor desceu rapidamente para cerca de 120 milissegundos, apresentando pequenas oscilações a partir de então (Figura 74).

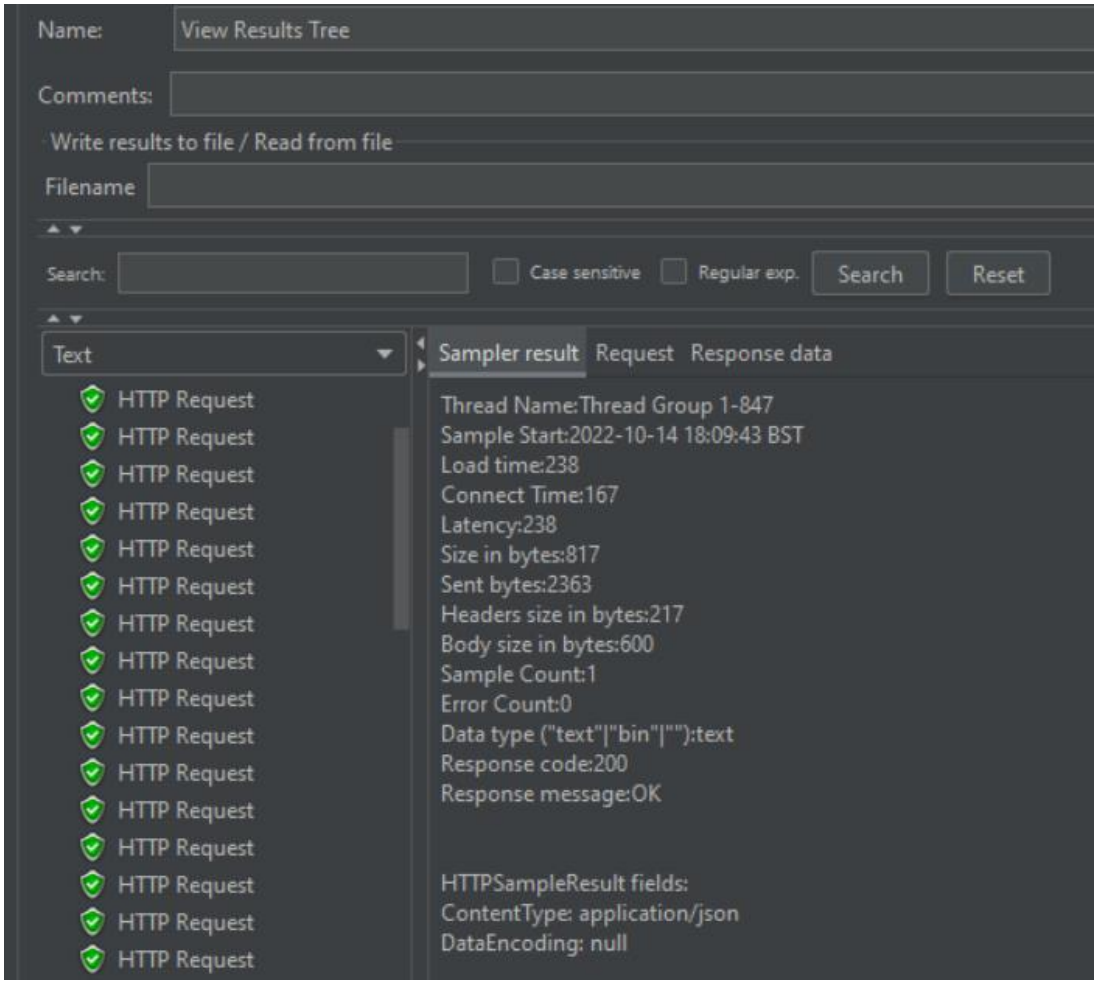

Figura 73 - Teste de carga - Vista em Árvore dos resultados (Apache JMeter)

Os resultados obtidos permitem assim comprovar o cumprimento do requisito não funcional RNF8.

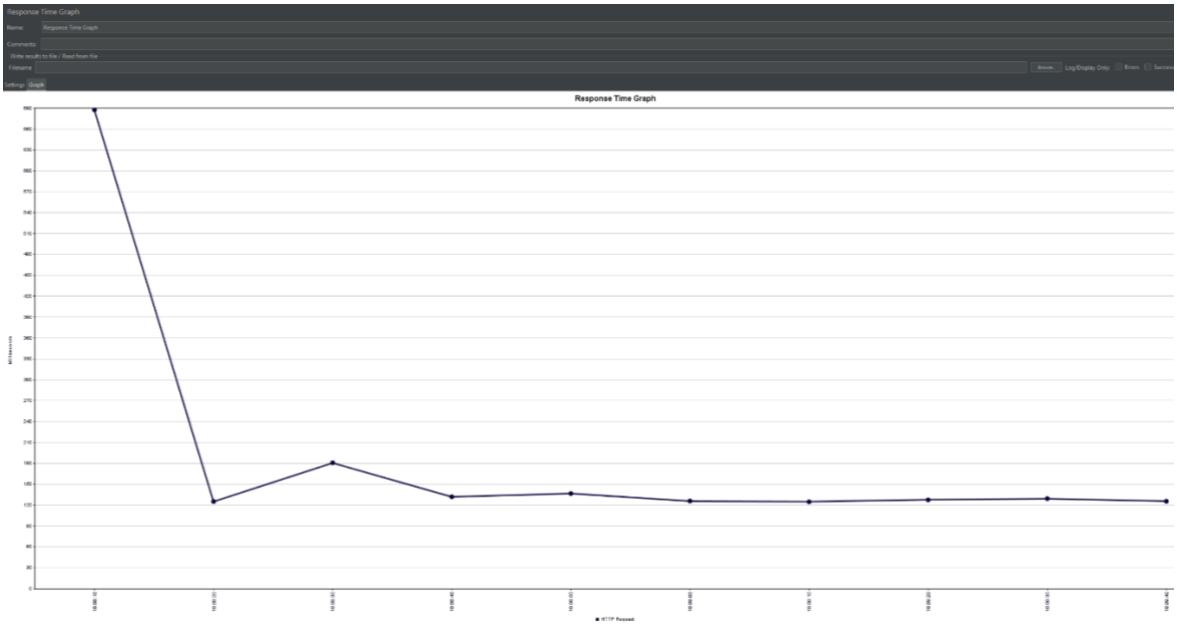

Figura 74 - Teste de carga - Gráfico de tempo de resposta (Apache JMeter)

6.5 Resultados – Interface Gráfica

Nesta secção são apresentados os resultados da implementação relativa a cada caso de uso. Com recurso a figuras, são exibidas as várias interfaces gráficas, referentes a cada *widget*, demonstrando, na medida do possível, os resultados finais.

6.5.1 Caso de Uso: Associar Dispositivos

Para este caso de uso era essencialmente necessária uma interface gráfica que possibilitasse ao profissional de saúde ver todas as *smartboxes* disponíveis para associar ao paciente, permitindo a filtragem pelo nome das mesmas e a associação propriamente dita.

Na Figura 75 apresenta-se o *widget* ***device-search-widget***, responsável pela interface gráfica deste caso de uso. Como se pode ver na imagem, as *smartboxes* livres estão listadas, sendo possível o mecanismo de colapsar e expandir, de forma a ver ou esconder os sensores a elas associados. É também possível a pesquisa pelo nome de determinadas *smartboxes*, através da zona de escrita da parte superior do *widget*, que aplica um filtro à lista e destaca a laranja o texto dos resultados procurados.

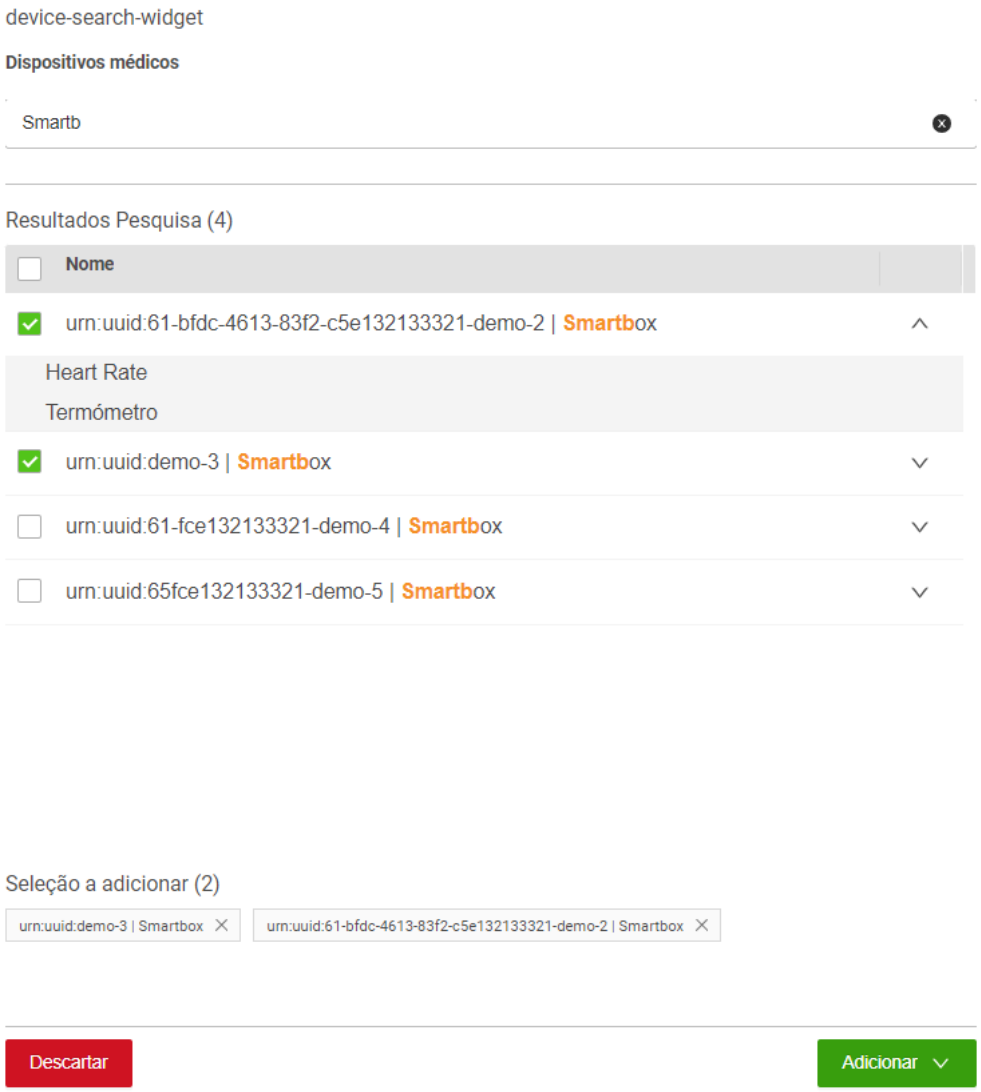

Figura 75 - Associação de smartboxes: device-search-widget

6.5.2 Casos de Uso: Consultar Dispositivos

Em relação a este caso de uso, o objetivo era o de apresentar todos os dispositivos associados a um determinado paciente num dado momento, com informação acerca de cada *smartbox* e respetivos sensores.

Na Figura 76 apresenta-se o *widget* ***associated-devices-widget***, responsável pela interface gráfica deste caso de uso.

De acordo com a imagem, a informação está associada a uma tabela, onde cada linha colapsada corresponde a uma *smartbox*, revelando o número de sensores associados, o estado do dispositivo, assim como a data de associação e o profissional de saúde responsável pela mesma. Adicionalmente, cada linha é expansível, acrescentado informação sobre os sensores da *smartbox*.

associated-devices-widget

Descrição	Nº Dispositivos	Date	Pedido por	Aplicado por	Estado	
Smartbox	4	2021-11-16 | 17:13	John Doe (Médico)	-	ACTIVE	^
Termómetro					ACTIVE	
Monitor de Emergência da Frequência Cardíaca					ACTIVE	
Monitor de Frequência Respiratória					ACTIVE	
Oxímetro de Pulso					ACTIVE	
Smartbox	1	2022-08-26 | 16:02	Luís Saraiva (Enfermeiro)	-	ACTIVE	^
Termómetro					ACTIVE	
Esfigmomanómetro	1	2022-08-26 | 16:04	John Doe (Médico)	-	ACTIVE	^
Termómetro					ACTIVE	

Figura 76 - Consulta de dispositivos do paciente: associated-devices-widget

6.5.3 Caso de Uso: Adicionar Intervenções

Para este caso de uso, o requisito associado consistia em adicionar intervenções a qualquer *smartbox*, ou seja, permitir a calendarização de monitorizações a todos os seus sensores, com definição de recorrência.

Na Figura 77 e na Figura 78 apresenta-se o *widget* ***associated-devices-widget***, responsável pela interface gráfica deste caso de uso.

Na Figura 77 vêm-se as intervenções existentes para a *Smartbox A50* (nome do dispositivo no canto superior esquerdo do ecrã). Estas estão representadas em painéis cinza colapsáveis.

Através do botão transparente “Adicionar”, no canto superior direito do ecrã, é gerado um novo painel, correspondente a uma nova intervenção para monitorização de sinais vitais.

Para cada intervenção pode ser escrita uma nota, no campo observação, e pode ser definido o executante da monitorização. Nesta fase do projeto, o executante será sempre o próprio dispositivo, podendo no futuro ser um profissional de saúde, para intervenções não relacionadas com monitorizações, como aplicação ou remoção dos sensores do paciente.

Adicionalmente, pode ainda ser definida uma recorrência temporal para a monitorização, através de um clique no botão negro “Definir”, que disponibiliza imediatamente ao utilizador um componente para indicar esta periodicidade (Figura 78).

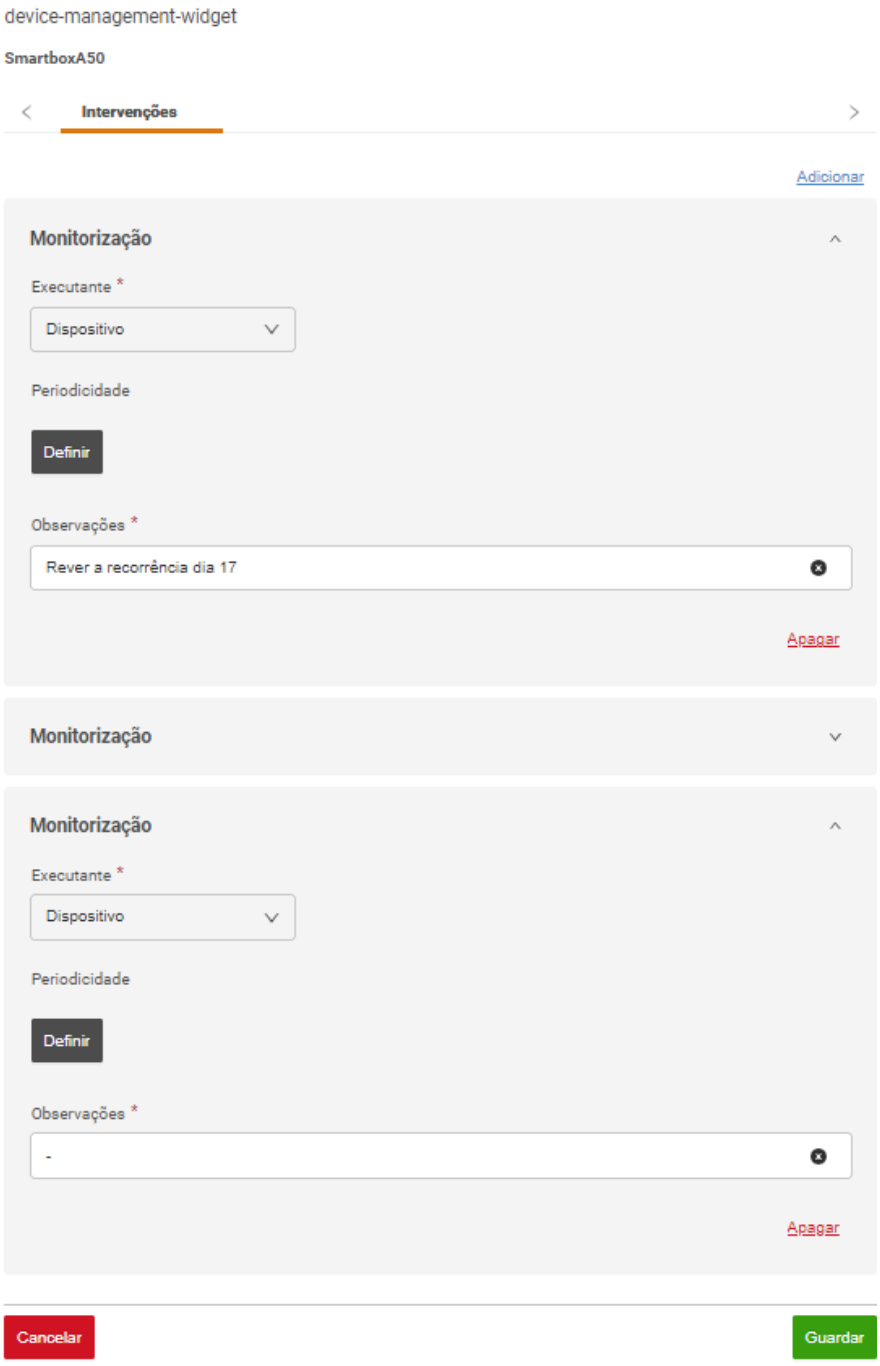

Figura 77 - Adição de Intervenções: device-management-widget

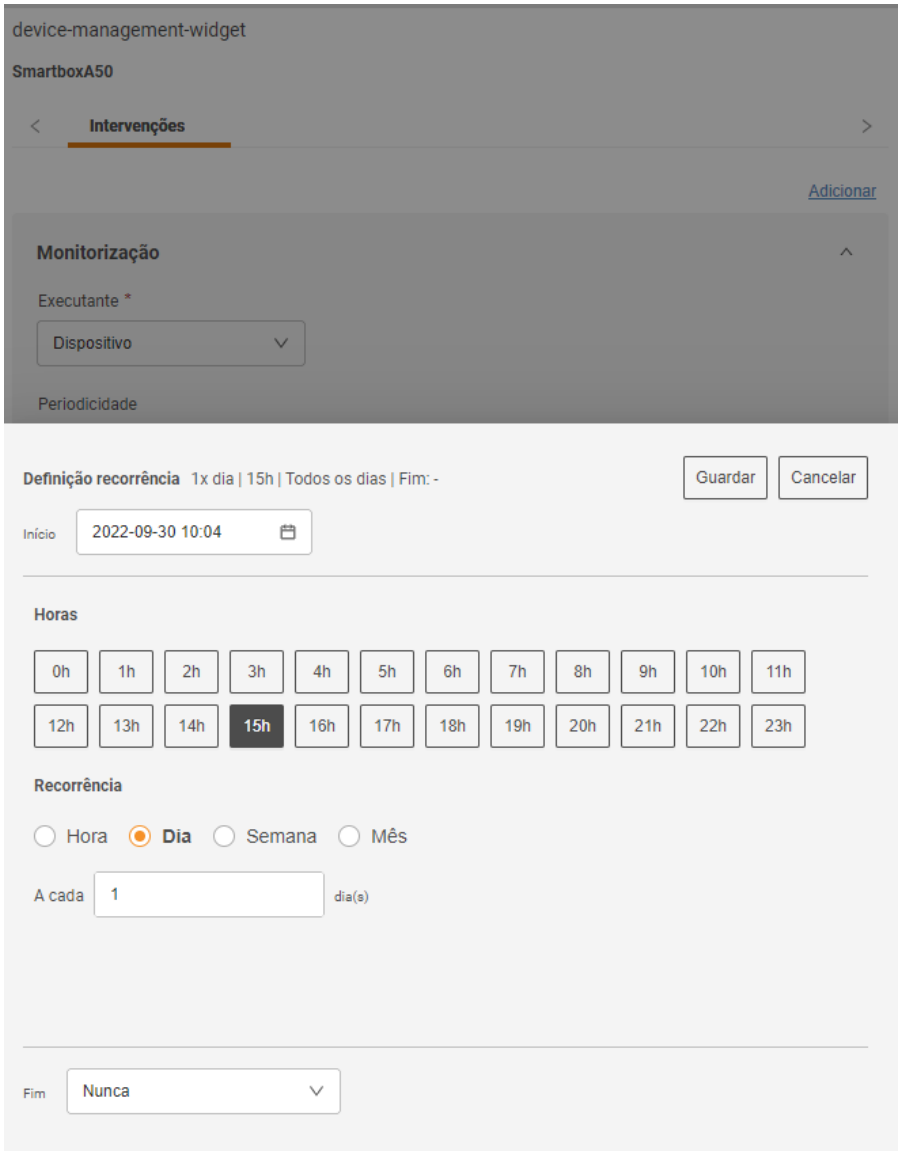

Figura 78 – Definição de recorrência: device-management-widget

6.5.4 Caso de Uso: Consultar Observações

Para este caso de uso era necessária uma interface gráfica que permitisse ao profissional de saúde ver os últimos resultados das monitorizações feitas aos sinais vitais de determinado paciente.

Na Figura 79 apresenta-se o *widget* ***vital-signs-widget***, responsável pela interface gráfica deste caso de uso.

Neste caso, o paciente foi monitorizado para os sinais vitais de Frequência cardíaca, Frequência respiratória, Saturação de Oxigénio, Temperatura e Pressão arterial. Como se pode observar, todos os valores são resultantes de monitorizações feitas a cada hora certa, fruto da definição prévia desta recorrência.

Sinais Vitais

Frequência Cardíaca(bpm)	**63** *06h00*	**66** *07h00*	**65** *08h00*	**73** *09h00*	- -
Frequência Respiratória(cpm)	**13** *06h00*	**13** *07h00*	**12** *08h00*	**15** *09h00*	- -
Saturação de O2(%)	**98** *06h00*	**98** *07h00*	**98** *08h00*	**98** *09h00*	- -
Temperatura(ºC)	**36.1** *06h00*	**36.1** *07h00*	**36.3** *08h00*	**36.5** *09h00*	- -
Pressão Arterial(mmHg)	**120/82** *06h00*	**125/81** *07h00*	**123/81** *08h00*	**132/86** *09h00*	- -

Figura 79 - Consulta de Observações: vital-signs-widget

7 Experimentação e Avaliação

Após o desenvolvimento de uma solução é impreterível averiguar a qualidade da mesma e esclarecer se os requisitos e expetativas iniciais foram satisfeitos na sua totalidade. Desta forma, e para tirar tais conclusões, é realizada uma avaliação detalhada neste capítulo.

7.1 Hipóteses

Tendo em conta o âmbito do projeto e seus objetivos é definida uma única hipótese:

- A solução CPOE permite toda a gestão transversal à área de monitorização de sinais vitais, respondendo corretamente a todos os pedidos do profissional de saúde, com alto grau de qualidade, usabilidade e suportabilidade.

Esta hipótese é assim alvo de avaliação sob os pontos de vista de qualidade e usabilidade. De recordar que esta se baseia no objetivo principal do projeto, definido na Secção 1.3.

7.2 Quantitative Evaluation Framework

O modelo de avaliação *Quantitative Evaluation Framework* (QEF) (Escudeiro *et al.*, 2008) procura parametrizar aspetos que conferem qualidade a uma solução. Para isto, faz uso de uma tríade de conceitos: Dimensão, Fator e Requisito. Estes conceitos relacionam-se, sendo que a Dimensão é composta por Fatores, ao passo que o Fator é constituído por Requisitos.

Cada Requisito é composto pelos seguintes elementos:

- **Relevância** – medida de avaliação do requisito, que pode tomar os valores de 0, 2, 4, 6 8 e 10. Assim atribui um peso ao requisito, condizente com a importância que assume na solução.
- **Avaliação** – medida que corresponde à taxa de incumprimento de determinado requisito. Aqui é usada uma escala percentual, aceitando os valores 0, 25, 50 e 100.

Por sua vez, cada Fator é composto por:

- **Importância** – medida que é o quociente entre a soma das medidas de relevância dos requisitos que constituem o fator e a soma das medidas de relevância dos requisitos de todos os fatores que constituem a dimensão.
- **Cumprimento** – medida obtida através do cálculo da média percentual do grau de avaliação de todos os requisitos do Fator.

Através destes parâmetros, este sistema de avaliação permite uma comparação entre o sistema real e o sistema ideal. Ou seja, avalia com uma métrica objetiva, a distância entre o estado mais atual da solução desenvolvida (sistema real) e o estado inicialmente planeado e pretendido no final do desenvolvimento da solução (sistema ideal).

De forma a iniciar este processo de avaliação é necessário definir as Dimensões. Para o âmbito deste projeto, foram escolhidas as dimensões **Funcionalidade**, **Usabilidade** e **Suportabilidade**, de acordo com a Hipótese definida na Secção 7.1

7.2.1 Metodologia de Avaliação

Dado que a avaliação deve ser realizada nas dimensões de funcionalidade, suportabilidade e usabilidade, têm de ser definidas as metodologias a ser seguidas.

Para o efeito, tomou-se a decisão de utilizar métricas definidas e avaliadas pelo autor.

Embora tenha sido considerada uma metodologia diferente para avaliar a Dimensão Usabilidade, através de um questionário a um grupo de utilizadores, esta foi descartada devido a algumas condicionantes. Através do questionário seria possível representar os requisitos do QEF através de perguntas, com as respostas a assumir o papel de medida de Avaliação. No entanto, por questões de confidencialidade e estratégicas da Glintt-HS, e por limitações ao nível da disponibilidade dos dispositivos de monitorização, não foi possível optar por esta metodologia para avaliar esta Dimensão.

7.2.2 Dimensões

Para a Dimensão de **Suportabilidade**, optou-se por escolher um conjunto de requisitos associados aos Fatores de Adaptabilidade e Manutibilidade, como está expresso na Figura 80.

O peso do Fator Navegação é bastante superior ao de Linguagem dado ser mais relevante ter a aplicação completamente apta para apenas uma linguagem, do que ter a aplicação com falhas, ainda que disponível para um maior grupo de clientes.

A escala de avaliação para cada um dos Fatores desta Dimensão encontra-se no Apêndice B.1.

Dimensão	Fator	Wij (Peso do fator)	Rwjk (Relevância do Requisito no Fator)	Requisito
Suportabilidade	Navegação	0,85	8	UN1 - A navegação entre funcionalidades/ widgets deve ser simples e intuitiva
			10	UN2 - Utilizador recebe feedback após ações
			8	UN3 - Tempos de carregamento dos widgets e processamentos de ações devem ser rápidos (< 3 segundos)
	Linguagem	0,15	6	UL1 - Está disponível nos idiomas Português e Inglês
			10	UL2 - Terminologia é adequada e agnóstica a FHIR

Figura 80 - Dimensão Suportabilidade

Para a Dimensão de **Usabilidade** foram selecionados os Fatores Adaptabilidade e Manutibilidade. Neste caso, existe um maior equilíbrio entre os pesos dos dois Fatores, como demonstrado na Figura 81.

A escala de avaliação para cada um dos Fatores desta Dimensão encontra-se no Apêndice B.2.

Dimensão	Fator	Wij (Peso do fator)	Rwjk (Relevância do Requisito no Fator)	Requisito
Usabilidade	Adaptabilidade	0,43	10	SA1 - A aplicação está disponível em diferentes browsers
			10	SA2 - A implementação e arquitetura não causam dificuldades na adição de novas funcionalidades
	Manutibilidade	0,57	10	SM1 - A implementação de novas funcionalidades não terá impacto negativo nas já existentes
			6	SM2 - A arquitetura e detalhes técnicos de implementação estão documentados
			8	SM3 - A implementação não afetou a aplicação Viewer

Figura 81 - Dimensão Usabilidade

A última Dimensão selecionada foi a de **Funcionalidade** (Figura 82). Os Fatores envolvidos foram os de Integridade e Tratamento de dados, Notificações e Profissional de Saúde.
Este último tem o maior peso, com maioria absoluta relativamente aos restantes. Esta valorização deve-se ao facto dos requisitos a si associados estarem intimamente ligados aos casos de uso do projeto, representando as funcionalidades ao nível mais macro.

A escala de avaliação para cada um dos Fatores desta Dimensão encontra-se no Apêndice B.3.

Dimensão	Fator	Wij (Peso do fator)	Rwjk (Relevância do Requisito no Fator)	Requisito
Funcionalidade	Integridade e Tratamento de dados	0,25	10	FITD1 - O sistema garante a autenticidade e integridade de todos os dados.
			10	FITD2 - O sistema garante a segurança e privacidade de todos os dados dos pacientes.
	Notificações	0,13	8	FN1 - O utilizador é notificado quando associa dispositivos ao paciente, em caso de sucesso ou insucesso
			10	FN2 - O utilizador é notificado quando elimina uma intervenção de um dispositivo, em caso de sucesso ou insucesso
			8	FN3 - O utilizador é notificado quando edita uma intervenção de um dispositivo, em caso de sucesso ou insucesso
			10	FN4 - O utilizador é notificado quando adiciona uma intervenção de um dispositivo, em caso de sucesso ou insucesso
	Profissional de Saúde	0,62	10	FPS1 - O sistema permite a listagem de dispositivos livres para associação.
			10	FPS2 - O sistema permite a associação de dispositivos ao paciente.
			6	FPS3 - O sistema permite a listagem dos dispositivos associados ao paciente.
			10	FPS4 - O sistema permite a adição e calendarização de intervenções para dispositivos
			8	FPS5 - O sistema permite aconsulta dos resultados obtidos nas monitorizações dos sinais vitais do paciente

Figura 82 - Dimensão Funcionalidade

7.2.3 Resultados

Os resultados obtidos para as três Dimensões, com identificação de cumprimento por requisito e média ponderada de cumprimento por Fator, estão indicados na Figura 83.

Através dos dados obtidos por avaliação e calculados para as médias ponderadas da Figura 83, é também possível calcular as percentagens de cumprimento para cada uma das Dimensões:

- **Funcionalidade**: 100%
- **Suportabilidade**: 84,1145%
- **Usabilidade**: 92,875%

De ressalvar que todos os 11 requisitos associados aos Fatores da Dimensão **Funcionalidade** foram avaliados no valor percentual máximo, sendo esta Dimensão a que merece maior destaque pela positiva.

Já a Dimensão **Suportabilidade** ficou um pouco aquém das expetativas. Ambos os Fatores apresentam taxas de cumprimento inferiores a 100%. De destacar pela negativa os requisitos UN1, UN3 e UL1.

Relativamente ao UN1, com a sua taxa de cumprimento avaliada em 75%, considera-se que a navegação poderia ser um pouco mais intuitiva, utilizando-se por exemplo um menu ou uma central para acesso direto a todos os *widgets* e funcionalidades.

O Requisito UN3 tem uma taxa de cumprimento de 75% dada alguma inconsistência nos tempos de resposta, que pode ter diversos motivos, relacionados com a solução ou até com a aplicação *Viewer*.

Por fim, o requisito UL1, justifica a taxa de cumprimento de 50% dado que apenas o idioma português estava disponível no término do projeto. No entanto, implementou-se parcialmente a interface gráfica em inglês, mas esse é um trabalho ainda em progresso, à data da escrita (outubro de 2022).

Quanto à Dimensão **Usabilidade**, o destaque vai para o Fator Manutibilidade, onde se encontra o único requisito da Dimensão com uma taxa de cumprimento inferior a 100%, mais concretamente de 50%.

O Requisito referido é o SM2 e está relacionado com a documentação criada para a solução. Apesar desta dissertação documentar a arquitetura e implementação, existem detalhes de índole mais técnica e informação útil às equipas de desenvolvimento do *Viewer* que acabaram por não ser documentados em tempo útil.

Ainda assim, existe um baixo impacto deste requisito na média ponderada de cumprimento, dado que a sua Relevância é apenas de 6.

Dimensão	Fator	Wij (Peso do fator)	Rwjk (Relevância do Requisito no Fator)	Requisito	Cumprimento	Média Ponderada de cumprimento
Funcionalidade	Integridade e Tratamento de dados	0,25	10	FITD1 - O sistema garante a autenticidade e integridade de todos os dados.	100%	100%
			10	FITD2 - O sistema garante a segurança e privacidade de todos os dados dos pacientes.	100%	
	Notificações	0,13	8	FN1 - O utilizador é notificado quando associa dispositivos ao paciente, em caso de sucesso ou insucesso	100%	100%
			10	FN2 - O utilizador é notificado quando elimina uma intervenção de um dispositivo, em caso de sucesso ou insucesso	100%	
			8	FN3 - O utilizador é notificado quando edita uma intervenção de um dispositivo, em caso de sucesso ou insucesso	100%	
			10	FN4 - O utilizador é notificado quando adiciona uma intervenção de um dispositivo, em caso de sucesso ou insucesso	100%	
	Profissional de Saúde	0,62	10	FPS1 - O sistema permite a listagem de dispositivos livres para associação.	100%	100%
			10	FPS2 - O sistema permite a associação de dispositivos ao paciente.	100%	
			6	FPS3 - O sistema permite a listagem dos dispositivos associados ao paciente.	100%	
			10	FPS4 - O sistema permite a adição e calendarização de intervenções para dispositivos	100%	
			8	FPS5 - O sistema permite aconsulta dos resultados obtidos nas monitorizações dos sinais vitais do paciente	100%	
Suportabilidade	Navegação	0,85	8	UN1 - A navegação entre funcionalidades/ widgets deve ser simples e intuitiva	75%	84,62%
			10	UN2 - Utilizador recebe feedback após ações	100%	
			8	UN3 - Tempos de carregamento dos widgets e processamentos de ações devem ser rápidos (< 3 segundos)	75%	
	Linguagem	0,15	6	UL1 - Está disponível nos idiomas Português e Inglês	50%	81,25%
			10	UL2 - Terminologia é adequada e agnóstica a FHIR	100%	
Usabilidade	Adaptabilidade	0,43	10	SA1 - A aplicação está disponível em diferentes browsers	100%	100%
			10	SA2 - A implementação e arquitetura não causam dificuldades na adição de novas funcionalidades	100%	
	Manutibilidade	0,57	10	SM1 - A implementação de novas funcionalidades não terá impacto negativo nas já existentes	100%	87,50%
			6	SM2 - A arquitetura e detalhes técnicos de implementação estão documentados	50%	
			8	SM3 - A implementação não afetou a aplicação Viewer	100%	

Figura 83 - Resultados QEF

Através dos valores percentuais calculados para cada uma das Dimensões, é calculada a qualidade do sistema real:

$$\text{qualidade} = \frac{\log\left(1+\frac{100}{100}\right) + \log\left(1+\frac{84.1145}{100}\right) + \log\left(1+\frac{92.875}{100}\right)}{3 * \log(2)} = 0{,}942836$$

Conclui-se assim que a qualidade do sistema real é cerca de 94%.

Este valor poderia ser mais alto, no entanto algumas falhas nas Dimensões de Suportabilidade e Usabilidade não o permitiram. Ainda assim, o autor considera esta avaliação da solução satisfatória. Além disto, considera ainda que a hipótese colocada foi cumprida.

8 Conclusões

Após o término do projeto, segue-se a fase de retirar conclusões acerca de todo o trabalho desenvolvido. Assim, este capítulo começa por fazer uma reflexão sobre o cumprimento dos objetivos previamente definidos. De seguida, são referidas limitações do projeto e perspetivas de trabalho futuro que levem à melhoria e crescimento do mesmo. Para finalizar, é feita uma apreciação final de todo o trabalho efetuado no âmbito desta dissertação.

8.1 Objetivos e Contributos

No capítulo Introdução, na Secção 1.3, foram definidos os objetivos e contributos esperados para o projeto. Nesta secção torna-se pertinente averiguar o cumprimento dos mesmos, em forma de retrospetiva.

Relembrando o leitor, o objetivo definido para o projeto foi o de conceptualizar e desenvolver uma solução MVP de um CPOE para a área de Monitorização de Sinais Vitais. Neste sentido, esta dissertação, no seu todo, documenta os contributos para este objetivo.

É importante também referir as tarefas adjacentes ao projeto, que foram fundamentais para atingir o objetivo proposto. O levantamento dos requisitos foi de extrema importância, assim como o estudo do CPOE e da área de Sinais Vitais e sua monitorização.

Outra tarefa fundamental foi a sistematização de protocolos *standard* de comunicação de *Healthcare*. Todo o trabalho está assente num destes protocolos, o FHIR, e como evidenciado ao longo da fase de implementação, tem um papel fundamental, central e transversal a todo o desenvolvimento, incluindo do ponto de vista de negócio.

Foram sistematizadas abordagens distintas para resolver o problema, através da conceptualização a nível tecnológico e arquitetural e foram estudadas alternativas que valorizaram as opções tomadas, através da tarefa de desenho da solução.

Por fim, as tarefas de implementação e avaliação da solução foram detalhadamente explicadas, denotando-se as evidências de todo o trabalho realizado.

Relativamente aos contributos, é de registar o seu cumprimento. De facto, a solução Viewer saiu beneficiada com este projeto, com um incremento do seu valor. Por outro lado, salienta-se a implementação associada à Monitorização de Sinais Vitais no CPOE, dado esta ser uma área extraordinária ao comum neste tipo de sistemas.

8.2 Limitações e Trabalho Futuro

Apesar de, de uma forma geral, as expetativas para o projeto terem sido cumpridas, há um conjunto de aspetos a melhorar e outros onde são encontradas limitações, mais proeminentes nos requisitos não funcionais.

A aplicação *Viewer* já lida com as temáticas relacionadas com os requisitos não funcionais RNF3 (políticas de privacidade e RGPD), e RNF7 (integridade de dados de intervenções), pelo que na realidade não foi necessário desenvolver esforços neste sentido.

O RNF10, relacionado com a interoperabilidade entre dispositivos e entidades externas, acabou por também apresentar limitações. A dependência de terceiros limitou integrações externas para a comunicação da solução com os dispositivos de monitorização e *gateways*. Assim, não foi possível testar o software em tempo útil numa situação real, com pacientes em contexto hospitalar.

Em relação a melhorias e trabalho futuro, dado que a solução é um MVP, identificam-se vários pontos relevantes:

- **Áreas CPOE** – Existem diversas áreas que podem e devem ser acrescentadas ao CPOE. No futuro este poderá ser um software que dê resposta a dezenas de áreas. É uma meta ambiciosa e exigente, no entanto este produto só assim poderá ganhar maior relevância e perder o estatuto de MVP.
- **Monitorização de Sinais Vitais** – Mesmo para esta área, em foco nesta fase primordial do CPOE, é identificado trabalho que pode ser feito de forma a acrescentar valor à solução. Como mencionado previamente, seria interessante ter uma interface gráfica com a função de ser uma central para fácil e rápido acesso a todas as funcionalidades preconizadas. Por outro lado, ao realizar a adição de intervenções, todos os sensores associados à *smartbox* recebem as instruções para realizar a monitorização. Assim, acrescentaria valor se o utilizador pudesse escolher quais os sensores envolvidos, pelo que seria relevante trabalhar nesse sentido.

8.3 Apreciação Final

Para o autor este foi um projeto extremamente interessante. A nível académico e a nível profissional adquiriu uma vasta quantidade de conhecimentos, fruto de todo o trabalho desenvolvido.

Quando foi escolhido o projeto, o próprio conceito de CPOE era uma novidade para o autor. Toda a informação recolhida durante o levamento teórico do estado da arte de CPOE foi uma descoberta, tornando todo o esforço mais aprazível e recompensador.

Em relação à área de monitorização de sinais vitais, este era um tema previamente conhecido pelo autor e representava até um campo de interesse. Ainda assim, os conhecimentos nesta área foram expandidos.

Relativamente ao estudo e comparação de trabalhos relacionados, tecnologias e até questões de índole mais técnica, como o desenho arquitetural, é de destacar a sua enorme importância no ponto de vista do autor. Caso este fosse um projeto meramente profissional e não académico, estes aspetos teriam possivelmente sido alvo de menor escrutínio. No entanto, e tendo em conta as circunstâncias, apelaram ao sentido crítico, à idealização de alternativas para solução e à reflexão para fazer as escolhas arquiteturais mais acertadas. Este processo foi importante para compreender que não existem alternativas boas ou más, mas sim mais ou menos adequadas ao propósito que servem, onde existem sempre *trade-offs*.

No que concerne à fase de implementação, o autor destaca o ponto de vista profissional. As várias áreas de desenvolvimento a que esteve sujeito obrigaram a que o autor saísse de algumas zonas de conforto, confrontando-se com áreas em que tinha pouca experiência. Este processo resultou em aprendizagem e em crescimento profissional.

A nível académico, o autor reconhece a importância da documentação da fase de implementação, que permite evidenciar o trabalho desenvolvido e explicar alguns conceitos técnicos. No entanto, considera que representa conteúdo com menor valor intrínseco quando comparado com o estudo do estado da arte ou a fase de desenho arquitetural.

Destaca-se também a aplicação da metodologia de avaliação QEF, desconhecida *à priori* pelo autor. A utilização do QEF como parâmetro de avaliação permitiu uma apreciação global do projeto e produziu resultados bastante satisfatórios.

Em suma, o projeto foi exigente, mas contribuiu em larga escala para o desenvolvimento do autor a nível académico e profissional, além de ter cumprido com as necessidades e expetativas iniciais.

Referências

Aggarwal, S. (2018) 'Modern Web-Development Using ReactJS | Document Object Model | Model–View–Controller', *International Journal of Recent Research Aspects*, 5(1), pp. 133–137.

Aghazadeh, S., Aliyev, A. Q. and Ebrahimnejad, M. (2011) 'The role of computerizing physician orders entry (CPOE) and implementing decision support system (CDSS) for decreasing medical errors', *2011 5th International Conference on Application of Information and Communication Technologies, AICT 2011*, pp. 4–6. doi: 10.1109/ICAICT.2011.6110916.

Ahmad, R. and Salama, K. N. (2018) 'Physical Sensors for Biomedical Applications', *Proceedings of IEEE Sensors*, 2018-October. doi: 10.1109/ICSENS.2018.8589646.

Alencar, G. A. *et al.* (2019) 'Non-functional requirements in health information systems: A systematic mapping research', *Iberian Conference on Information Systems and Technologies, CISTI*, 2019-June. doi: 10.23919/CISTI.2019.8760720.

Baysari, M. T. *et al.* (2017) 'Longitudinal study of user experiences of a CPOE system in a pediatric hospital', *International Journal of Medical Informatics*, 109, pp. 5–14. doi: 10.1016/j.ijmedinf.2017.10.018.

Cappon, G. *et al.* (2019) 'Continuous glucose monitoring sensors for diabetes management: A review of technologies and applications', *Diabetes and Metabolism Journal*, 43(4), pp. 383–397. doi: 10.4093/DMJ.2019.0121.

Danylenko, A. and Löwe, W. (2012) 'Context-aware recommender systems for non-functional requirements', *2012 3rd International Workshop on Recommendation Systems for Software Engineering, RSSE 2012 - Proceedings*, pp. 80–84. doi: 10.1109/RSSE.2012.6233417.

Dhamanti, I. *et al.* (2021) 'Implementation of Computerized Physician Order Entry in Primary Care: A Scoping Review'. doi: 10.2147/JMDH.S344781.

Diaz, T., Olmedo, F. and Tanter, E. (2020) 'A mechanized formalization of GraphQL', *CPP 2020 - Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, co-located with POPL 2020*, pp. 201–214. doi: 10.1145/3372885.3373822.

Escudeiro, P. *et al.* (2008) 'Quantitative Evaluation Framework (QEF)', *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, 176 LNICST, pp. 117–124. doi: 10.1007/978-3-319-51055-2_15.

Gregório, J. *et al.* (2021) 'The role of Design Science Research Methodology in developing pharmacy eHealth services', in *Research in Social and Administrative Pharmacy*. Elsevier, pp. 2089–2096. doi: 10.1016/J.SAPHARM.2021.05.016.

Griffon, N. *et al.* (2017) 'Physician satisfaction with transition from CPOE to paper-based prescription', *International Journal of Medical Informatics*, 103, pp. 42–48. doi: 10.1016/j.ijmedinf.2017.04.007.

Jakupovic, A., Pavlic, M. and Candrlic, S. (2010) 'Application of analytic hierarchy process (AHP) to measure the complexity of the business sector and business software', *ACM International Conference Proceeding Series*, pp. 35–42. doi: 10.1145/1822327.1822332.

Jungreithmayr, V. *et al.* (2020) 'The impact of a computerized physician order entry system implementation on 20 different criteria of medication documentation-a before-and-after study'. doi: 10.1186/s12911-021-01607-6.

Kellett, J. and Sebat, F. (2017) 'Make vital signs great again – A call for action', *European Journal of Internal Medicine*, 45, pp. 13–19. doi: 10.1016/J.EJIM.2017.09.018.

Khanna, R. and Yen, T. (2014) 'Improving Health Care Quality: Reviews Computerized Physician Order Entry: Promise, Perils, and Experience'. doi: 10.1177/1941874413495701.

Khurana, A. and Rosenthal, S. R. (1998) 'Towards holistic "front ends" in new product development', *Journal of Product Innovation Management*, 15(1), pp. 57–74. doi: 10.1016/S0737-6782(97)00066-0.

Kim, J. *et al.* (2019) 'Wearable biosensors for healthcare monitoring', *Nature Biotechnology*, 37(4), pp. 389–406. doi: 10.1038/S41587-019-0045-Y.

Koen, P. *et al.* (2001) 'Providing clarity and a common language to the "fuzzy front end"', *Research Technology Management*, 44(2), pp. 46–55. doi: 10.1080/08956308.2001.11671418.

Korb-Savoldelli, V. *et al.* (2018) 'Prevalence of computerized physician order entry systems–related medication prescription errors: A systematic review', *International Journal of Medical Informatics*, 111, pp. 112–122. doi: 10.1016/J.IJMEDINF.2017.12.022.

Kruchten, P. B. (1995) 'The 4+1 View Model of Architecture', *IEEE Software*, 12(6), pp. 42–50. doi: 10.1109/52.469759.

Kumar, A. and Singh, R. K. (2016) 'Comparative analysis of angularjs and reactjs', *International Journal of Latest Trends in Engineering and Technology*, 7(4), pp. 225–227. doi: 10.21172/1.74.030.

Li, R. T. *et al.* (2016) 'Wearable Performance Devices in Sports Medicine'. doi: 10.1177/1941738115616917.

Li, Y., Chen, W. and Lu, L. (2021) 'Wearable and Biodegradable Sensors for Human Health Monitoring', *ACS Applied Bio Materials*, 4(1), pp. 122–139. doi: 10.1021/ACSABM.0C00859.

Osterwalder, A. *et al.* (2014) *Value Proposition Design: How to Create Products and Services Customers Want*.

Rahman, M. A. *et al.* (2019) 'Classifying non-functional requirements using RNN variants for quality software development', *MaLTeSQuE 2019 - Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, co-located with ESEC/FSE 2019*, pp. 25–30. doi: 10.1145/3340482.3342745.

Rouayroux, N. *et al.* (2019) 'Medication prescribing errors: a pre- and post-computerized physician order entry retrospective study', *International Journal of Clinical Pharmacy*, 41(1), pp. 228–236. doi: 10.1007/S11096-018-0747-0.

Saaty, T. L. (1980) *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*.

Saaty, T. L. (2008) 'Decision making with the analytic hierarchy process', *International Journal of Services Sciences*, 1(1), pp. 83–98.

Salvaneschi, P. (2010) 'Safety testing of computerized provider order entry systems', *Proceedings - International Conference on Software Engineering*, pp. 74–82. doi: 10.1145/1809085.1809095.

Salvo, P. *et al.* (2018) 'A wearable sweat rate sensor to monitor the athletes' performance during training', *Science and Sports*, 33(2), pp. e51–e58. doi: 10.1016/J.SCISPO.2017.03.009.

Sapra, A., Malik, A. and Bhandari, P. (2022) 'Vital Sign Assessment', *StatPearls*. Available at: https://www.ncbi.nlm.nih.gov/books/NBK553213/ (Accessed: 31 July 2022).

Seabra, M., Nazário, M. F. and Pinto, G. (2019) 'REST or GraphQL? A performance comparative study', *ACM International Conference Proceeding Series*, pp. 123–132. doi: 10.1145/3357141.3357149.

Subramanian, S. *et al.* (2007) 'Computerized physician order entry with clinical decision support in long-term care facilities: Costs and benefits to stakeholders', *Journal of the American Geriatrics Society*, 55(9), pp. 1451–1457. doi: 10.1111/J.1532-5415.2007.01304.X.

Tavakoli, M., Benussi, C. and Lourenco, J. L. (2017) 'Single channel surface EMG control of advanced prosthetic hands: A simple, low cost and efficient approach', *Expert Systems with Applications*, 79, pp. 322–332. doi: 10.1016/J.ESWA.2017.03.012.

Ulaga, W. and Eggert, A. (2006) 'Relationship value and relationship quality: Broadening the nomological network of business-to-business relationships', *European Journal of Marketing*, 40(3–4), pp. 311–327. doi: 10.1108/03090560610648075.

Walters, D. and Lancaster, G. (2000) 'Implementing value strategy through the value chain', *Management Decision*, 38(3), pp. 160–178. doi: 10.1108/EUM0000000005344.

Wang, B. and Liu, X. (2019) *The Implementation and Effects of Computerized Physician Order The Implementation and Effects of Computerized Physician Order Entry in Healthcare Settings Entry in Healthcare Settings Recommended Citation Recommended Citation THE IMPLEMENTATION AND EFFECT*.

Woodall, T. (2003) 'Conceptualising "value for the customer": an attributional, structural and dispositional analysis', *Academy of Marketing Science Review*, 2003(12).

Yasri, S. and Wiwanitkit, V. (2022) 'Sustainable materials and COVID-19 detection biosensor: A brief review', *Sensors International*, 3, p. 100171. doi: 10.1016/J.SINTL.2022.100171.

Apêndice A

Análise de Valor

A análise de valor é um conjunto de técnicas que procuram identificar, quantificar e retificar fraquezas em produtos e processos através de uma abordagem organizada com o intuito de melhorar a rentabilidade dos mesmos.

O Conceito de que o valor é subjetivo e implica que não há uma forma perfeita de o medir de forma consistente e eficiente. Isto deve-se em parte à subjetividade a ele inerente e prévia à associação clara entre o valor e o propósito que deve servir. Relativamente ao propósito, será possível definir os benefícios e custos associados e assim fazer um cálculo do valor, balanceando estes dois aspetos, tendo em conta que os benefícios acrescentam valor e os custos retiram valor.

A análise de valor surge com o objetivo principal de permitir o aumento do valor com o menor custo e sacrifício de qualidade possível.

A.1 Processo de Inovação

O processo de inovação – ver Figura 84 – divide-se em 3 partes:

- Fuzzy Front End (FFE)
- New Product Development (NPD)
- Comercialização

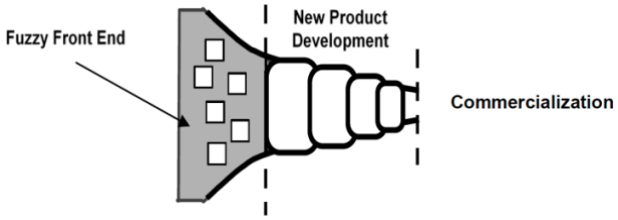

Figura 84 - Processo de inovação

O FFE tem uma natureza de trabalho experimental, pode ser incerta e imprevisível, altamente especulativa e com um caráter informal. O NPD contrasta com o FFE, apresentando-se com uma natureza de trabalho disciplinada, com objetivos definidos e com um plano a seguir. Tem também um grau elevado de previsibilidade e de certeza, sendo um processo formal.

O FFE refere-se às ações no início do processo de desenvolvimento, quando o produto ou serviço a que se aponta ainda não está bem decidido e onde possíveis alterações ao mesmo

ainda não teriam impacto no custo. A visão convencional diz que que o FFE é concluído quando a unidade de negócio se compromete ao financiamento e lança o projeto de NPD, ou, pelo contrário, decide não o fazer, desistindo da ideia que levou ao início do processo de inovação. Esta decisão é normalmente tomada após avaliações do produto/ serviço quanto ao seu retorno financeiro potencial, baseado em estimativas de potencial de mercado, necessidades do mercado e requisitos de recursos, e a sua ligação à estratégia de produto existente (Khurana and Rosenthal, 1998) .

O New Concept Development (NCD), descrito na Subsecção A.1.1, tem como propósito traduzir o FFE para uma linguagem comum e estruturada.

A.1.1 New Concept Development (NCD)

O NCD foi elaborado de forma coletiva por 8 empresas membro da *Industrial Research* Institute. Este modelo surgiu aquando da necessidade destas empresas determinarem as melhores práticas do FFE. Dada a natureza informal do FFE, onde não existe uma linguagem comum ou uma definição consistente dos seus elementos chaves, foi definido o NCD (Koen *et al.*, 2001).

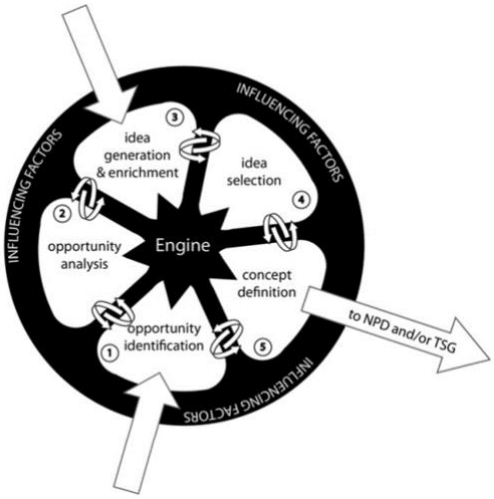

Figura 85 - New Concept Development Model (Koen *et al.*, 2001)

O modelo NCD é constituído por 3 elementos-chave, como se pode ver na Figura 85:

- **Elementos Chave** – representados pelas 5 fases de atividade:
    1. Identificação de Oportunidade
    2. Análise da Oportunidade
    3. Génese de Ideias
    4. Seleção de Ideias
    5. Conceito e desenvolvimento tecnológico
- **Engine** ou **"bull's eyes"** – representa o motor dos elementos chave e é alimentado pela liderança e cultura da organização.

- **Fatores influenciadores –** fatores externos que podem influenciar os elementos-chave e o *engine*.

A.1.1.1 Identificação de Oportunidade

A organização identifica oportunidades que lhes possam acrescentar valor e a que vai dar resposta. Esta identificação pode ser feita através de diferentes formas, como *brainstorms*, análises de tendências, *road mappings*, e pesquisas de mercado (Koen *et al.*, 2001).

Contextualizando com o tema do documento, foi encontrada uma oportunidade de desenvolvimento de um CPOE no contexto de uma aplicação diferenciada em vários aspetos – ver Figura 86 - como apresentar uma visão 30 segundos, onde o médico em 30 segundos tem acesso a toda a informação relevante do doente sem que seja necessário mudar de ecrã ou realizar várias operações. Além disto, um fator de destaque e onde se encontra outra grande oportunidade é o facto desta solução procurar ser agregadora de diferentes fontes externas, centralizando toda a informação. Assim, os utilizadores não necessitam de aceder a vários softwares e poupam mais tempo no processo, ao contrário do que é comum no setor.

O desenvolvimento do CPOE encaixa nesta visão: agilizar os processos de prescrição e pedidos por parte dos profissionais de saúdem, reduzindo o esforço e tempo necessário nestas operações e agregando informação e pedidos de requisição de áreas diferentes, centralizando-as.

Dada a oportunidade, segue-se a análise.

Figura 86 - Diagrama Visão 30 Segundos – Solução Viewer[81]

A.1.1.2 Análise de Oportunidade

Após a identificação da oportunidade é preciso trabalhar na direção de avaliar e estudar a oportunidade, perceber como esta se insere no negócio, de que forma a organização se revê nela e como poderá fazer parte da sua estratégia. É necessária também informação adicional para traduzir a identificação de oportunidade para o negócio e oportunidades tecnológicas.

[81] Retirado em fevereiro de 2022 da documentação interna da organização

Esta análise pode ser suportada também com grupos de foco, estudos de mercado e/ ou experimentações científicas. As incertezas tecnológicas e de mercado vão, no entanto, permanecer quando finalizada esta análise (Koen *et al.*, 2001).

No contexto da temática do documento, a análise de oportunidade ocorreu no mercado de software hospitalar, quando se procurou estudar valências negativas e falhas que poderiam ser supridas com um produto novo.

Seguem-se alguns pontos pertinentes que foram abordados neste contexto:

- A nova solução/ abordagem seria algo que interessasse realmente aos hospitais, ou os clientes estariam mais interessados apenas em melhorar os produtos existentes?
- Seria uma solução específica para um pequeno setor do mercado hospitalar ou seria transversal ao mesmo?
- A migração de dados (críticos) relativa, entre outros, a doentes, coloca entraves e traz um risco acrescido à oportunidade?
- A centralização de diferentes fontes acrescenta valor ao cliente?

A análise e avaliação destas questões, levada a cabo através dos métodos supracitados, foi positiva e levou à conclusão de que a oportunidade era viável. Desta forma, o processo avançou para as próximas fases.

A.1.2 Value, value for the costumer, perceived value

A.1.2.1 Value

A criação de valor é um conceito difícil de explanar e compreender. Alguns autores consideram que este é alcançado através de um compromisso entre benefícios e sacrifícios do ponto de vista dos clientes na oferta do fornecedor (Walters and Lancaster, 2000)

Benefícios:

- A centralização de pedidos e requisições médicas do software simplifica o processo do staff médico.
- A Interoperabilidade com consumo de serviços externos permite a centralização.
- O uso de micro aplicações permite uma maior escalabilidade e flexibilidade.

Sacrifícios:

- Implementação possivelmente mais custosa em relação ao tempo despendido *versus* uma implementação com arquitetura monolítica.
- A interoperabilidade é uma operação complexa.

A.1.2.2 Valor para o cliente

"Valor para o cliente" pode ser usado para representar tanto o que o cliente percebe e recebe como aquilo que pode oferecer (Woodall, 2003).

Na temática do documento, o valor para o cliente é a mais-valia que o staff médico obtém em ter uma solução centralizada que permite gerir todo o processo de requisição e pedidos das diferentes áreas médicas.

A.1.2.3 Valor percecionado

O valor percecionado é um termo usado para ilustrar a diferença de perceção que os clientes têm para o mesmo produto/ serviço. Da mesma forma, as organizações podem ter também diferentes perceções do valor que os clientes atribuem ao produto/ serviço (Ulaga and Eggert, 2006) .

Nadim Habib, professor da Nova School of Business and Economics, defende que a qualidade é relativa às expetativas. O cliente verá o produto/ serviço com mais ou menos qualidade consoante este ultrapasse ou não as expetativas que tem em relação ao mesmo[82]. Esta é uma visão que coloca o cliente numa posição parcial onde a qualidade é alvo da sua perceção.

O autor deste documento propõe um paralelismo entre estes dois conceitos, onde o valor percecionado é a qualidade, relativa às expetativas, que o cliente atribui de forma singular a um produto/ serviço que dê resposta a um problema.

Dado que o staff médico hospitalar está consciente de que a informação descentralizada custa tempo que poderia ser dedicado aos doentes, este produto acrescentará sempre valor ao cliente dado que a centralização permitirá reduzir esse tempo. No entanto, dependendo do cliente e das suas expetativas, este valor estará, tal como o nome indica, sujeito à sua perceção.

A.1.3 Value Proposition

A.1.3.1 Modelo de Negócio de Canvas

O Value proposition é uma vista geral do conjunto de produtos e serviços de uma organização que ajudam a acrescentar valor ao cliente.

Em específico o modelo Canvas de value proposition subdivide-se em dois componentes – ver Figura 87 – o **Costumer Profile**, onde é clarificada a perceção dos clientes, e o **Value Map**, onde é descrita a forma como se pretende criar valor para o cliente. Quando estas duas componentes se encontram, é alcançado uma posição de consenso mútuo conhecida pelo nome **Fit** (Osterwalder *et al.*, 2014).

O **Value Map** ou Value proposition Map (ver Figura 87 ,bloco da esquerda) é representado por um quadrado, que se divide em três partes:

[82] Retirado em fevereiro de 2022 de vídeo - https://jornaleconomico.sapo.pt/noticias/o-que-e-afinal-qualidade-114902

- **Products and Services** – lista todos os produtos e serviços que estão envolvidos na value proposition previamente contruída.
- **Gain Creators** – descreve como os produtos e serviços representam ganhos para o cliente.
- **Pain Killers / Relievers** – descreve como os produtos e serviços aliviam problemas do cliente.

Por outro lado, o **Costumer Profile** (ver Figura 87, bloco da direita) é representado por um círculo, também este dividido em três partes:

- **Gains** – descreve os *outcomes* e benefícios que o cliente procura.
- **Pains** – descreve os riscos, obstáculos e possíveis *outcomes* indesejados relacionados com o trabalho do cliente.
- **Jobs ou Costumer Jobs** – descreve o que o cliente está a tentar alcançar no seu trabalho.

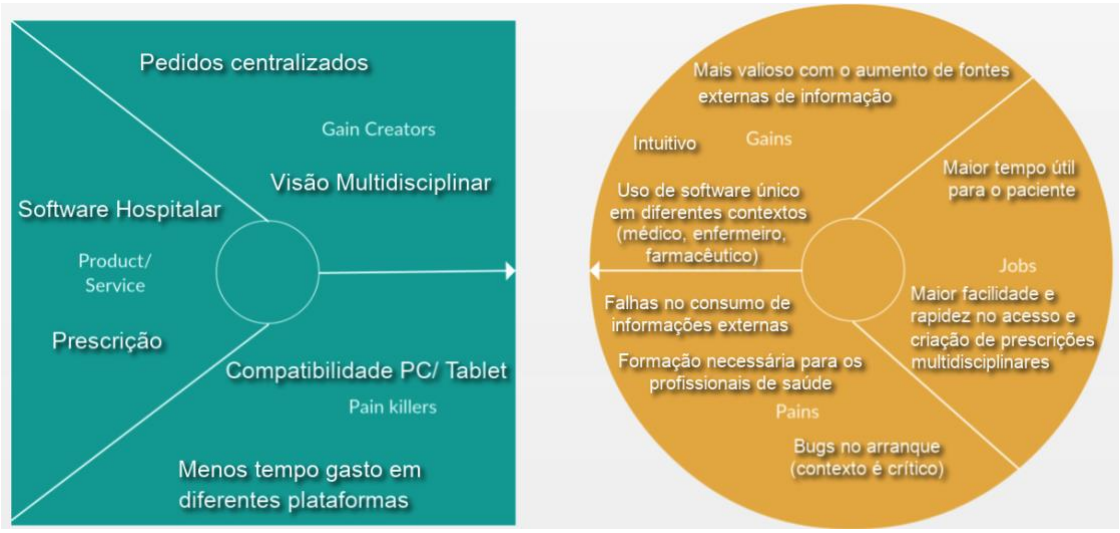

Figura 87 - Value proposition CANVAS aplicado no contexto do documento

A.1.3.2 FAST Diagram

O Function Analysis System Technique (FAST) diagram é uma representação gráfica daquilo que deve ser executado pelo produto ou serviço de forma a alcançar o seu propósito, permitindo também analisar custos.

Esta representação é feita na forma de um diagrama com uma relação How? / Why? que procura de uma forma sistemática e lógica estabelecer relações de funções de um projeto, produto, processo ou serviço[83].

As maiores valências deste modelo são a utilidade para verificar e ilustrar como uma solução proposta alcança aquilo a que se propõe e a utilidade para identificar funções em falta, desnecessárias ou até duplicadas.

No contexto do documento e como se pode analisar na Figura 88, de forma a responder à pergunta "How?" a leitura deve ser feita da esquerda para a direita:

- O objetivo principal é desenvolver um CPOE para um software hospitalar. Como? (How?) Satisfazendo os critérios/ requisitos do cliente. Como? Através de validação de requisitos.
- Como? Realizando diversos testes.
- Como se faz os testes? Através do desenvolvimento das micro-aplicações que é feito com o recurso a mecanismos de Interoperabilidade com FHIR que poderão assim ser testados.
- Como se desenvolve? Através do desenho da solução.
- Como se chega ao desenho da solução? Através do levantamento de requisitos.

O mesmo exercício pode ser feito para a leitura do diagrama da direita para a esquerda, respondendo à pergunta "Why?" (Porquê)?

1. Vai ser feito o levantamento de requisitos. Porquê?
2. Para poder ser feito o desenho da solução. Porquê?
3. Para ser feito o desenvolvimento das micro-aplicações, com o recurso a mecanismos de Interoperabilidade com FHIR. Porquê?
4. Para validar os requisitos? Porquê?
5. Para satisfazer os critérios/ requerimentos do cliente.

[83] Retirado em fevereiro de 2022 de https://extrudesign.com/function-analysis-and-system-technique-fast-diagram/

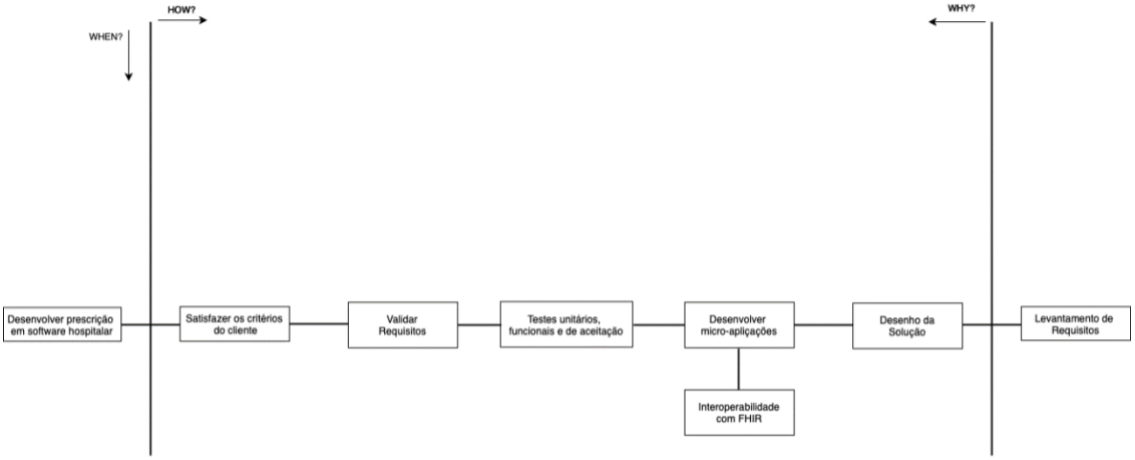

Figura 88 - Diagrama FAST aplicado ao contexto do documento

A.1.3.3 AHP

O Analytic Hierarchy process (AHP) foi desenvolvido em 1980 para resolver problemas de decisão complexos. O AHP usa critérios de avaliação baseados em comparações e permite dividir o problema em níveis hierárquicos (Saaty, 1980, 2008).

Este método tem especial interesse para processos de descrição em que aspetos quantitativos e qualitativos devem ser considerados.

Através de critérios de comparação considerando alternativas, este modelo reduz a complexidade da decisão, sistematizando-a (Jakupovic, Pavlic and Candrlic, 2010).

O modelo AHP consiste nas seguintes operações:

1. Construção da hierarquia do processo de decisão.
2. Comparação entre elementos da hierarquia.
3. Prioridade relativa de cada critério.
4. Avaliação da consistência das prioridades relativas.
5. Construção da matriz de comparação paritária para cada critério tendo e conta as alternativas.
6. Obter a prioridade composta para as alternativas.
7. Escolha da alternativa.

Na **construção da hierarquia do processo de decisão** ou **Divisão Hierárquica** a prioridade é definir um objetivo. Neste caso será o de escolher uma tecnologia para a camada de *front-end* do desenvolvimento do CPOE.

De seguida devem ser definidos os critérios que serão tidos em conta para a escolha final: Desempenho; Suporte; Complexidade de Uso; Escalabilidade.

Finalmente devem ser definidas as alternativas: *React*; *Angular*; *Vue*.

Na **Comparação entre elementos da hierarquia** são estabelecidas prioridades entre os elementos da hierarquia, através de uma matriz de comparação. Nesta fase é importante usar uma escala de valores para que as comparações sejam coerentes e consistentes. Para este efeito será utilizada a Escala Fundamental de Saaty.

Desta comparação resultou a Tabela 12.

Tabela 12 - Matriz de comparação dos critérios

	Desempenho	Suporte	Complexidade	Escalabilidade
Desempenho	1	3	3	1/4
Suporte	1/3	1	1/2	1/8
Complexidade	1/3	2	1	1/6
Escalabilidade	4	8	6	1

Na terceira fase, a **Prioridade relativa de cada critério**, os valores da matriz de comparação são normalizados, ou seja, os critérios são igualados a uma mesma unidade, e é calculado o vetor de prioridades, que determina a ordem de importância de cada critério. Cada critério terá assim o seu vetor de prioridade a Prioridade relativa e esta é tanto maior quanto maior o valor absoluto do vetor próprio.

Tabela 13 - Matriz normalizada com prioridade relativa

	Desempenho	Suporte	Complexidade	Escalabilidade	Prioridade Relativa
Desempenho	3/17	3/14	2/7	6/37	0,2097
Suporte	1/17	1/14	1/21	3/37	0,0647
Complexidade	1/17	1/7	2/21	4/37	0,1013
Escalabilidade	12/17	4/7	4/7	24/37	0,6243

A matriz normalizada com a prioridade relativa de cada critério está representada na Tabela 13.

Através da análise desta tabela pode concluir-se a partir dos resultados relativos que o critério de Escalabilidade aparece em primeiro lugar, tal como seria de esperar, dado que os softwares hospitalares são críticos e devem permitir e comportar uma alta utilização do sistema. O desempenho surge em segundo lugar, seguido da Complexidade e finalmente do Suporte. Estão assim definidos os pesos de cada critério.

A fase seguinte é a **Avaliação da consistência das prioridades relativas** e consiste no cálculo da razão de consistência (RC) que permite compreender se o decisor dos passos anteriores foi racional. A lógica matemática em que se A depende de B e B depende de C, então A depende de C, é aplicada nesta avaliação.

De forma a fazer esta avaliação é necessário calcular a Razão de consistência através do vetor próprio. Assim, deve ser feita a média da divisão entre - a multiplicação do valor de cada vetor próprio pelo número de critérios – e o valor do vetor próprio:

$$\lambda\max = \text{average}\left\{\frac{0{,}84}{0{,}21}, \frac{0{,}24}{0.06}, \frac{0{,}40}{0{,}10}, \frac{2{,}48}{0{,}62}\right\} = 4.00$$

De seguida deve ser calculado o İndice de Consistência (IC):

$$IC = (\lambda\max - \mathrm{n})/(\mathrm{n} - 1) = (4.00 - 4)/(4 - 1) = 0.00$$

Analisando a Figura 89,Figura 89 - Valores de IR para matrizes quadradas de ordem n tabela que representa os valores a serem usados para o cálculo da Razão de Consistência (RC) para cada número de critérios, podemos que verificar no caso estudado o valor que deve ser usado para o número de critérios (4) é 0.90.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

Figura 89 - Valores de IR para matrizes quadradas de ordem n

Assim, fazendo o cálculo da RC:

$$RC = \frac{IC}{RC} = \frac{0{,}00}{0.90} = 0{,}0 < 0{,}1$$

Dado que RC é inferior a 0.1, conclui-se que os valores das prioridades relativas utilizadas são consistentes.

A fase seguinte, **Construção da matriz de comparação paritária para cada critério tendo e conta as alternativas**, é feito um processo em todo semelhante ao que foi feito anteriormente. É construída a matriz de comparação paritária e a matriz normalizada com a prioridade relativa, mas desta vez este processo é feito para cada um dos critérios.

Desempenho

Tabela 14 - Matriz de comparação paritária de Desempenho

	React	Angular	Vue
React	1	1	1/2
Angular	1	1	1/2
Vue	2	2	1

Tabela 15 - Matriz normalizada com prioridade relativa de Desempenho

	React	Angular	Vue	Prioridade Relativa
React	1/4	1/4	1/4	0,2500
Angular	1/4	1/4	1/4	0,2500
Vue	1/2	1/2	1/2	0,5000

Suporte

Tabela 16 - Matriz de comparação paritária de Suporte

	React	Angular	Vue
React	1	1	4
Angular	1	1	4
Vue	1/4	1/4	1

Tabela 17 - Matriz normalizada com prioridade relativa de Suporte

	React	Angular	Vue	Prioridade Relativa
React	4/9	4/9	4/9	0,4400
Angular	4/9	4/9	4/9	0,4400
Vue	1/9	1/9	1/9	0,1111

Complexidade

Tabela 18 - Matriz de comparação paritária de Complexidade

	React	Angular	Vue
React	1	3	1/2
Angular	1/3	1	1/5
Vue	2	5	1

Tabela 19 - Matriz normalizada com prioridade relativa de Complexidade

	React	Angular	Vue	Prioridade Relativa

React	3/10	1/3	5/17	0,3092
Angular	1/10	1/9	10/85	0,1096
Vue	3/5	5/9	10/17	0,5813

Escalabilidade

Tabela 20 - Matriz de comparação paritária de Escalabilidade

	React	Angular	Vue
React	1	2	7
Angular	1/2	1	6
Vue	1/7	1/6	1

Tabela 21 - Matriz normalizada com prioridade relativa de Escalabilidade

	React	Angular	Vue	Prioridade Relativa
React	14/23	12/19	1/2	0,5800
Angular	7/23	6/19	3/7	0,3496
Vue	2/23	1/19	1/14	0,070

Obter a prioridade composta para as alternativas é a fase seguinte e consiste em multiplicar a matriz das prioridades com o peso de cada critério, cujos resultados levam à última fase, a **Escolha da alternativa**:

$$\begin{bmatrix} 0{,}2500 & 0{,}4400 & 0{,}3092 & 0{,}5800 \\ 0{,}2500 & 0{,}4400 & 0{,}1096 & 0{,}3496 \\ 0{,}7500 & 0{,}1100 & 0{,}5813 & 0{,}070 \end{bmatrix} \times \begin{bmatrix} 0{,}2097 \\ 0{,}0647 \\ 0{,}1013 \\ 0{,}6243 \end{bmatrix} = \begin{matrix} 0{,}4742 \\ 0{,}3103 \\ 0{,}2670 \end{matrix}$$

Assim, foi calculada a prioridade composta para cada alternativa:

1. React – 0,4742
2. Angular – 0,3103
3. Vue – 0,2670

Analisando estes dados, chega-se à conclusão de que a alternativa *React* é a mais vantajosa em função dos critérios e pesos definidos.

Apêndice B

B.1 Dimensão Suportabilidade – Escala de Avaliação

Navegação	Wfk- Fullfilment %				
Requisito	0	25	50	75	100
UN1 - A navegação entre funcionalidades/ widgets deve ser simples e intuitiva	Navegação morosa e desconexa	Navegação inconsistente nos tempos de resposta e desconexa	Navegação simples, mas desconexa	Navegação poderia ser mais intuitiva	Navegação intuitiva, simples e condizente com os fluxos de negócio idealizados
UN2 - Utilizador recebe *feedback* após ações	Não existe qualquer tipo de feedback	-	Utilizador é notificado em algumas ações, quando estas são efetuadas com sucesso	-	O utilizador é notificado do resultado das suas ações, em caso de sucesso ou falha
UN3 - Tempos de carregamento dos widgets e processamentos de ações devem ser rápidos (< 3 segundos)	Tempos superiores a 10 segundos	-	Tempos geralmente próximos dos 5 segundos	-	Tempos consistentemente de 3 segundos, ou menores

Figura 90 - Escala de Avaliação do Fator Navegação

Linguagem	Wfk- Fullfilment %				
Requisito	0	25	50	75	100
UL1 - Está disponível nos idiomas Português e Inglês	-	-	Apenas suporta um idiomas	-	Suporta ambos os idiomas
UL2 - Terminologia é adequada e agnóstica a FHIR	Utilização incorreta ou pouco precisa	-	A terminologia é correta, mas não a mais adequada para o profissional de saúde	-	Terminologia completamente adequada

Figura 91 - Escala de Avaliação do Fator Linguagem

B.2 Dimensão Usabilidade – Escala de Avaliação

Manutibilidade	Wfk- Fullfilment %				
Requisito	0	25	50	75	100
SM1 - A implementação de novas funcionalidades não terá impacto negativo nas já existentes	Serão necessárias alterações a nível de desenho e implementação para as funcionalidades existentes	-	Pontualmente serão neccessários ajustes ao nível da implementação	-	Não serão necessárias alterações para a adição de funcionalidades
SM2 - A arquitetura e detalhes técnicos de implementação estão documentados	Não há documentação	Existe documentação incompleta relativa a algumas decisões e processos	Existe documentação completa relativa às decisões e processos mais relevantes	Existe documentação transversal a todo o projeto, com pequenas falhas	Documentação completa e detalhada
SM3 - A implementação não afetou a aplicação Viewer	Foram necessárias alterações em grande escala	-	Foram necessárias algumas alterações	-	Não foram necessárias alterações

Figura 92 - Escala de Avaliação do Fator Manutibilidade

Adaptabilidade	Wfk- Fullfilment %				
Requisito	0	25	50	75	100
SA1 - A aplicação está disponível em diferentes browsers	Acesso apenas num browser	Acesso em 2 browsers	Acesso em 3 browsers	Acesso em 4 browsers	Acesso em 5 ou mais browsers
SA2 - A implementação e arquitetura não causam dificuldades na adição de novas funcionalidades	É necessário um grande conhecimento a nivel técnico da implementação atual	-	É necessário algum conhecimento prévio	-	É necessário pouco ou nenhum conhecimento prévio para desenvolver novas funcionalidades

Figura 93 - Escala de Avaliação do Fator Adaptabilidade

B.3 Dimensão Funcionalidade – Escala de Avaliação

Notificações	Wfk- Fullfilment %				
Requisito	0	25	50	75	100
FN1 - O utilizador é notificado quando associa dispositivos ao paciente, em caso de sucesso ou insucesso	Inexistente	-	Com falhas de implementação	-	Completamente implementado
FN2 - O utilizador é notificado quando elimina uma intervenção de um dispositivo, em caso de sucesso ou insucesso	Inexistente	-	Com falhas de implementação	-	Completamente implementado
FN3 - O utilizador é notificado quando edita uma intervenção de um dispositivo, em caso de sucesso ou insucesso	Inexistente	-	Com falhas de implementação	-	Completamente implementado
FN4 - O utilizador é notificado quando adiciona uma intervenção de um dispositivo, em caso de sucesso ou insucesso	Inexistente	-	Com falhas de implementação	-	Completamente implementado

Figura 94 - Escala de Avaliação do Fator Notificações

Integridade e Tratamento de Dados	Wfk- Fullfilment %				
Requisito	0	25	50	75	100
FITD1 - O sistema garante a autenticidade e integridade de todos os dados.	Inexistente	-	Com falhas de implementação	-	Completamente implementado
FITD2 - O sistema garante a segurança e privacidade de todos os dados dos pacientes.	Inexistente	-	Com falhas de implementação	-	Completamente implementado

Figura 95 - Escala de Avaliação do Fator Integridade e Tratamento de dados

Profissional de Saúde	Wfk- Fullfilment %				
Requisito	0	25	50	75	100
FPS1 - O sistema permite a listagem de dispositivos livres para associação.	Inexistente	-	Com falhas de implementação	-	Completamente implementado
FPS2 - O sistema permite a associação de dispositivos ao paciente.	Inexistente	-	Com falhas de implementação	-	Completamente implementado
FPS3 - O sistema permite a listagem dos dispositivos associados ao paciente.	Inexistente	-	Com falhas de implementação	-	Completamente implementado
FPS4 - O sistema permite a adição e calendarização de intervenções para dispositivos	Inexistente	-	Com falhas de implementação	-	Completamente implementado
FPS5 - O sistema permite aconsulta dos resultados obtidos nas monitorizações dos sinais vitais do paciente	Inexistente	-	Com falhas de implementação	-	Completamente implementado

Figura 96 - Escala de Avaliação do Fator Profissional de Saúde