INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Plataforma de Integração na Área da Saúde

JORGE MIGUEL SILVA PEREIRA
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Plataforma de Integração na Área da Saúde

Deloitte Touche Tohmatsu Services, Inc.

1150528 – Jorge Miguel Silva Pereira

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

Porto, outubro 2020

Deloitte.

Plataforma de Integração na Área da Saúde

Deloitte Touche Tohmatsu Services, Inc.

1150528 – Jorge Miguel Silva Pereira

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

Orientador ISEP: **Paulo Maio**

Supervisores Externos: **Miguel Gaspar e Eduardo Oliveira**

Porto, outubro 2020

Agradecimentos

Agradeço a todos os que possibilitaram a realização deste projeto. Começando pelo Eng. Miguel Gaspar que me acolheu na área de integração na Deloitte, e ao Eduardo Oliveira por tentar apoiar-me sempre que possível naquilo que conseguia.

Agradeço ao professor Paulo Maio por toda a sua disponibilidade e ajuda ao longo deste projeto, dando animo e não desistindo mesmo quando tudo parecia encaminhar-se para esse lado, a ele o meu muito obrigado.

Agradeço à Joana Cruz por todo o apoio dado ao longo desta fase difícil, em que o tempo não era muito e mesmo assim conseguiu sempre ajudar-me direta e indiretamente a concluir esta dissertação.

Por último, mas não menos importante, agradeço aos meus pais por todos os esforços feitos para que eu tivesse aquilo que eles não tiveram hipótese de ter, a eles estarei eternamente agradecido.

A todos, o meu sincero agradecimento.

Resumo

Esta dissertação foi elaborada em contexto empresarial, a empresa em questão é a Deloitte Portugal, que se foca em prestar serviços de consultadoria a terceiros e não só. Foi então aí que surgiu a oportunidade de desenvolver este projeto.

A globalização e o desenvolvimento tecnológico alteraram a realidade de diversas organizações. Atualmente há um aumento da necessidade de ter informação disponível online e atualizada, pois, esta garante que as decisões sejam tomadas com mais assertividade. Com o surgimento de novas soluções de *Information Technology* (TI) para atender as demandas do mercado em geral, ter uma integração de sistemas tornou-se essencial para otimizar processos, centralizar dados e melhorar a experiência dos utilizadores.

O problema descrito nesta dissertação passa pela a ausência de uma camada intermédia capaz de efetuar comunicações entre sistemas heterogéneos e fazer gestão dessas mesmas comunicações.

O objetivo desta dissertação é o estudo e implementação de uma *framework* de integração capaz de responder aos diversos pedidos que o software de gestão hospitalar, já desenvolvido, ePatient consiga comunicar com os diversos serviços que um hospital tenha implementado.

Parte do desenvolvimento desta *framework* passa por tratar erros, redirecionar pedidos e monitorizar *Application Programming Interfaces* (APIs), para isso é usado um software de integração.

Palavras-chave: Integração, ePatient, Hospital, Gestão

Abstract

This dissertation was developed in a business context, the company in question is Deloitte Portugal, which focuses on providing consultancy services to third parties and beyond. It was then that the opportunity to develop this project arose.

Globalization and technological development have changed the reality of several organizations. Currently, there is an increasing need to have information available online and updated, as this ensures that decisions are made with more assertiveness. With the emergence of new Information Technology (IT) solutions to meet the demands of the market in general, having a system integration has become essential to optimize processes, centralize data and improve the user experience.

The problem described in this dissertation is the absence of an intermediate layer capable of making communications between heterogeneous systems and managing those communications.

The objective of this dissertation is the study and implementation of an integration framework capable of responding to the different requests that the hospital management software, already developed, ePatient is able to communicate with the various services that a hospital has implemented.

Part of the development of this framework involves handling errors, redirecting requests and monitoring Application Programming Interfaces (APIs), for which an integration software is used.

Keywords: Integration, ePatient, Hospital, Management

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

IT	*Information Technology*
API	*Application Programming Interface*
NCD	*New Concept Development*
FEI	*Front End of Innovation*
SLA	*Service Level Agreements*
RALM	*RESTful API Modeling Language*
BPM	*Business Process Management*
EAI	*Enterprise Application Integration*
ESB	*Enterprise Service Bus*
AD	*Active Directory*
MOM	*Message Oriented Middleware*
SO	Sistema Operativo
HL7	*Health Level 7*
HTTP	*Hypertext Transfer Protocol*
JWT	JSON *Web Token*
EIP	*Enterprise Integration Pattern*
JSON	*JavaScript Object Notation*

Capítulo 1

Introdução

Neste primeiro capítulo é apresentado o enquadramento da dissertação, o problema subjacente a este trabalho e consequentes objetivos que se pretendem atingir. Por fim, apresenta-se a estrutura geral do documento.

1.1 Enquadramento

Atualmente é cada vez mais importante informatizar tudo aquilo que é possível. Toda esta informatização surgiu com a necessidade de poupar tempo/recursos, gerando uma maior facilidade e comodidade no que toca ao acesso à informação, facilitando assim a respetiva gestão, havendo consecutivamente a necessidade da existência de diversos sistemas. A informatização dos processos das organizações conduziu a uma proliferação de sistemas/aplicações dentro da mesma organização, cada uma dedicada a um subconjunto específico de tarefas/processos. Essa mesma proliferação levou ao aparecimento de novos problemas/questões como por exemplo, redundância de tarefas como a introdução da mesma informação em sistemas diferentes [1].

De forma a resolver/minimizar as questões mencionadas no paragrafo anterior, torna-se necessário que os vários sistemas interajam entre si, no sentido de partilharem dados/informação, sendo a interoperabilidade entre sistemas algo fundamental atualmente [1].

A necessidade de uma interoperabilidade aplica-se a diversas áreas de negócio, como a indústria, tecnologias, ente outros. Como tal isto também se aplica na área da saúde, sendo esta o âmbito do presente trabalho.

Neste contexto, a Deloitte desenvolveu um software de apoio aos hospitais, denominado *ePatient*. Este é um sistema de gestão hospitalar focado na monitorização de pacientes que dispõe de diversas funcionalidades e, em particular, de alarmística, como falta de medicamentos, tensão arterial, orientadas essencialmente ao apoio aos profissionais de saúde (e.g. médicos, enfermeiros, auxiliares).

Numa perspetiva prática, este sistema é composto por duas aplicações distintas. A primeira aplicação é para uso/acesso exclusivo pelos diferentes profissionais de saúde com vista à monitorização dos pacientes internados como, por exemplo, a cama onde estão, o seu historial clínico, medicamentos que estão a tomar, entre outros dados. A segunda é uma aplicação móvel direcionada aos pacientes internados no hospital. Através da mesma, o paciente pode dar *feedback* do seu estado atual como, por exemplo, se se encontra com dores ou com algum desconforto, como também tem a possibilidade de pedir algo que esteja em falta, como comida, bebida, entre outras coisas.

Para tal, este sistema reúne e/ou manipula dados clínicos de diversas áreas/especialidades como, por exemplo, tomas de medicamentos, medição de temperatura, refeições, e dados não clínicos, como quantidade de medicamentos armazenados. De forma a evitar/minimizar a introdução repetida de informação, e consequentes problemas daí advindos, é necessário que o *ePatient* interaja com múltiplos sistemas/serviços já em uso nos hospitais, tais como, sistemas dedicados à gestão de utentes e de profissionais de saúde, de recursos físicos e materiais, do historial clínico, entre outros.

1.2 Problema

Como referenciado no subcapítulo anterior, o sistema desenvolvido *ePatient* tem a necessidade de comunicar com outros serviços, serviços esses que comunicam de forma diferente e contêm uma estrutura de dados diferentes uns dos outros. Com isto, e apesar das diferentes maneiras de comunicação, o *ePatient* tem de ser capaz de comunicar com estes sem que haja qualquer problema derivado dos sistemas serem heterogéneos.

Como referenciado também no subcapítulo anterior, existem diversos problemas que poderão surgir com o aumento de sistemas que necessitam de comunicar entre si, como é o caso de cada sistema/serviço disponibiliza/permite diferentes formas de interoperabilidade. Por exemplo, para alguns a interoperabilidade apenas é possível por acesso direto à base de dados,

contudo noutros casos a interoperabilidade é conseguida através da invocação de serviços web (*webservices*) e outros ainda por troca de ficheiros através de um diretório partilhado.

Depois de analisadas as diversas maneiras de como os diferentes sistemas disponibilizam as suas formas de comunicação, existe, ainda dentro de cada interoperabilidade, diferentes protocolos usados, por exemplo, nos serviços web uns sistemas podem usar REST [2] e outros o protocolo SOAP [2].

Por fim verifica-se também a heterogeneidade ao nível das estruturas de dados utilizadas como, por exemplos, diferentes nomenclaturas ou até mesmo de estrutura.

Neste momento o *ePatient* lida com esta variabilidade (de forma, de protocolo, de estrutura e de semântica) através de uma ligação/integração direta com cada um dos sistemas em causa. Isto obriga a que para cada sistema seja desenvolvido/codificado um método de interação especifica. Esta situação é agravada pelo facto dos sistemas com o qual o *ePatient* necessita de integrar, variar de hospital para hospital.

A falta de uma camada de integração, capaz de fazer a ligação entre os vários serviços de modo simples e organizado, faz com que ocorram vários problemas como por exemplo:

- **Custos elevados / dificuldade de manutenção das aplicações e respetiva evolução**
  As eventuais modificações ao *ePatient* ao longo do tempo, para ajustamentos e melhorias não desenhados de forma integrada, tornam a respetiva manutenção do sistema de integração (SI) quase impraticável [3];

- **Proliferação de diferentes aplicações que repitam procedimentos comuns**
  Devido à necessidade de introdução de dados de terceiros (como por exemplo, pela necessidade de informação de várias áreas dos hospitais), as mesmas entidades eram criadas isoladamente pelas diferentes áreas, por diferentes aplicações, causando redundância da mesma entidade na base de dados [3];

- **Rigidez das aplicações**
  Falta de flexibilidade da aplicação *ePatient* devido aos pontos anteriormente esclarecidos [3].

Com isto, é então necessário procurar uma solução alternativa, capaz de resolver/minimizar estes problemas e mais fácil para o momento da venda.

1.3 Objetivos

Após expor o problema, é clara a existência de quatro objetivos essenciais para o sucesso no desenvolvimento da presente dissertação:

1. Estudar conceitos, abordagens e tecnologias relacionadas com a interoperabilidade de sistemas e a sua aplicabilidade na área da saúde;
2. Reformular o processo/método de integração do *ePatient* com os sistemas hospitalares de forma a reduzir/minimizar os problemas já identificados e reportados no subcapítulo anterior;
3. Contemplar o processo de integração com novos requisitos de negócio relacionados com a necessidade, cada vez maior, de garantir a segurança da informação e o seu acesso, bem como facilitar e promover a adoção de mecanismos de auditoria e rastreio;
4. O processo de integração resultante deve ter um desempenho igual ou melhor do que o processo de integração atual.

No fundo eliminar os problemas referidos no subcapítulo 1.2, pois existe uma enorme rigidez na implementação atual da solução.

1.4 Estrutura da Dissertação

Este documento apresenta-se divido por capítulos, sendo esses: Introdução, Contexto, Integração de Aplicações, Análise das tecnologias de Integração, Desenho da solução, Implementação e Testes.

No presente capítulo, Introdução, são descritos o enquadramento, o problema e os objetivos a alcançar. Esta também apresentada toda a estrutura da dissertação.

No capítulo 2, identificado como Contexto, descreve o estudo e análise de soluções existentes no mercado. Depois das várias soluções é apresentado mais ao detalhe a aplicação *ePatient* e dado um pouco de contexto sobre a mesma. É também apresentada uma descrição do problema e a área de negócio onde este projeto incide. Por fim, é realizado um pequeno inquérito, com vista à comparação de diferentes formas de integração.

De seguida no capítulo 3, Integração de Aplicações, existe uma contextualização sobre o que é integração, os vários tipos de integração, as suas vantagens e desvantagens e todos os conceitos importantes para uma compreensão mais técnica. Neste capítulo são também apresentadas várias tecnologias, existentes no mercado, capazes de contruir a *framework* necessária para a resolução do problema apresentado.

No capítulo 4, Análise das Tecnologias de Integração, é apresentada uma análise mais aprofundada a cada uma das tecnologias consideradas mais indicadas para o desenvolvimento deste projeto. É realizada uma avaliação comparativa às várias e apresentada a decisão tomada sobre qual escolher.

No capítulo 5, Desenho da Solução, é apresentado todo o desenho da solução final e alguns padrões de desenvolvimento de software.

Nos capítulos 6 e 7, Implementação e Experiências e Avaliação, é apresentado a implementação e os testes realizados à mesma. O capítulo 6 é dedicado à apresentação de detalhes relacionados com o enquadramento e implementação das soluções. O capítulo 7 é dedicado aos testes realizados à aplicação para comprovar o valor da solução desenvolvida.

E por fim, no capítulo 8, Conclusões, é realizada uma retrospetiva sobre os objetivos traçados e sobre as funcionalidades implementadas.

Capítulo 2

Contexto

No capítulo anterior foi descrito, de forma simples e sucinta, o problema em mãos. Neste capítulo, procede-se à análise mais aprofundada do contexto, consistindo na análise de soluções existentes no mercado e principais concorrentes do *ePatient* e da área, uma análise sistemática do *ePatient* e da área de negócio em que este se enquadra.

2.1 Soluções Existentes

Neste subcapítulo são então abordadas algumas das soluções existentes no mercado. As soluções aqui referenciadas são soluções que pertencem à mesma área de negócio do sistema desenvolvido (*ePatient*), tendo sido todas desenvolvidos em Portugal. Estas aplicações contêm um número elevado de funcionalidades comuns a todos os sistemas, existindo pequenas diferenças entre eles.

2.1.1 Global Care

O *Global Care* [4] é uma solução capaz de integrar vários módulos, tanto de gestão de hospitais como de clínicas. Esta solução foi desenvolvida pela empresa Glintt, sendo talvez o concorrente mais direto do *ePatient*.

Dado o seu desenvolvimento em módulos é capaz de seguir um paciente desde o momento que chega às urgências, ao passar pelo ambulatório e até ao momento da sua alta. É capaz também de registos simples como consultas, prescrições, entre outas.

GlobalCare é um software com diversos e variados módulos sendo que contém dois que se identificam com o software *ePatient*. Estes são *Hospital Management System* e *Eletronic health Record* [4].

2.1.1.1 Hospital Management System

Este módulo tem como objetivo dar suporte administrativo à identificação e gestão do paciente em unidades hospitalares e clínicas. Tem a possibilidade de registar e monitorizar todo o processo e fluxo de pacientes, desde o primeiro contacto com o hospital até à sua administração no mesmo. Para além da gestão de pessoas, faz também o acompanhamento de pacientes em ambulatório, gestão de consultas entre outros [5].

2.1.1.2 Electronic health record

Neste módulo o software simplifica a interação entre o médico e o paciente, permitindo assim aceder e registar todas as informações pertinentes do foro clínico, possibilitando ao paciente ser tratado de uma forma mais personalizada e centrado em si. O profissional de saúde tem ao seu dispor o acompanhamento da evolução clínica, os diagnósticos, alertas e opção de marcação de consultas [6].

Tem uma *framework* de interoperabilidade robusta, utilizando e disponibilizando um conjunto de conectores "*out-of-the-box*" suportados pela tecnologia *Mulesoft*. Aposta fortemente na segurança e auditoria, permitindo assim ao cliente ter uma maior autonomia no controlo dos seus processos de integração. Permite também a monitorização em tempo real dos serviços a que está conectado, consequentemente, existe uma maior rapidez na identificação, podendo posteriormente relatar os problemas nas respetivas interfaces [6].

2.1.2 F3M

Este software assegura um controlo global dos recursos disponíveis, otimizando a sua utilização tendo em conta as necessidades dos diferentes serviços. Engloba diversas áreas como a gestão financeira e administrativa, os serviços de atendimento em ambulatório, a gestão da capacidade de internamento, o planeamento cirúrgico e a logística hospitalar.

Em conclusão este software é mais focado na área de gestão hospitalar e não tanto no acompanhamento do doente desde o momento em que este dá entrada no hospital [7].

2.1.3 Software de Gestão – MANTHOSP

É um software para a gestão e otimização das atividades de manutenção do sector da saúde. Este facilita a interligação com outros sistemas no ambiente hospitalar para a consolidação de informações e tomada de decisões.

Tem como principais objetivos a otimização e manutenção dos hospitais, minimizar os tempos de respostas, garantindo o cumprimento dos *Service Level Agreements* (SLAs) e por fim facilitar a integração dos processos de manutenção com outras áreas. Esta interligação (integração) com outros sistemas é feita através de mensagens *Health Level 7* (HL7) [8].

2.1.4 ALERT® PAPER FREE HOSPITAL (PFH)

É um sistema pertencente à empresa *Alert*, e trata-se de uma solução que tem como objetivo informatizar por completo o hospital, fazendo com que seja possível que a documentação, a integração e a revisão de toda a informação relacionada com operações hospitalares, com registo de informação clinica em tempo real, apresentando assim as seguintes vantagens [9]:

- Melhorar a satisfação dos pacientes e profissionais;
- Aumentar a eficiência das operações hospitalares;
- Fornecer ao administrador hospitalar informações fiáveis para uma melhoria da gestão;
- Controlar os custos;
- Aumentar a performance dos profissionais;
- Aumentar as perspetivas de melhoria dos pacientes e reduzir os erros médicos.

2.2 SePatient

O *ePatient* é um sistema de software dedicado à gestão hospitalar composto por múltiplos componentes que permitem uma gestão eficiente dos pacientes hospitalizados, como também a monitorização de mais de 300 indicadores clínicos, médicos, sociais e administrativos de cada paciente. Este tem como objetivo principal facilitar o dia-a-dia dos profissionais de saúde e dos pacientes baixando a complexidade que existe no registo de toda a atividade referente ao episódio que conta a "história" do paciente desde que deu entrada no hospital. Mantendo registo dos pacientes numa só aplicação e melhorando a comunicação entre os profissionais de saúde e os pacientes.

Na Figura 1 é possível ver uma representação geral (de contexto) de como o *ePatient* neste momento interage, tanto com os atores, profissionais de saúde e pacientes, como também com os serviços externos. Os atores deste sistema interagem com o mesmo através de duas aplicações que suportam as atividades diárias dos profissionais de saúde e simplifica a tomada de decisão, *Whiteboard* e *ePatient Mobile*.

A comunicação com os serviços externos é através de *web services* disponibilizados pelo *ePatient*, sendo que estes não contêm qualquer tipo de segurança nem informação sobre quem efetua os registos, pedidos ou alterações. Toda a lógica dos *web services* está na componente *ePatient Core*.

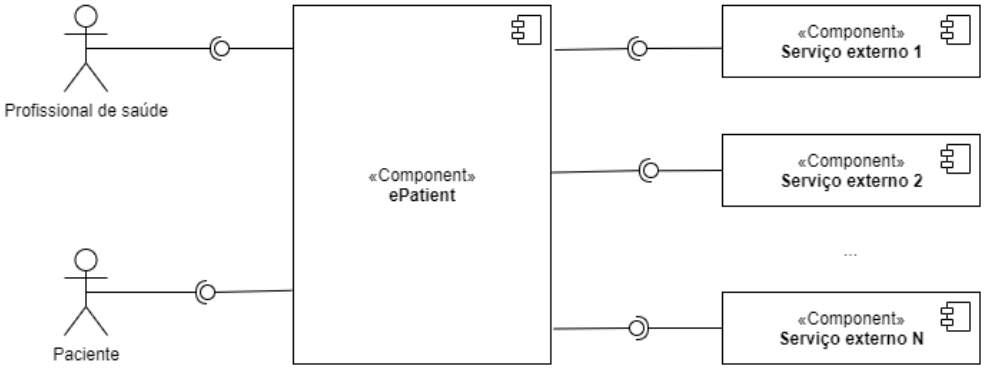

Figura 1 – Representação geral das comunicações do ePatient

Na Figura 2 encontra-se a representação mais detalhada da componente *ePatient*. Nesta é possível ver tantos os componentes como as ligações existentes no seu interior. Existindo então dois componentes responsáveis pelo tratamento da informação que é recebida, pelas aplicações que interagem com os utilizadores, a base de dados, que armazena todos os dados registados na aplicação, e, por fim, a parte mais importante para o projeto o *ePatient Core*, sendo que é neste que está presente toda a lógica por detrás do *ePatient*.

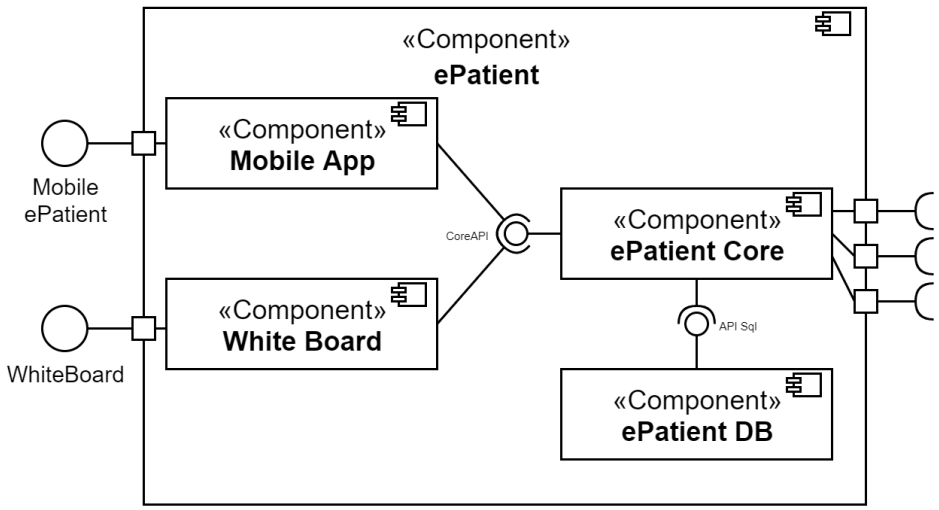

Figura 2 – Representação detalhada ePatient

O componente *ePatient Core* é responsável manter a lógica de negócio do sistema *ePatient* e realizar para todas as comunicações com os sistemas externos, sendo neste que residem todas as definições para que estas comunicações sejam possíveis. A manutenção deste componente é complicada dado que todas as definições são implementadas em código, sendo que se torna um processo complexo qualquer tipo de alteração necessária. Um exemplo de boa prática para combater esta implementação seria a parametrização de todas as definições.

2.2.1 Whiteboard

Whiteboard é uma aplicação desenvolvida em .NET que consiste num portal direcionado para os profissionais de saúde. Esta aplicação é responsável por monitorizar o estado dos pacientes, desde que estes chegam ao hospital até receberem a sua alta. Para isto contem informação pertinente de cada paciente como, por exemplo, (i) cama em que se encontra; (ii) medicação; (iii) alergias; (iv) doenças; (v) médico responsável pelo processo; (vi) evolução do estado clínico; (vii) alertas; etc.

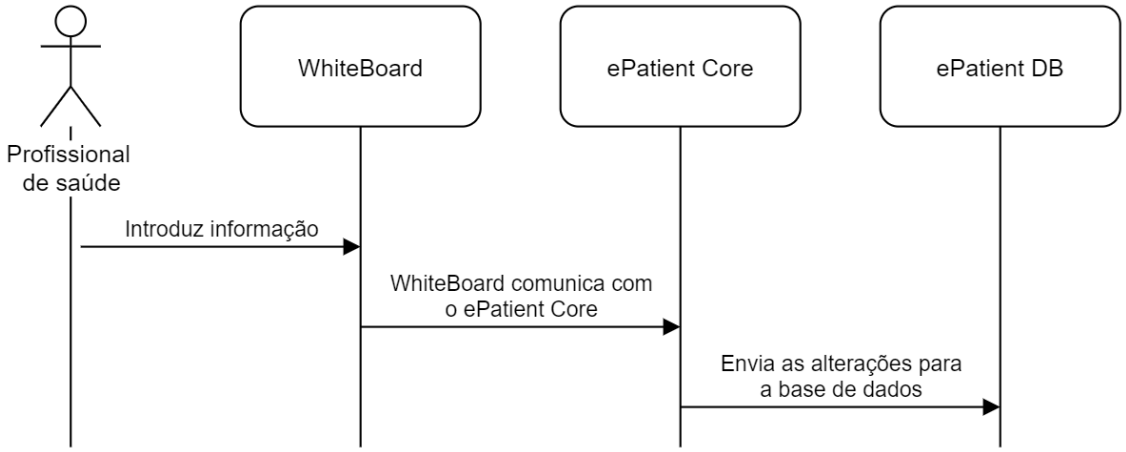

Figura 3 – Comunicação entre o profissional de saúde e a aplicação

Na Figura 3, está representada, de uma maneira simplificada, o processo de introdução de informação por um profissional de saúde no *ePatient*, no portal direcionado para o mesmo. Este submete informação relevante, esta informação é trata no *ePatient Core* e só depois guardada na respetiva base de dados.

O *ePatient Core* pode também comunicar com serviços externos, mas, a título de exemplo, foi considerado um exemplo simplificado.

Concluindo também que o objetivo do projeto não passa por introduzir alterações no funcionamento do *WhiteBoard* e qualquer alteração feita no *ePatient* não deverá impactar esta aplicação.

2.2.2 ePatient Mobile

ePatient Mobile, desenvolvida em *Xamarim*, é direcionada aos utentes internados no hospital. Esta aplicação foi desenvolvida exclusivamente como sendo uma aplicação para instalação em dispositivos móveis (e.g. *tablets* junto às camas do hospital).

Deste modo, o utente é capaz de dar *feedback* do seu estado atual, se se encontra com dores ou com algum desconforto, como também tem a possibilidade de pedir algo que esteja em falta, como comida, bebida, entre outras coisas.

Na Figura 4 está representado, de uma maneira simplificada, o processo de envio de *feedback* de um paciente, através da aplicação direcionada a este, para o profissional de saúde. Quando introduzida a informação, esta é enviada para o *ePatient Core*, sendo tratada e distribuída a partir deste. A distribuição é feita consoante a necessidade do pedido, sendo neste exemplo apenas um pedido simplificado. Consoante isto, envia um sinal de alerta para o *WhiteBoard* e, ao mesmo tempo, faz o registo desse mesmo *feedback* na base de dados.

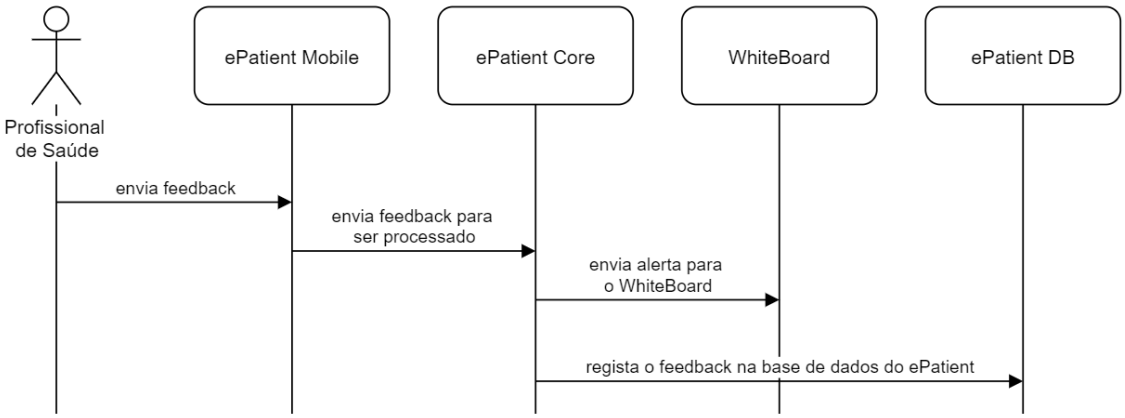

Figura 4 – Comunicação entre o profissional de saúde e a aplicação

Concluindo também que o objetivo do projeto não passa por introduzir alterações no funcionamento do *ePatientMobile* e qualquer alteração feita no *ePatient* não deverá impactar esta aplicação.

2.2.3 ePatient Core

Como referido anteriormente, no ponto 2.2, o *ePatient Core* é a componente responsável por toda a lógica dos *web services*, isto é, esta componente é responsável pelo tratamento de dados, e consoante o pedido que recebe/consome, responde às necessidades deste, sejam comunicações com a base de dados ou com sistemas externo.

De seguida é então feita uma análise mais aprofundada ao *ePatient Core*, desde os seus componentes internos até ao esclarecimento de como são feitas as comunicações com os componentes externos.

No diagrama da Figura 5 é possível verificar todas as comunicações existentes entre os diferentes componentes dentro o *ePatient*.

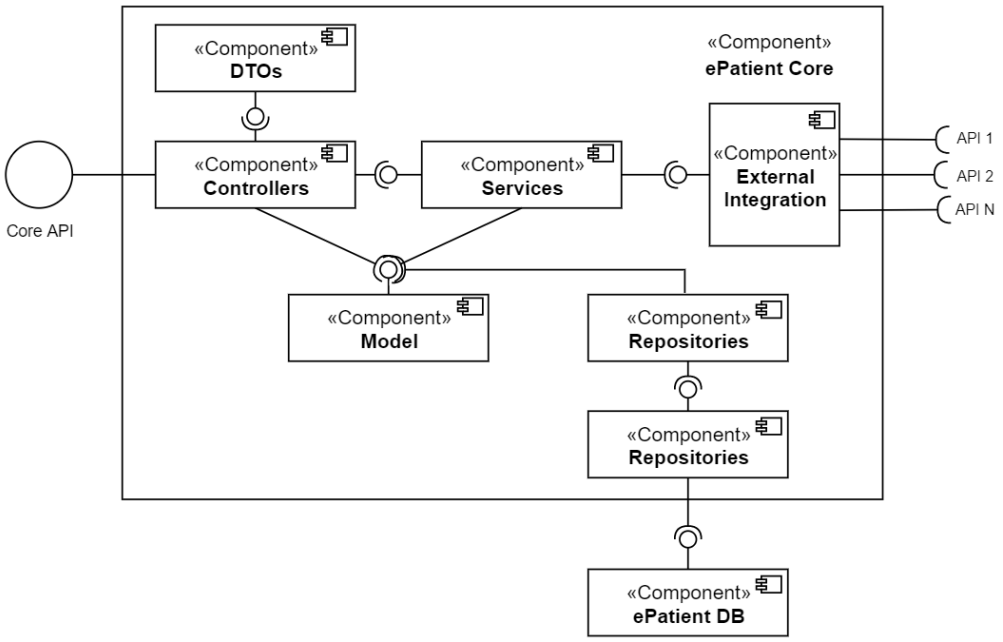

Figura 5 – Análise ePatient Core

O componente *Controllers* funciona como ponte entre a interface gráfica (acrónimo do inglês UI – *User Interface*) da aplicação e a parte responsável por toda a lógica. É neste componente que se transforma as mensagens, com o auxílio da componente DTOs, para que todas sigam o mesmo formato.

O componente DTOs contém todos os modelos de mensagens possíveis a serem transmitidas. O componente *Model* contém a estrutura dos objetos e por fim o componente *Repositories* contem a informação de comunicação com a base de dados.

O componente *Services* contem grande parte da lógica de negócio, sendo neste que se efetuam todo o tipo de validações. Este componente comunica com o *External Integration* que contem toda a lógica de integração entre o *ePatient* e os serviços externos, é aqui que residem todas as configurações de comunicações externas.

Analisando mais ao detalhe o componente *External Integration*, representado na Figura 6 de um modo abstrato, é importante perceber que, cada conexão a um serviço externo, exige a configuração de um adaptador. É essa configuração que torna a escalabilidade do ecossistema *ePatient* mais complicada devo ao facto de requerer a respetiva codificação do adaptador, o que envolve muito provavelmente uma duplicação de coisas já desenvolvidas noutros adaptadores.

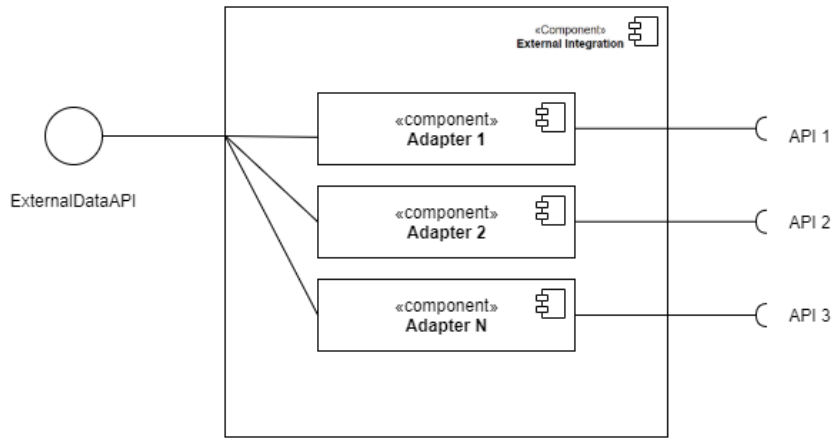

Figura 6 – Representação abstrata de ligações entre o ePatient e sistemas externos

Analisando agora a composição real do componente *External Integration*, representado na Figura 7, é possível verificar neste momento a existência de três adaptadores.

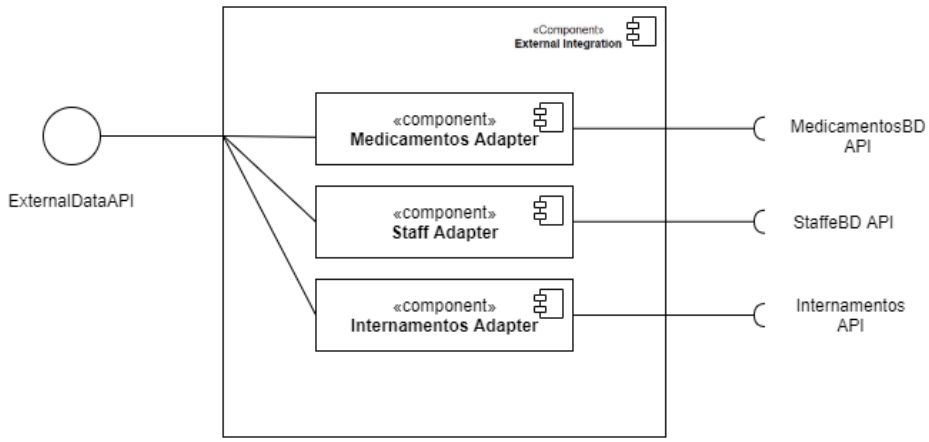

Figura 7 – Comunicação entre o External Integration e os sistemas externos

O primeiro adaptador, Medicamentos *Adapter*, contém a configuração que permite comunicar com a API MedicamentosBD. Nesta API está armazenado toda a informação de medicamentos do hospital. Esta comunica também com serviços externos para que seja possível, por exemplo, repor stocks, entre outras operações.

O segundo adaptador, Staff *Adapter*, contém a configuração que permite comunicar por sua vez com a API StaffBD. Nesta API está armazenada toda a informação dos profissionais de saúde. Esta API, como a API MedicamentosBD, comunica também com serviços externos, como por exemplo, aplicações de recursos humanos, entre outros.

O último adaptador, Internamentos *Adapter*, contém a configuração que permite comunicar com a API Internamentos. Esta API Internamentos trata-se de um Gestor de Internamentos, isto é, onde se processa um pedido de internamento.

Quando é necessário internar um paciente, e o gestor efetua esse pedido ao *ePatient*, é necessária a comunicação com o componente de internamento para que seja verificada a disponibilidade do hospital, tanto a nível de infraestruturas como de pessoal médico capaz de tratar o paciente. Esta comunicação é feita através de *web services*, no momento do pedido de internamento, entre o *ePatient* e a API Internamentos.

Com isto, podem surgir alguns problemas dado que todos os pedidos estão dependentes do mesmo componente, *External Integration*.

Na Figura 8 é possível verificar um problema existente com esta solução devido a essa dependência. Este caso trata a necessidade de efetuar atualizações em três bases de dados diferentes com dados redundantes nas três. Quando adicionado um novo paciente, caso de uso dado neste exemplo, é então necessário criar o seu registo na base de dados de pacientes, é também necessário criar o registo na base de dados de internamentos e por fim o seu registo na base de dados de medicamentos.

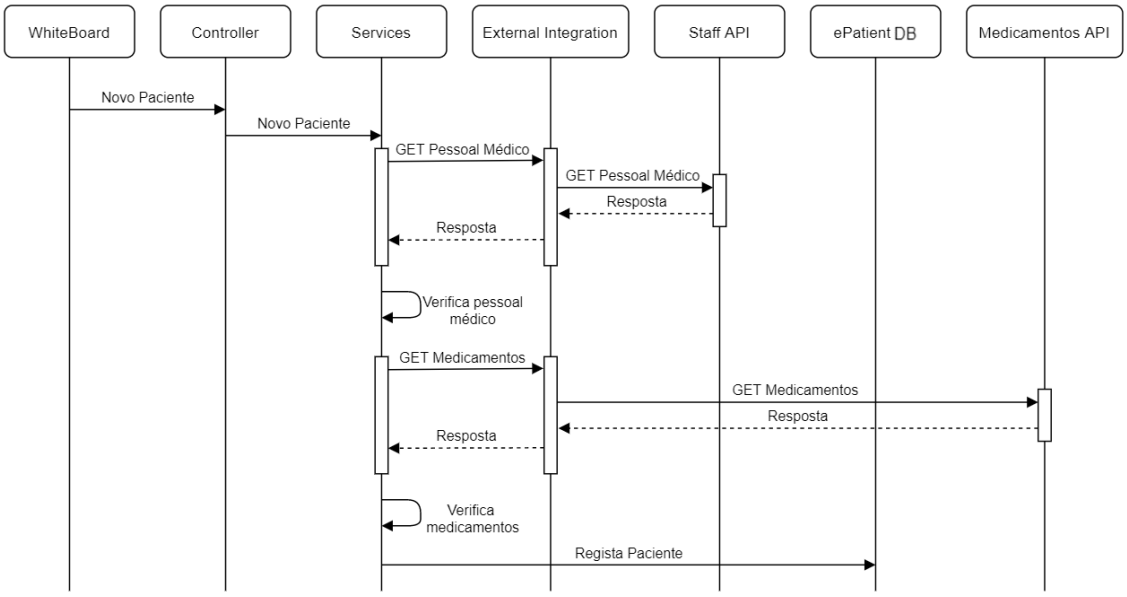

Figura 8 – Vista de cenário adicionar novo paciente

Outro problema que pode surgir é quando adicionada uma nova cirurgia, representado no diagrama da Figura 9. Este diagrama demonstra também a necessidade de múltiplas comunicações entre o portal e as bases de dados existente para que uma nova cirurgia seja registada. Para este registo, inicialmente é necessário verificar se existem salas de cirurgia disponíveis, para depois ser adicionado um médico destinado a esta cirurgia.

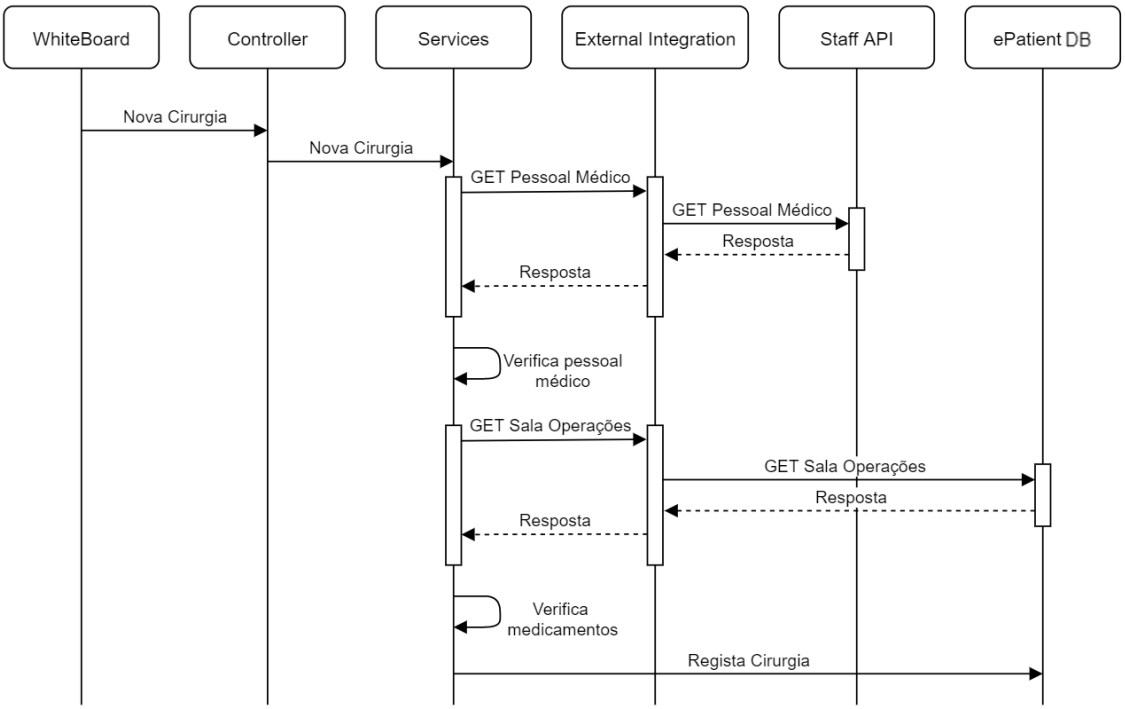

Figura 9 – Vista de cenário adicionar nova cirurgia

Nestas últimas duas figuras, Figura 8 e Figura 9, foi possível verificar as diversas comunicações entre o portal e o componente *External Integration*. Estas diversas comunicações traduzem-se num número elevados de pedidos à componente, trazendo também problemas de segurança. Com isto, é necessário retirar alguma parte da lógica e responsabilidade ao componente *ePatient Core*.

2.3 Descrição do Problema e da Área de Negócio

O desenvolvimento desta *framework* está inserida na área da saúde, uma área particularmente importante pois lida com a saúde do ser humano. De forma a minimizar erros, que podem ter custos excessivamente elevados, é necessário ter especial atenção em certos pontos, como testes unitários e de aceitação bem definidos, e o uso de normas HL7.

Neste momento, e como referido anteriormente no problema (subsecção 1.2), a conexão ponto a ponto levanta vários constrangimentos. Constrangimentos esses como a falta de controlo de tráfego, a demora na entrega do produto, a existência de alguns erros sem qualquer possibilidade de resolução e a possibilidade de tratamento de alguns problemas a custos elevados. Todos estes constrangimentos podem ser ultrapassados com o desenvolvimento de uma *framework* de integração sistematizada e sólida.

2.3.1 StakeHolders

Existem vários *Stakeholders* no que toca a este projeto como:

- Responsável do projeto, que pretende ter uma solução fidedigna e de fácil implementação em qualquer hospital;
- Cliente, que pretende que a rapidez de instalação e a sua configuração seja o mais rápida possível;
- Profissionais de saúde, este querem ter sempre a informação atualizada.

2.3.2 Sistemas a interligar

Neste momento não existe um sistema específico, cliente interessado em adquirir o *ePatient*, que necessita de interligação com o mesmo.

Apesar disto, e para que seja possível verificar o potencial da nova *framework* desenvolvida para o *ePatient*, foram criadas, a título de exemplo, bases de dados hospitalares e APIs para comunicarem com o sistema. Estas bases de dados contêm desde o pessoal medico, ao staff, os medicamentos e todas as informações a serem fornecidas por um hospital.

2.4 Inquérito

Após uma análise mais detalhada ao *ePatient*, e ter sido verificado que o uso de uma ferramenta de *middleware* é mais vantajoso para a gestão de comunicações entre sistemas distintos, foi realizado um pequeno inquérito. Este inquérito foi realizado no âmbito de perceber se realmente uma ferramenta de *middleware* seria a mais adequada, com destino a pessoas mais experientes no ramo de integração de sistemas, tendo sido realizado apenas a colaboradores da Deloitte.

Neste subcapítulo é apresentado o desenvolvimento do inquérito para se perceber se a escolha de uma ferramenta de *middleware* é a mais indicada. Para isto, o inquérito baseia-se na comparação do sistema de integração atual, um sistema mais rudimentar de integração ponto a ponto, e um sistema implementado através de uma ferramenta de *middleware*, sendo este o responsável pelas comunicações entres os vários sistemas.

2.4.1 Preparação e Configuração

O inquérito desenvolvido teve em vista oito profissionais da área de engenharia informática, especializados na área de integração de sistemas, tendo já tido contacto com os vários sistemas de integração apresentados no questionário.

Este inquérito aborda, no geral, três tópicos: a facilidade de aprendizagem; a arquitetura de desenvolvimento e a rapidez de desenvolvimento.

Tendo em conta então as duas possíveis abordagens, e focando mais na capacidade de cada implementação, e não tanto no contexto deste projeto o inquérito continha as seguintes seis questões:

1. Quão fácil é aprender integração ponto a ponto?
2. Quão fácil é aprender integração por *middleware*?
3. Como avalia a arquitetura de um sistema quando adota integração ponto a ponto?
4. Como avalia a arquitetura de um sistema quando adota ferramentas *middleware* para integração de sistemas?
5. Quão rápido é o desenvolvimento de um sistema adotando integração ponto a ponto?
6. Quão rápido é o desenvolvimento com *middlewares* de integração de sistemas?

Todas as respostas são respondidas de um modo quantitativo, de 0 a 10, sendo que, em cada contexto, o 0 significa insatisfeito e o 10 significa muito satisfeito.

2.4.2 Resultados Obtidos

Após a recolha das respostas obtidas, foi então tempo de tratar dos resultados. Na Tabela 1 é possível verificar a pontuação atribuída a cada questão por parte de cada sujeito interveniente neste inquérito.

Tabela 1 – Resultados do Inquérito

	Questão 1	Questão 2	Questão 3	Questão 4	Questão 5	Questão 6
Sujeito A	6	4	3	8	5	8
Sujeito B	4	3	2	9	2	8
Sujeito C	6	6	4	8	3	7
Sujeito D	7	5	5	9	6	6
Sujeito E	7	6	3	8	7	8
Sujeito F	6	4	2	7	5	7
Sujeito G	6	5	5	8	4	7
Sujeito H	6	6	5	7	3	9
Total	48	39	29	64	35	60

Já na Tabela 2 é possível verificar a pontuação média respondida a cada questão, bem como o valor mínimo e máximo dado nessa mesma questão.

Tabela 2 – Análise aos resultados do inquérito

	Média	Mínimo	Máximo
Questão 1	6	4	7
Questão 2	4.875	3	6
Questão 3	3.625	2	5
Questão 4	8	7	9
Questão 5	4.375	2	7
Questão 6	7.5	6	9

2.4.3 Análise dos Resultados

Na Tabela 1 estão representadas pontuação atribuída a cada questão por parte de cada sujeito interveniente neste inquérito. Nesta tabela é possível verificar que, em relação a uma integração ponto a ponto, só foi obtida uma pontuação mais elevada, em relação à integração de sistema, no que toca à facilidade de aprendizagem. Em relação à arquitetura do sistema e ao desenvolvimento do mesmo, a integração mais bem qualificada, pelos mesmos intervenientes, é a integração de sistemas.

As duas primeiras questões focam-se apenas na facilidade de aprendizagem de cada tipo de integração e, como é possível verificar na Tabela 2, a média de respostas é superior no que toca à integração ponto a ponto. Isto deve-se ao facto de a integração de sistemas ser uma área relativamente recente e ser pouco abordada em contexto académico. A integração ponto a ponto torna-se mais acessível pois baseia-se muito em código no seu desenvolvimento.

Em relação às questões três e quarto, direcionas à arquitetura, é possível verificar na Tabela 2 uma diferença considerável entre as medias de resposta. A integração de sistemas tem uma arquitetura de fácil compreensão e mais organizada em relação à arquitetura ponto a ponto, melhorando a sua manutenção.

Por fim, nas questões cinco e seis, o foco reside na rapidez de desenvolvimento de cada uma das integrações, existindo também uma diferença considerável na média de respostas, como é possível verificar na Tabela 2. Fica claro que o desenvolvimento de integração de sistema é mais rápido em comparação com a integração ponto a ponto. Uma das razões para que o desenvolvimento desta seja consideravelmente mais rápido deve-se ao facto de conter uma arquitetura simples e intuitiva.

Concluindo assim que, segundo este questionário, a aprendizagem de integração de sistemas é mais complicada, compensando mais tarde na sua arquitetura e desenvolvimento, tornando-se mais acessível e de fácil compreensão.

2.5 Objetivos técnicos

Após analisar a solução existente é necessário abordar os requisitos técnicos essenciais para o desenvolvimento desta dissertação.

Inicialmente é necessário explorar as várias ofertas existentes no mercado, isto é, tecnologias que oferecem serviços de integração de sistemas, e mais importante, que se adequem aos requisitos existentes. Existem dois cuidados extra a ter na escolha da nova tecnologia de integração. O primeiro cuidado a ter é sobre a elevada carga de pedidos que existe, sendo necessário uma solução capaz de contornar esse problema. O segundo problema recai na segurança necessária no tratamento destes pedidos, sendo necessária uma tecnologia capaz de monitorizar e garantir a segurança dos mesmos.

Após a escolha da tecnologia mais indicada ao problema, é então importante o desenvolvimento de um design claro e explicito, isto para que, no momento do desenvolvimento e implementação da solução, não existam quaisquer dúvidas.

Por fim, e uma das partes mais importantes do documento, quando desenvolvida e implementada a solução é necessário a realização de testes à mesma para atestar a validade da solução construída.

2.6 Sumário

Neste capítulo foram abordados os temas de estado da arte e análise do *ePatient*.

Na primeira parte deste capítulo foram abordadas as diferentes aplicações que tem objetivos comuns ao *ePatient* e a maneira como faziam a sua integração.

Na segunda parte do mesmo realçou-se os problemas referentes a soluções desenvolvidas com recurso a integração ponto a ponto, e ao mesmo tempo foi feito um *overview* sobre o ePatient.

Capítulo 3

Integração de Aplicações

Nos dias de hoje é cada vez mais importante e necessário fazer integração de vários programas construídos, sendo que esta integração deve ser feita de maneira coesa e escalável. Devido a essa grande necessidade, por parte de quem desenvolve, e para que a integração possa ser feita de uma maneira eficaz, existem diversas tecnologias especializados nesta área como, por exemplo Mulesoft, WSO2, TIBCO, Kong, entre outros que, embora façam um trabalho muito parecido, diferenciam-se em detalhes que, apesar de pequenos, fazem (ou podem fazer) a diferença.

Deste modo, como a oferta é bastante grande, não é fácil escolher o *software* que melhor se adequa ao que é necessário para o problema em questão. Devido à existência desta dificuldade para escolher aquele que mais se adequa ao problema, é necessário então reunir todas as características, tanto positivas como também negativas, de cada software para depois então escolher a mais apropriada ao desenvolvimento deste projeto.

3.1 Integração

Como já foi referido, existem cada vez mais sistema informatizados que necessitam de comunicar entre si.

A Figura 10 representa um exemplo de vários serviços que necessitam de comunicar entre si. Este problema é frequentemente encontrado nos dias de hoje entre sistemas que pretendem comunicar entre si. Com isto, é então necessário encontrar uma solução eficaz capaz de fazer essa mesma comunicação.

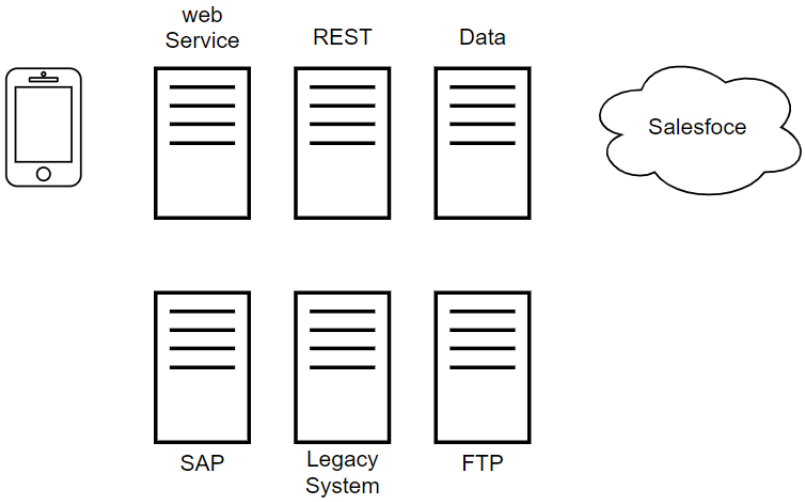

Figura 10 – Diferentes sistemas as tentar comunicar entre si

A integração é, tipicamente, realizada de duas formas: (i) ponto-a-ponto e/ou (ii) através de um intermediário. Independentemente da forma, pode-se ainda dizer que a integração é feita pela aplicação de um ou mais estilos de integração e tem em consideração vários aspetos/necessidades, tais como:

- *Network capabilities* - habilidade usar a rede para efeitos de segurança de acessos;
- *Protocol Adapters* - Uso de protocolos para efeitos de comunicação;
- *Message handlers* – trata de gerir os pedidos feitos;
- *Data transformation* - recebe uma mensagem e traduz para que o sistema final possa aceitar essa mesma mensagem;
- *Decomposition e Recomposition* – Decompõe a mensagem em "sub" mensagens a serem enviadas para diferentes sistemas;
- *Routing/navigation* – direciona uma mensagem tendo em conta o seu conteúdo;

3.1.1 Integração ponto-a-ponto

Integração ponto-a-ponto era mais utilizada quando existia apenas uma aplicação a comunicar com outra aplicação. Mas anos passaram e foi surgindo a necessidade de mais aplicações comunicarem entre si. Esta situação foi ultrapassada através da codificação das configurações necessárias para efetuar essas mesmas comunicações, resolvendo assim a necessidade de comunicação entre vários sistemas, mas tornando-as redundantes e pouco seguras.

Na integração ponto-a-ponto, um sistema tem apenas informação referente a si mesmo e com quem estabelece comunicação. Posto isto, dois sistemas que comunicam entre si têm grande acoplamento, isto apenas resulta se o número de sistemas for reduzido.

Na Figura 11 é possível ver, se todos os sistemas necessitarem de comunicar uns com os outros, o número de ligações que terão de existir, e com isto, todas as configurações necessárias que terão de ser realizadas para que todos estes serviços fiquem interligados.

Nesta mesma figura todas as responsabilidades que foram descritas no subcapítulo anterior ficam associadas a cada sistema, o que causa uma repetição de código e configurações, que um sistema intermédio poderia resolver.

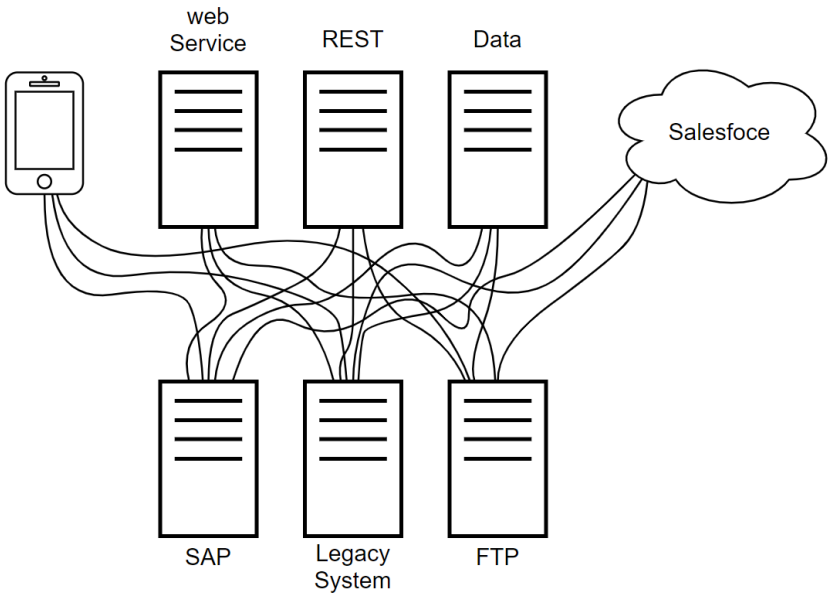

Figura 11 – Exemplo comunicação sem intermediário

3.1.2 Integração Gerida por Intermediário

O uso de um intermediário é uma maneira mais flexível para integrar uma aplicação. No caso de um intermediário de integração, é um estilo de arquitetura de software que é usado para implementar as iterações e comunicações entre aplicações. Esta integração é alcançada com o encapsulamento e expondo cada funcionalidade da aplicação como um conjunto de recursos reutilizáveis.

Assim cada aplicação tem a necessidade de integrar diretamente com outra aplicação, estas integram-se através da infraestrutura do intermediário de integração como é ilustrado na Figura 12.

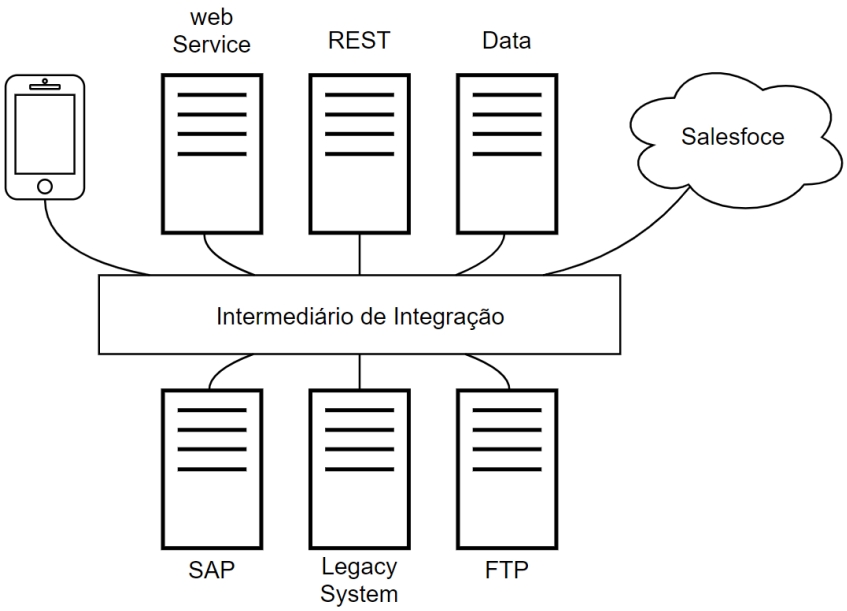

Figura 12 – Exemplo de comunicação com recurso ao ESB

Na figura acima representada toda a informação referente a aos pontos no subcapítulo 3.1 fica alojado na camada intermediaria denominada de “Intermediário de Integração”.

3.1.3 Considerações Finais

Após uma análise comparativa entre integração ponto a ponto e a utilização de um intermediário entre os diferentes sistemas, fica claro as vantagens conceptuais de se utilizar um componente intermédio tanto em termos arquiteturais como também para manutenção da camada de integração.

3.2 Conceitos de Integração Intermediada

Este subcapítulo é direcionado à introdução de alguns conceitos que serão referenciados e utilizados nos próximos capítulos. Esta introdução especifica a forma como os mesmos devem ser compreendidos e interpretados no âmbito deste documento.

3.2.1 Message

Message refere-se ao conjunto de informação que é enviado e recebido por APIs, nesta mensagem encontra-se toda a informação necessária para a execução do pedido. Esta mensagem pode conter coisas como o caminho, *Headers, Payload,* entre outros [10].

3.2.2 Message Broker

Message Broker funciona como uma espécie de intermediário entre dois sistemas que comunicam. Este traduz mensagens de quem envia para mensagens capazes de serem lidas por quem recebe. A definição de *Broker*, do nome *Message* Broker, é um serviço usado por quem envia mensagens e por quem as recebe, usando protocolos normais de comunicação.

De uma forma mais geral, é um programa que toma conta da entrega das mensagens. Quando visto mais ao detalhe, tem funções como validação, transformação e redireccionamento de mensagens [11].

3.2.2.1 Reutilização

Reutilização é uma das principais vantagens quando se adota uma solução de integração, porque significa um benefício na eficiência no desenvolvimento e manutenção do sistema. Os serviços deverão ser desenhados para que estes sejam reutilizados em diferentes cenários de negócio.

Por exemplo, se uma empresa desejar expor informações a terceiros, por exemplo, parceiros comerciais de um site de reservas de feriados, a mesma API será usada por todos os parceiros para conectar e compartilhar informações especificas no site do cliente. Caso haja alguma personalização necessária para um cliente em específico, uma nova API de experiência será criada como um *wrapper* na parte superior da API existente, reutilizando a funcionalidade e as filtragens existentes de acordo com os requisitos do cliente [12].

3.2.2.2 Data Transformation

Data Transformation é o processo de conversão de informação de um determinado formato/estrutura para informação de outro tipo de formato/estrutura. É um dos aspetos mais importantes no que toca à integração e gestão de informação.

Data Transformation pode ser de dois tipos, simples ou complexo, baseado no tipo de transformações pretendidas. É tipicamente efetuado usando uma mistura de transformações manuais e automáticas [13].

É possível dividir em certos passos o processo de transformação, desde o *Data discovery* até ao *Data Review*.

- *Data discovery*: é o primeiro passo no processo de transformação. Normalmente, os dados são criados usando ferramentas de criação de perfis. Isto faz com que o sistema entenda melhor a estrutura e as características dos dados. Depois disto, é decidido como os dados serão transformados.
- *Data mapping*: é o processo responsável pela definição dos campos individuais e como estes serão transformados.

- *Code generation*: é o processo responsável pela geração de código que irá transformar a informação, baseado nas regras do *Data Mapping* [14].
- *Code execution*: momento em que o código gerado nos processos anteriores é executado, sendo aplicando à informação que se quer transformar. A execução deste código poderá ser curta, sendo neste ponto em podem ser aplicadas transformações manuais.
- *Data review:* é último passo, trata-se do processo responsável por assegurar que o resultado da transformação vai de acordo com os requisitos. Tipicamente executado pelo responsável de negócio que verifica se existe algum tipo de erro. No caso de existir, volta a ser enviado para o desenvolvedor para que esses erros possam ser corrigidos [15].

3.2.2.3 Complexidade e orquestração

Um intermediário de integração tem a capacidade de orquestrar blocos de funcionalidades fracamente acoplados, reduzindo assim a complexidade e reutilizando recursos individuais para suportar processos de negócio mais amplos ou complexos.

No fundo o intermediário de integração é então um orquestrador de serviços, agindo como uma figura central, este coordena a chamada de outros serviços para compor uma função de maior granularidade. É aqui que está a diferença para a coreografia, em que uma API já está preparada de antemão para receber ou não uma determinada mensagem e dispararem outras ações [16].

3.2.3 API Management

Com uma solução de integração por camadas a funcionar, um dos primeiros desafios é se ser possível ter uma visão clara do ecossistema de uma API, assim como conseguir geri-la. Sendo necessário APIs responsáveis por essa gestão. Fazendo assim que exista uma *API Portal* e uma *API Gateway* [17].

3.2.3.1 API Portal

Enquanto fazer a ligação entre aplicações via APIs é crucial, não deve ser menosprezado a funcionalidade de expor e publicar estas APIs para os respetivos consumidores. Para garantir que os serviços pedem ser efetivamente descobertos e interpretados, um API Portal dá a capacidade de criar a ponte entre os consumidores de APIs e os que as disponibilizam. Facilitando assim o acesso aos ciclos de vida de uma API.

Para os consumidores de APIs, um API Portal é o lugar para registar as suas aplicações, para partilhar e interagir com a documentação de uma API, para dar feedback sobre a qualidade da API e para reportar erros [18].

Um típico API Portal pode ser resumido aos seguintes elementos:

- *API catalogue*, descreve que APIs estão disponíveis para consumo, versões, exemplos nos métodos, incluindo documentação com autenticação, casos de uso e aplicações no mundo real.
- *API Lifecycle*, abrange todas as versões de um produto, incluído aquelas que já não tem suporte, permitindo assim ser visível a visualização da API em questão.

3.2.3.2 API Gateway

Um API Gateway é um componente que reside na frente das APIs internas como um único ponto de acesso para todos os clientes. Este é capaz de tratar de protocolos, de tratar da parte de segurança e trabalhar por redireccionamento através dos pedidos. Em vez de disponibilizar um *one-size-fits-all* API, o API Gateway consegue expor diferentes APIs por cliente [18].

A utilização de API Gateway é necessária devido às seguintes razões:

- A granularidade dos serviços é muitas vezes diferente do que o cliente necessita, exigindo que este faça um apanhado de vários serviços individuais, se tal *Gateway* não existir;
- Diferentes clientes podem requerer diferentes informações e detalhes;
- A performance da rede é diferente dependendo do tipo de cliente, por isso uma aplicação do lado do servidor pode fazer vários pedidos ao serviço de *backend* sem impactar a experiência de utilizador quando comparado com o cliente mobile.

3.2.3.3 API Versioning

Quando se gere APIs há a possibilidade de criar múltiplas versões de uma API, o que é particularmente vantajoso no desenvolvimento de uma arquitetura incremental.

Uma versão inicial de uma API com o mínimo de desenvolvimento para suportar requerimentos específicos do negócio, podem ser criados para serem publicados. Enquanto ao mesmo tempo, uma nova versão da mesma API pode ser desenvolvida em paralelo, com todas as funcionalidades para suportar todos os requisitos. Quando a versão posteriormente desenvolvida estiver pronta para ser usada, os sistemas que usavam a API inicialmente criada simplesmente tem de alterar a versão que estiverem a usar para deste modo ter acesso a novas funcionalidades [19].

Em termos de aplicação de boas práticas, um dos tipos esquemas seguidos para ajudar a gerir a API é o uso da *semantic versioning*, apresentado na Figura 13.

Figura 13 – *Semantic Versioning*

MAJOR – Uma mudança *major* altera por completo as versões até aqui então existentes. Isto significa que clientes que usam uma versão antiga da API não podem usar a nova versão enquanto não efetuarem a atualização na versão em que estão a fazer a chamada.

Exemplos algumas mudanças que despoletam a necessidade de criar uma nova versão são:

- Remover, renomear, mover entidades de uma API como:
  - *Endpoints*;
  - Métodos HTTP;
  - Alterações no *body* ou outros parâmetros;
  - Alterações na autorização de acesso.
- Novos requisitos que tem de ser preenchidos antes de efetuar a chamada;
- Alterar uma já presente orquestração.

MINOR – Uma mudança *minor* é quando novas funcionalidades são adicionadas à API, sendo que estas novas funcionalidades são compatíveis com a versão anterior. Esta mudança influencia, mas não põe em causa, o funcionamento das funcionalidades de um API.

PATCH – Um *Patch* são correções de bugs, melhoramentos na descrição e/ou exemplos nas especificações da API.

3.2.3.4 Throtling

Throtling é uma técnica para introduzir uma espécie de limite de pedidos a um determinado serviço. Sem esta proteção a camada de integração corre o risco de ter serviços sobrecarregados devido ao número elevado de pedidos [20].

Definir estratégias de *throtling* na framework de integração permite:

- Ter aplicações constantes em termos de "*up* and *running*", tendo em conta que um único cliente não esta a sobrecarregar aplicações com pedidos;
- Limite de pedidos em diferentes níveis, *resource*;
- Previne que um simples serviço monopolize os recursos disponíveis.

3.2.3.5 Gestão de Políticas

Uma política pode ser vista como um módulo que implementa uma função especifica. Políticas são designadas para adicionar tipos comuns de gestão de capacidades de uma API, numa fácil e pratica maneira. Políticas permitem implementar serviços como segurança, limite de trafego entre outros [21]. Estas políticas podem ser dividas em dois grandes tipos: integradas e costumáveis.

Políticas integradas

Políticas integradas refere se a políticas que estão já implementadas sem a necessidade de se configurar nada, podendo ser aplicadas diretamente. Ao utilizar políticas integradas e atribuí-las a uma API, os desenvolvedores podem assegurar que o ambiente em que estão a trabalhar usam as mesmas políticas, aumentando assim a consistência e reduzindo erros.

Um exemplo de políticas integradas é:

- *Message logging policy* – mensagens costumáveis definidas pelos desenvolvedores com a informação disponível entre políticas.

Custom policies

Por sua vez, existe uma habilidade para desenvolver políticas à medida que podem ser aplicadas pelas APIs todas ou em apenas algumas destas. Estas políticas são aquelas que os desenvolvedores podem implementar com a intenção de estender funcionalidades existentes ou até aplicar lógica que não está logo definida.

Inicialmente configurar estas políticas podem criar algum atraso na implementação, mas a longo termo irá ajudar no desenvolvimento e redução do tempo dedicado ao desenvolvimento.

3.2.4 Framework de Integração

O principal foco desta dissertação é o desenvolvimento de uma camada de integração como já foi referido anteriormente, mas para tal, é necessário saber ao certo o que é essa camada e quais as suas características que a fazem ser tao importante.

Uma *framework* de integração, permite que um grande número de serviços possa ser processado num curto espaço de tempo, principalmente através do reuso de funcionalidades.

Refere-se à implementação de uma solução que é usada para tomar conta de todos os serviços de integração numa maneira estruturada, com uma clara definição das camadas funcionais a serem implementadas e regras bem definidas. Esta *framework* dá uma base de configurações técnicas comuns, como também capacidades transversais e tratamento de erros [22].

Nesta *framework* os serviços são repartidos em camadas abstratas que refletem uma segregação lógica de tarefas alinhadas com os requisitos de negócio. As seguintes camadas são as camadas padrão definidas:

- ***Experience Layer***: é a camada pública, é o ponto de entrada para a camada de integração. Esta camada expõe diferentes funcionalidades para clientes e permite diferentes pedidos serem consumidos por diferentes canais. Permite além do que foi anteriormente referido que a informação seja sincronizada por uma variedade de serviços e o acesso a objetos ou recursos em múltiplos sistemas.

- ***System Layer***: também referida como camada técnica, interage diretamente com todos os serviços de backend. Permite assim o consumo de informação, como por exemplo bases de dados.

- ***Process Layer***: é a camada que efetua a ligação entre as camadas de *Experience* e System, esta camada é onde toda a logica de negócio e orquestração deverá estar, dando assim uma clara representação das funcionalidades dos componentes e interagir com eles. Os serviços nesta camada interagem e fazem a modelação de dados.

3.2.4.1.1 Framework Capabilities

A *framework* deverá implementar um conjunto de quatro funcionalidade, estas são: *Logging*; tratamento de erros; auditoria e segurança. Estas funcionalidades encontram-se descritas nos seguintes pontos:

Logging

Deverá incluir a capacidade de ter um sistema de *logs*, que podem seguir regras predefinidas aquando da configuração da solução. Ter uma base de dados com todos os *logs* é uma solução importante para aumentar a centralidade e relatórios sobre possíveis erros [22].

Logs na camada de integração deve conter apenas *metadata* das comunicações, essa *metadata* depende de projeto para projeto e o nível de detalhe, mas regra geral a campos que contem são:

- *Timestamp* – data e hora de registo da comunicação;
- *System* – o sistema que despoletou o pedido de log;
- *Type* – tipo de comunicação;
- *Correlation* id – identificação do pedido que permite a correlação com os outros passos na comunicação;
- *Action* – A ação associada à comunicação;
- *Message* – A mensagem transferida para o pedido.

Tratamento de Erros

Sistemas tratam dos seus erros independente dos outros. Isto resulta num grupo de potenciais códigos de erros e mensagens nesses erros.

A *framework* de integração trata de normalizar a todas as APIs os códigos e mensagens de erros. Uma mensagem de erro apropriada permite ao cliente saber que tipos de erros esperar e assim poder trata los da maneira que achar mais apropriada.

Dependendo do erro que ocorrer, uma tentativa automática de voltar a tentar submeter o pedido pode ser implementada. O comportamento da *framework* deverá ser específico para cada cenário e definir o alinhamento de conexão automático [22].

Auditoria

A *framework* dá também uma visibilidade de toda a infraestrutura com feedback dos componentes. Para o conseguir fazer, esta camada deve ser providenciada com ferramentas de monitorização que suportem equipas de desenvolvimento na identificação de problemas. Tais ferramentas devem ser configuradas para reduzir o tempo de identificação e resolução de problemas [22].

Segurança

Segurança deve estar presente na *framework* em toda a comunicação, usando a autenticação para verificar a veracidade dos comunicadores, como também mecanismos da autenticação como *tokens*.

Adicionalmente, procedimentos de encriptação e desencriptação devem ser implementados para manter a informação segura quando está em comunicação diversos serviços na camada de integração [22].

3.3 Tecnologias de Integração Intermediada

Existem diversas tecnologias de integração atualmente no mercado, o que se torna praticamente impossível fazer uma referência a todas estas. Existe então a necessidade de criar um critério de seleção. Neste caso optou-se por usar como base o gráfico *Forrester Wave* [23], representado na Figura 14. O *Forrester Wave* é um gráfico que representa a relação entre estratégia mais forte com o que cada *software* consegue oferecer atualmente. Neste gráfico estão apenas representados os *Strong Performers* e *Leaders* com uma presença relevante no mercado.

Figura 14 – Gráfico Forrester Wave

Para o desenvolvimento deste projeto existem certos pontos essenciais a ter em conta:

- Baixo Custo;
- Plataforma de gestão de APIs com acesso a análise de dados sobre consumos das mesmas;
- Capaz de integrar diversos componentes com alta fiabilidade;
- Suporte para HL7;
- Suporte de segurança;
- Suporte no desenvolvimento;

Seguindo estes pontos, é então possível ter uma visão mais clara do que procurar em cada um destes sistemas de integração.

3.3.1 Axway

Axway é um gestor de APIs focado na escalabilidade do mesmo, com Axway é possível criar e orquestrar rapidamente novas APIs, assegurar que só tem acesso à API apenas quem o desenvolvedor nomear, tem uma plataforma de analise em tempo real às APIs e tem também a existência de diversos portais que ajudam o desenvolvedor e cliente a fazer a escolha certa de que produto utilizar.

Tem também suporte para HL7, o que faz deste sistema de integração um potencial sistema a ter-se em conta para o desenvolvimento da *framework* [24].

3.3.2 CA Technologies

Pertencente à empresa Broadcom, CA Technologies oferece a possibilidade de modernizar a arquitetura de um sistema. Para isso apresenta três pontos chaves, desenvolvimento através de micro serviços, criar APIs numa perspetiva *Low-code* e uma vasta escolha de protocolos de segurança que se podem aplicar às APIs, tudo dependendo do nível de segurança que o cliente pretende ter. Contem ainda suporte para HL7 [25].

3.3.3 Apigee

Apigee é um software pertencente à Google, capaz de gerir API's, desenvolver serviços e analisar esses mesmos serviços. As funcionalidades do Apigee inclui o desenho da API, segurança, publicar, monitorizar e monitorizar para além de gerir micro serviços, usa Swagger para o desenvolvimento de API's, o que envolve um pouco de pesquisa sobre a mesma e tem a capacidade de se conectar com as normas HL7.

Em termos de segurança contem a opção de efetuar chamada apenas através de *tokens OAuth*, controlo de acesso a API's (ex. número de pedidos).

Por fim o preço é um pouco alto quando comparado com os seus concorrentes, segundo a informação dada pela fonte [26].

3.3.4 Software AG

Software AG é uma empresa com diversos sistemas, que no contexto desta dissertação há um foco na *Application Integration* que como o nome indica é um sistema de integração capaz de integrar qualquer tecnologia, assegurando suporte para os padrões de comunicação HTTP, XML, SOAP e WDSL. Em termos de segurança utiliza LDAP, *Active Directory* e plataformas como *Netegrity*.

Quanto à análise, este sistema tem um sistema integrado para analisar a saúde das APIs e onde estas estão alojadas. Por fim, este sistema tem integração com as normas HL7. No preço fica também um pouco caro, com a versão mais barata e simples começando nos 1000$ por mês [27].

3.3.5 Mulesoft

Mulesoft é um software capaz de desenvolver APIs usando *RESTful API Modeling Language* (RAML). Tem a capacidade de desenvolver regras podendo ser aplicadas e reutilizadas em APIs e parceiros comerciais. Traduzindo-se assim num software forte a nível de segurança para quem disponibiliza e consome a API, pois este só acede àquilo que lhe é permitido. Tem suporte para HL7 o que automaticamente também se traduz em normas de segurança fiáveis na comunicação entre sistemas de dados relacionados com saúde.

Tem como pontos positivos a existência de um *API Manager*, a existência de standards de saúde que podem ser extremamente valiosos no desenvolvimento das comunicações externas.

Um dos principais pontos negativos é não possuir um fluxo de trabalho *Business Process Management* (BPM) que irá ser explicado no próximo capitulo, sendo que as interfaces ficam um pouco aquém quando comparado com os seus concorrentes mais diretos, mas como no projeto para o qual se está a escolher a tecnologia não é um fator decisivo mas que no futuro poderá ser importante como por exemplo se for necessário efetuar a gestão desde a entrada do paciente no hospital até ao seu registo no hospital, mas reforçando novamente a ideia que não é para já o objetivo do trabalho [28].

Resumindo, esta tecnologia contém:

- *Routing* de mensagens através de regras, filtros e redireccionamento de mensagens.
- Transformação de dados.
- Hospedagem de Serviços.
- Mediação de serviços, permitindo a separação das responsabilidades conforme o design da lógica do negócio a ser implementado.

3.3.6 TIBCO

É um software que permite integrar, gerir e monitorizar aplicações desenvolvidas e publicadas neste mesmo software. TIBCO inclui aplicações para a coordenação de *business process* e atividades, inclui segurança na troca de informações, incluindo também outras funcionalidades relevantes.

Tibco é bastante usado devido à sua confiabilidade, flexibilidade e escalabilidade. Fornece uma estrutura comum para a integração de sistemas incompatíveis e distribuídos, o que se traduz em comunicações rápidas e fáceis, a ligação em tempo real em comparação com as tecnologias existentes no mercado atualmente.

A TIBCO possui vários softwares sendo um deles *Enterprise Application Integration* (EAI). Este permite a interação e comunicação entre as várias APIs, bases de dados e *mainframes*. Liga e transforma informações automaticamente num formato apropriado para os respetivos destinos [29]

Resumindo, esta tecnologia contém:

- *Routing* de mensagens através de regras, filtros e redireccionamento de mensagens.
- Transformação de dados.
- Transporte de dados.
- Hospedagem de Serviços.

3.3.7 WSO2

WSO2 é uma plataforma de integração de APIs, aplicações e *webservices* locais ou pela internet de alta confiabilidade, produtividade, simplicidade de uso e um retorno sobre o investimento elevado quando comparado com os restantes produtos disponíveis no mercado, com uma estrutura completa para desenvolver, reutilizar, executar e gerenciar as integrações.

WSO2 é um software *open-source*, possuindo diversos componentes capazes de serem usados separadamente ou em conjunto como uma plataforma coesa e ágil.

Tendo como vantagens recursos híbridos, podendo ser a integração feita através das várias plataformas, WSO2 simplifica os projetos, abrangendo ambientes tradicionais e de micro serviços. Difere-se de outros softwares pois, e como já referido, sendo um software *open-source* permite que cada desenvolvedor consiga adaptar e costumizar o software às suas conveniências [30].

3.3.7.1 WSO2 API Management

WSO2 API Management permite a publicação de serviços no modelo de *cloud*, permitindo assim que clientes possam aceder e consumir APIs desenvolvidas de forma segura e controlável. Esta gestão é feita pelas ferramentas *WSO2 API Publisher* e *WSO2 API Store*. Sendo estas duas os componentes centrais para implantar e gerir ecossistemas orientados por APIs [31].

3.3.7.2 WSO2 Integrator

É um modulo de integração alimentado por um *Enterprise Servie Bus* (ESB) desenvolvido em Java, sendo leve e orientado a componentes. Permite a conexão e reutilização de sistemas e serviços de forma heterogénea.

WSO2 Integrator torna possível orquestrar toda a comunicação entre diversos *WebServices*, Microserviços, bases de dados e muito mais [32].

Com isto, podemos afirmar que trabalha com os seguintes processos:

- Mediação de serviços, permitindo a separação das responsabilidades conforme o design da lógica do negócio a ser implementado.
- *Routing* de mensagens através de regras, filtros e redireccionamento de mensagens.
- Transformação de dados.
- Transporte de dados.
- Hospedagem de Serviços.

3.3.8 Oracle API Management

Oracle API Management fornece flexibilidade no carregamento de dados, tem diversos conectores para comunicar com APIs externas, mas existe pouca documentação [33].

3.3.9 IBM API Connect

IBM API Connect deixa os utilizadores criar, gerir, aplicar regras de segurança e socializar APIs. Disponibiliza vários portais destinados todos aos programadores tendo cada um a sua função.

Um dos portais é portal do desenvolvedor, destinado aos programadores. Este serve para que os programadores possam desenvolver as suas APIs podendo também para ver as que foram publicadas.

Outro portal é o de administração. Este permite aos programadores estabelecer políticas de uso para as suas APIs, como por exemplo, registo de utilizador, máximo de pedidos, entre outros.

Esta empresa disponibiliza, para além dos portais, outros serviços de interesse para os programadores. Um destes serviços é a possibilidade de cada desenvolvedor poder analisar o tráfego das suas APIs.

Existe também um serviço chamado de *Cloud Manager*, sendo que os portais podem ser controlados através deste para completar certas tarefas como atualizar e reiniciar servidores, monitorizar a *API Connect* (uso de memória, uso de processador, etc.), esta plataforma é configurada com *servers, cluters, gateways, user repositories*, etc. [34].

3.4 Considerações Finais

Terminando assim a analise das tenologias existentes no mercado, e já tendo uma noção do que é integração e dos objetivos pretendidos para a tecnologia, é tempo de passar à decisão de qual a tecnologia eleita. Devido à existência de um número elevado de APIs possíveis, a sua escolha passará por uma redução do leque de escolha e uma análise mais aprofundada de cada uma das escolhidas.

É de notar que as tecnologias *Sensedia, Red Hat e Perfoce Software* não foram consideradas nesta análise devido ao número de tecnologias já abordadas serem suficientes para chegar a uma conclusão de qual usar.

Capítulo 4

Análise das Tecnologias de integração

Após a identificação e apresentação de várias tecnologias existentes, até ao momento, que podem contribuir para a resolução do problema descrito no capítulo 1, foi ficando claro que, e tendo sempre em atenção a maneira como o *ePatient* foi desenvolvido e a sua fácil e eficaz integração sem que não ocorram erros graves ou perda de informação, a escolha recaía sobre uma das tecnologias de integração anteriormente apresentadas.

De modo a poder chegar à tecnologia eleita para o desenvolvimento do trabalho, foram consideradas três das tecnologias de integração apresentadas:

- Mulesoft – a razão da seleção destas recai no facto de a empresa Deloitte ser parceira da empresa, o que é vantajoso a nível financeiro;

- TIBCO – apesar de ser solução um pouco mais cara, quando comparada com as restantes tecnologias, a seleção desta foi devido às suas opções e funcionalidades, tendo sido também recomendada a sua utilização;

- WSO2 – a seleção desta tecnologia foi pela parte da empresa, dado que este pretende apostar nesta tecnologia, tendo a vantagem de existir um conhecimento prévio sobre a mesma por parte do candidato.

4.1 WSO2

Em 2019, *Gartner* atualizou o seu "*Magic Quadrant*" e colocou WSO2 posicionado no Quadrante de Visionários, o que reconhece o esforço que este software tem feito para que consiga aproximar-se dos líderes, WSO2 tem um forte leque de produtos que está agora a ganhar reconhecimento do mercado [30].

4.1.1 WSO2 ESB

O WSO2 ESB é um software leve e de alta performance com quase nenhuma latência, tem diversas configurações que o tornam adaptável a quase todos os problemas de quem usa, assegura que todas as conexões são feitas de forma segura e fiável, de forma a haver uma alta coesão e baixo acoplamento, tem à disponibilidade do desenvolvedor, diversos conectores que permitem a conexão a vários serviços externos, tem também uma ferramenta de *debug* e de monitorização para que se tenha controlo sobre tudo o que se passa na solução [35].

4.1.2 WSO2 API Manager

Wso2 API Manager é um produto de WSO2 bastante utilizado na gestão de APIs, desde a sua publicação por parte dos desenvolvedores até à subscrição por parte do cliente, pelo meio, é possível monitorizar, controlar e muito mais sobre APIs.

WSO2 lançou a primeira versão do API Manager em 2012 como uma ferramenta também ela open *source*, com a licença de Apache 2.0. Ao contrário de todos os outros produtos open *source*, não tem uma versão *enterprise*, ou seja, o mesmo produto é usado pela comunidade open *source* e nas empresas que o usam wso2. A grande diferença é que a versão *enterprise* recebe suporte, atualizações de bugs e de segurança, formação, entre outras coisas.

Estão comprimidos 8 componentes no API Manager, publisher, portal, *gateway*, portal do *admin*, *traffic* manager, *key* manager, *analitycs* e Business *process* server (BPS). Todos estes componentes implementados em Java [31].

4.1.2.1 API Publisher

O ciclo de uma API começa quando uma API é publicada. O desenvolvedor tem acesso a uma página gráfica que o ajuda nessa ação. Este produto permite que serviços de *backend* possam ter respostas guardadas em cache por um período de tempo para melhorar o desempenho da API. Desenvolvedores podem usar as subscrições já predefinidas ou definir as próprias subscrições. Políticas de tráfego, ajuda assim a controlar chamadas à API entre outras qualidades.

Em resumo, API publisher permite publicar APIs, definir regras de forma a ter uma melhor perspetiva da API publicada e da sua "saúde" [31].

4.1.2.2 API Portal

API Portal permite aos clientes subscrever testar e ler documentação de APIs, neste portal o Cliente tem acesso a todas as APIs disponíveis para o tipo de utilizador que este é. Existe também uma aba que permite criar uma comunidade sobre a API em questão, uma espécie de fórum [36].

4.2 Mulesoft

Assim como WSO2, Mulesoft é baseado em Java e é também uma plataforma de integração que ajuda a conectar bases de dados, aplicações. Tem uma plataforma chamada *Anypoint Platform* que inclui diversas ferramentas para desenvolver, gerir e testar aplicações [37].

4.2.1 Componentes de Mulesoft

Pode-se então enumerar algumas ferramentas que fazem parte da plataforma [37].

- API Designer: Uma ferramenta gráfica que o desenvolvedor pode usar para contruir e documentar uma API, assim como partilhar o seu trabalho com colegas de equipa. Para alem disso, um desenvolvedor pode escolher por usar componentes já desenvolvidos de uma API.
- APU Manager: é uma interface para o desenvolvedor gerir APIs, assim como implementar medidas de segurança, com a utilização de um API Gateway. Com este componente é possível controlar o acesso às APIs e criar políticas de acesso à API.
- *Anypoint Studio*: uma plataforma gráfica, que os desenvolvedores usam para contruir e configurar APIs e os seus processos, contruindo assim o ESB dessas APIs.
- *Anypoint Connectors*: são um conjunto de conectores que um desenvolvedor pode usar para integrar aplicações com aplicações terceiras.
- *Anypoint Analytics*: é uma ferramenta de analise de uma API, com desempenho e uso. Um desenvolvedor pode usar esta ferramenta para contruir gráficos personalizados e como também identificar a causa de algum problema que possa estar a acontecer.
- *Anypoint Runtime* Manager: é uma consola que o utilizador pode usar para supervisionar e monitorizar todos os recursos alojados no *Anypoint Platform*.
- *Anypoint Exchange*: é uma espécie de mercado que o desenvolvedor pode usar para guardar e acessar APIs, conectors, documentação entre outros recursos.
- *Anypoint Monitoring*: É uma *dashboard* que ajuda o desenvolvedor a monitorizar a aplicação

4.2.2 Mulesoft ESB

4.2.2.1 Arquitetura Mulesoft

O ESB de Mulesoft funciona como um agente que redireciona mensagens. Este recebe mensagens de um determinado sítio, faz alguma logica de integração (validação, transformação, …) e depois envia a mensagem para o seu destino. Mensagens são uma espécie de pacotes que são transferidos de um sítio para outro num canal específico [38].

4.2.2.2 Formato das Mensagens

Ao contrário de outros ESBs, Mulesoft apenas faz a conversão entre a mensagem recebida e a mensagem enviada apenas de for necessário, isto ajuda a acelerar o processo de roteamento de mensagens [10].

4.3 TIBCO

Tibco tem também dois sistemas que podem ser usados para o desenvolvimento da *framework*, começando pelo de API Management, designado por TIBCO *Cloud™ Mashery® API Management Platform*, que tem como capacidades [39]:

- Criar APIs através de qualquer tipo de *data source*;
- Alterar o acesso a APIs;
- Aplicar protocolos de segurança;

O segundo sistema que é responsável pela criação do ESB, designado TIBCO *BusinessWorks*™ Software. Este é um pouco diferente dos restantes *softwares* sobre os quais as avaliações foram feitas.

Tibco BW é um produto baseado em Java de fácil compreensão pois o mesmo parece um fluxo de trabalho. A lógica de negócios geralmente é implementada na forma de fluxo de trabalho, daí o título da aplicação.

Tibco BW tem diversos recursos, o que faz dele um dos mais conceituados *middlewares* existentes no mercado. Este contém um grande poder de integração, conseguindo integrar qualquer sistema abstrato, possui uma interface gráfica para definir processos de trabalho e além disso contém nele próprio o servidor para executar esses mesmo processos.

Em Tibco BW toda a lógica é mais simples, relativa aos seus concorrentes diretos, devido ao facto de existir uma abstração de código e todos os processos são configurados usando ligações e funções já pré-definidas. Em relação às comunicações, Tibco BW usa uma solução não tanto utilizada pelo restante mercado que é o uso de mensagens em *queues* e *topics* [40].

4.4 Análise Comparativa

Depois de uma análise técnica mais detalhada das três tecnologias foi então decidido que se irá considerar as *reviews* apresentadas no site *Gartner* e no relatório da *Forrester-wave*. Estas são empresas direcionadas à pesquisa das tecnologias.

4.4.1 Indicadores Subjetivos

Neste ponto serão utilizados indicadores subjetivos. Subjetivos, pois, advêm da opinião de utilizadores experientes aquando a utilização das tecnologias a avaliar. Esta avaliação foi possível através da empresa *Gartner*.

Esta é uma empresa líder em pesquisa e consultoria. Fazem pesquisas das tecnologias para fornecer os melhores concelhos às empresas. Existe também uma parte em que desenvolvedores podem dar a sua opinião sobre certas tecnologias e escrever as suas opiniões, sendo esta parte a ser utilizada inicialmente para a comparação das três tecnologias escolhidas. Estas opiniões são dadas consoante a experiência dos desenvolvedores nas mesmas, o que faz deste site um site bastante fiável para a análise que está a ser feita.

Na sua plataforma são apresentados vários questionários à cerca de diversas características das tecnologias. Estes questionários serão impulsionadores na escolha da tecnologia a utilizar. São tidas em considerações algumas características no que toca à granularidade alta do sistema. Características essas como:

- Contrato;
- Capacidades de integração;
- Facilidade de integração;
- Suporte.

O primeiro questionário a apresentar é o de WSO2, representado na Figura 15. Neste gráfico temos a relação entre a avaliação dada pelos utilizadores experientes com a tecnologia e as características a considerar. Esta avaliação é uma avaliação quantitativa, de 0 a 5, sendo 5 a pontuação correspondente à excelência daquela característica.

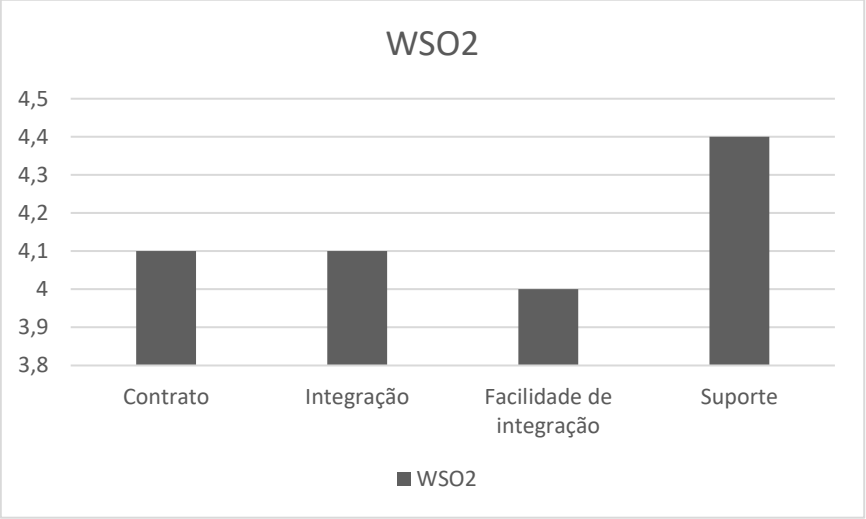

Figura 15 – Questionário WSO2

Observando o gráfico relativo a WSO2 podemos verificar que o destaque vai para o suporte, uma característica importante e com uma certa relevância dado que, apesar o nível de experiência de que desenvolve nesta, é sempre importante ter a segurança que existe ajuda no caso de surgir um problema. Em relação às outras características, e apesar de apresentarem barras mais baixas, é de notar que apresentam avaliações elevadas.

O segundo questionário a apresentar é o de Mulesoft, representado na Figura 16. Neste gráfico temos a relação entre a avaliação dada pelos utilizadores experientes com a tecnologia e as características consideraras também no questionário anterior. Neste gráfico também a avaliação é quantitativa, com as mesmo regras do gráfico acima representado.

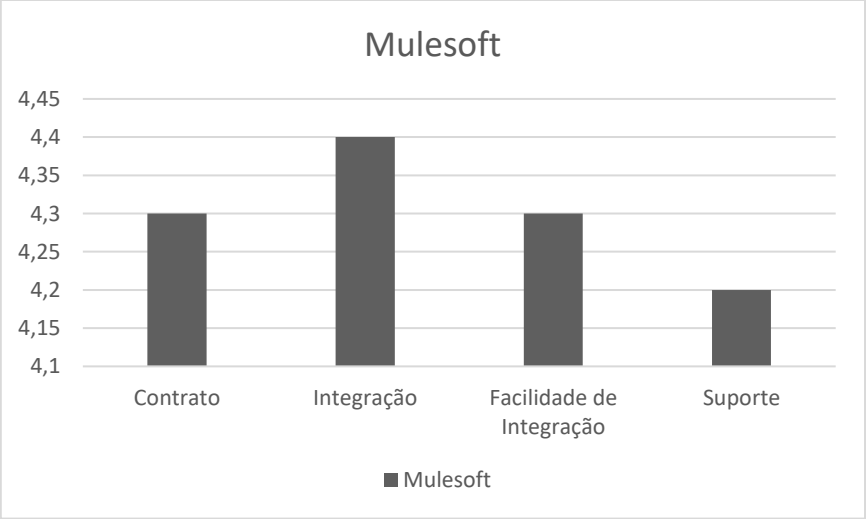

Figura 16 – Questionário Mulesoft

Observando agora o gráfico relativo a Mulesoft, é possível verificar que o forte de WSO2 é, neste caso, uma das principais fraquezas. Esta tecnologia apresenta uma elevada capacidade de integração, sendo o seu forte, devido às suas funcionalidades dificilmente igualadas pelos

restantes competidores. Apesar de se verificar diferentes avaliações a cada característica é possível verificar que nenhuma foi avaliada abaixo de 4.

O terceiro, e final, questionário apresentado é o de TIBCO, representado na Figura 17. Neste gráfico temos a relação entre a avaliação dada pelos utilizadores experientes com a tecnologia e as características consideraras também nos dois questionários anteriores. Neste gráfico também a avaliação é quantitativa, com as mesmo regras dos gráficos acima representados.

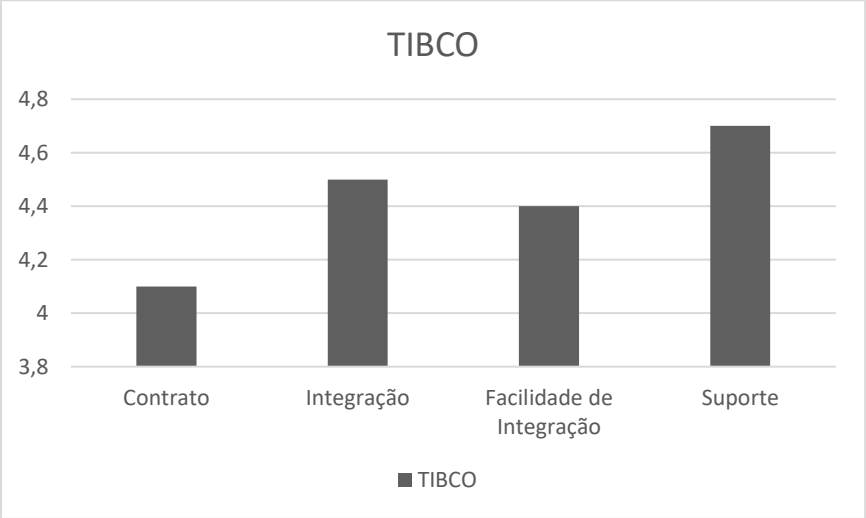

Figura 17 – Questionário TIBCO

Observando por fim o gráfico relativo a TIBCO, é possível verificar que, do três apresentado, é a tecnologia que apresenta melhores resultados na maioria das características a avaliar, tendo apenas uma avaliação inferior a 4,4 no que toca ao contrato. A característica melhor avaliada é o suporte, como em WSO2 apesar de uma diferença de 0,3. É de mencionar que o suporte fornecido a esta tecnologia é uma subscrição à parte dos restantes produtos [41].

4.4.2 Indicadores Objetivos

Por sua vez, neste ponto serão utilizados indicadores objetivos. Isto é, é realizada uma avaliação técnica, objetiva e igual para todas as tecnologias. Esta avaliação foi possível através da empresa *Forrester wave*.

Esta empresa fornece um guia para apresentar e ajudar a quem tenciona comprar ou mudar de tecnologia. É uma avaliação mais objetiva dado que, esta empresa, para oferecer um processo equitativo a todos, segue uma metodologia disponibilizada publicamente [42]. Num relatório apresentado por esta empresa, relativo à gestão de APIs [43], existe referencia a estes três tecnologias, fazendo assim uma análise mais objetiva, concreta e equitativa que se deve ter em conta, sendo apresentado nos pontos 4.4.2.1, 4.4.2.2 e 4.4.2.3 referencias a esse mesmo relatório sobre as tecnologias escolhidas.

4.4.2.1 TIBCO

Um antigo *player* no mercado, a TIBCO agora faz parte de uma ampla linha de produtos para integração e gestão de APIs.

Agora a TIBCO está a voltar-se para investimentos mais profundos na própria *Mashery*, particularmente na criação de uma nova e mais flexível integração com várias tecnologias de serviço e micros serviços. Com base em suas fortes parcerias com *Azure* e *Amazon Web Services* (AWS), o produto já tem uma integração AWS interessante que torna mais fácil para os clientes venderem as suas APIs por meio do mercado da AWS.

Na avaliação feita no estudo, a solução da TIBCO estava em pé de igualdade com outras em toda a linha, com pontos fracos na criação de políticas e micros serviços (que irão melhorar assim que as melhorias pendentes forem completadas). A visão futura da TIBCO para a gestão de APIs centra-se na gestão de produtos de API, ecossistemas estendidos e passando para além das APIs REST. Os planos de aperfeiçoamento de produtos concretos da empresa vão além do portal e micros serviços para incluir mais envolvimento do utilizador de APIs, melhor controlo de APIs e gestão de portfólio, melhor documentação de API, mais suporte de CI / CD, gestão de produto de API, proteção contra-ataques mais profunda e suporte *multicloud* adicional. Os clientes de referência da TIBCO expressaram satisfação relativamente alta com o fornecedor e o produto. A solução é especialmente boa para compradores que procuram um parceiro estratégico para gestão de APIs, juntamente com vários estilos de integração (por exemplo, dados, aplicativos e processos) e as tecnologias analíticas da TIBCO agregam mais valor. [44]

4.4.2.2 WSO2

Para a sua reputação, WSO2 continua a projetar uma presença de gestão de API maior do que seu tamanho, tanto em termos de funcionalidade da solução quanto de participação no mercado. A empresa direciona os seus investimentos em produtos estratégicos de forma a fornecer funções-chave e flexibilidade para os clientes. A natureza totalmente de código aberto do produto adiciona uma camada adicional de flexibilidade, caso os clientes precisem, além de oferecer um caminho gratuito para iniciar o gerenciamento de API por meio de auto-suporte de código aberto. Recursos notáveis incluem suporte de publicação *GraphQL*, integração profunda da plataforma de micros serviço, bom pacote de produtos de API e recursos de integração de aplicativos completos no WSO2's *Enterprise* Produto integrador. Os modelos comerciais da solução contam com CPUs ou volumes de chamadas de API dependendo do modelo de implantação.

Em avaliação, o WSO2 demonstrou muita força na maioria das categorias de avaliação, e nas restantes igualou-se. Tem, especialmente, um portal de utilizador de API flexível e configurável, um forte começo indo além das APIs REST, bons investimentos em estratégia combinada de micros serviços de API (incluindo liderança de pensamento importante sobre como os dois se relacionam) e opções de implantação e arquitetura de produto fortes. As principais prioridades

de produtos futuros incluem design de API REST, mais investimento em micros serviços e APIs não REST, integração de serviço mais profunda, aperfeiçoamentos de integração / entrega contínua (CI / CD) e ecossistemas de API e gestão de produtos. A solução é particularmente adequada para compradores com estratégias que combinam integração, APIs e micros serviços com o desejo de obter os benefícios do código aberto. [44]

4.4.2.3 Mulesoft

Quando a *Salesforce* comprou a *MuleSoft*, alguns questionaram o que é que a *Salesforce* receberia. A melhor pergunta a ser feita é o que a *MuleSoft* receberia. Resposta: uma rápida expansão dos clientes em potencial e dos recursos de aliciamento dos utilizadores da API. O seu *API Community Manager* utiliza ativos do *Salesforce* para enriquecer a funcionalidade do portal, especificamente para envolver o público de utilizadores de API. Embora a maior parte da solução, incluindo o portal *Anypoint* Exchange tradicional, esteja disponível para implantação gerida pelo cliente, os novos recursos da comunidade são apenas SaaS. As melhorias arquitetónicas desde a última avaliação permitiram que a *MuleSoft* oferecesse um modelo comercial notavelmente mais competitivo. O seu componente *Anypoint Runtime Fabric* permite que o plano de controle baseado em SaaS da *MuleSoft* conduza recursos de tempo de execução geridos pelo cliente (sejam baseados em nuvem ou no local).

Na avaliação, os pontos fortes da *MuleSoft* foram o envolvimento do utilizador da API, design e proxy da API REST, gestão da entrega e análises. As restantes áreas estavam no mesmo nível dos seus concorrentes. Seu suporte a micros serviços por meio do *Anypoint Service Mesh* é um bom começo em APIs e micros serviços combinados. A visão da *MuleSoft*, apoiada pela sua equipe de Estratégia de API *Global*, inclui agilidade de negócios, micros serviços, APIs não REST e *DevOps*, com planos de produtos específicos nessas áreas, bem como proteção contra-ataques de API, ecossistemas e infusão de inteligência no processamento de API. Os clientes deram notas altas ao fornecedor e ao produto. A sua solução é boa para compradores que desejam integração combinada e recursos de gerenciamento de API, que a orientação da sua equipa de estratégia de API pode complementar [44].

4.4.3 Análise Final

Após a análise gráfica (Figura 15, Figura 16, Figura 17) ficou claro que as três tecnologias estão muito equiparadas. Com a análise dos gráficos e a apresentada no relatório da *Forrester Wave* podemos então tirar algumas conclusões.

A primeira conclusão a tirar, em termos de tecnologia, é que a Mulesoft apresenta dados em como se encontra à frente, das outras avaliadas, mostrando ser um software mais capaz. A segunda a tirar, em termos de suporte, é que, apesar de Mulesoft ser tecnologicamente melhor,

o suporte dado por parte da empresa WSO2 é ligeiramente melhor, já tendo sido referido como uma característica importante no desenvolvimento de qualquer software. Por último, em termos financeiros, TIBCO é um sistema muito capaz, mas tem o problema de ser uma solução com custos elevados. Apesar de possuir um suporte bastante bom, é também bastante dispendioso.

É de notar que esta diferença entre as várias tecnologias não é de uma discrepância considerada enorme, como é possível ver pela escala do gráfico da *Gartner*, onde não existem avaliações abaixo de 4 numa escala de 0 a 5, e na análise da *Forrester Wave*, em termos de funcionalidades.

Em resumo, e como dito anteriormente, as três tecnologias estão muito equiparadas, fazendo em geral um trabalho muito idêntico. Com isto, é necessário ponderar os prós e contras de cada uma das tecnologias, tanto para o projeto como para a empresa.

Pros de Mulesoft:

- Usa RAML como editor de APIs, é menos complexo e mais fácil de entender;
- Mule ESB tem diversos conectores para aplicações terceiras. Isto permite ao ESB integrar com outros sistemas mais rapidamente;
- Mule API Gateway suporta monitorização e gestão bastante bem, permite a utilização de políticas de restrição por exemplo, entre outros serviços como *throttling*;
- Suporta o uso de HL7.

Contras de Mulesoft:

- Sem o suporte por parte da empresa Mulesoft torna-se difícil aprender a dominar a ferramenta;
- Diversas configurações fazem com que se perca facilmente o objetivo, é recomendado o uso de um *roadmap*;

Prós de WSO2:

- Um dos requisitos básicos do ESB é que suporte transformações e WSO2 ESB dá suporte a XSLT, sendo assim possível transformar todo o tipo de formato;
- WSO2 ESB é uma excelente ferramenta para tratamento de erros, com personalização de mensagens dependendo do erro para enviar ao cliente.
- Suporta o uso de HL7;

Contras de WSO2:

- Embora seja fácil de configurar para uma iniciação rápida, é, no entanto, difícil de configurar para cenários complexos de integração;
- Não e muito estável em produção, tendo diversos erros triviais;

- Não tem uma ferramenta de se reiniciar automaticamente caso ocorra algum problema;

Pros de TIBCO:

- Desenvolvimento rápido;
- Alojamento rápido;
- Ferramentas de monitorização bastante boas quando compradas com a concorrência;
- Elevada possibilidade de escalabilidade.

Contras de TIBCO:

- Suporte pago;
- *User* interface pouco configurável nos portais;
- Restrições no desenvolvimento de APIs, como por exemplo limite de configurações aplicadas no desenvolvimento de APIS;

4.4.4 Decisão Tomada

Tendo todos os pontos em consideração referentes às capacidades de cada tecnologia, avaliações realizadas por empresas externas e prós e contras de cada uma, e tendo também em consideração o projeto a realizar, a decisão recaí sobre o uso de WSO2 como sendo a tecnologia indicada, divido a diversos fatores.

Um dos primeiros fatores a ter em consideração é o preço. WSO2 é uma solução bastante mais económica, sendo que a diferença de valor entre as restantes duas tecnologias não ser significante.

Outro dos fatores a ter em consideração é a empresa. Como referido no início deste capítulo, no ponto 0, era desejo de a empresa iniciar o desenvolvimento de projetos de integração com a tecnologia WSO2, tornando-se assim um dos projetos pioneiros na tecnologia da empresa.

Capítulo 5

Desenho da Solução

Analisada e escolhida a tecnologia a utilizar, prossegue-se com q etapa do design da solução. Com isto, é necessário ter em atenção certos pontos. Pontos esses como os *standards* de comunicação, as questões de segurança, a monitorização, e por fim, mas não menos importante, o poder de escalabilidade do sistema, dado que neste tipo de *softwares* é muito comum a existência de vários sistemas que se pretende integrar.

5.1 Perspetiva geral

Depois de encontrados e apresentados todos os problemas existentes na solução atualmente implementada, é então necessário começar a resolvê-los. Para isto, e para que seja possível uma melhor resposta a estes pontos, foi necessário um desenvolvimento para além daqueles que fazem parte dos requisitos para este trabalho.

Inicialmente é necessária a implementação de uma camada externa à aplicação capaz de lidar com todos os pedidos que chegassem a esta. Para isto, foi criada então a camada de integração. Esta camada de integração tem de ser capaz de receber pedidos externos ao *ePatient*, efetuar

as diversas verificações necessárias para concluir o pedido e, ao mesmo tempo, fazer uma monitorização dos pedidos que chegam a este mesmo sistema.

Na Figura 18 é possível verificar esta nova camada de integração, representada pelo componente WSO2 Suite. Com isto, todas as interações efetuadas pelo *ePatient* são feitas e tratadas através do WSO2 Suite, contrapondo a arquitetura original onde o *ePatient* comunicava diretamente com serviços externos.

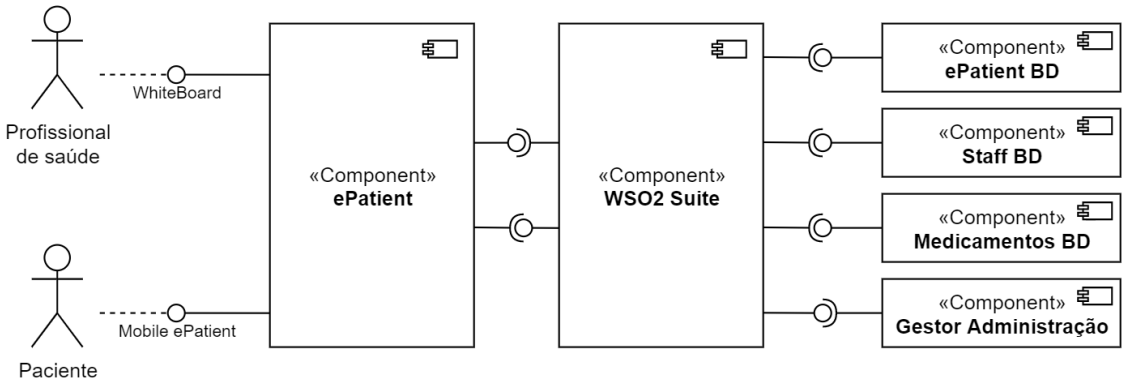

Figura 18 – Arquitetura com camada de integração

Com esta solução, o WSO2 Suite fica então responsável por:

- Gestão de todas as comunicações entre *ePatient* e sistemas externos;
- Gestão de todas as validações necessárias;
- Controlo sobre acessos aos serviços de *ePatient*;
- Monotorização dos pedidos;

O componente *ePatient* deixa de ter conhecimento e de interagir diretamente com os vários sistemas externos, passando a interagir apenas, e só, com o *WSO2 Suite*. Esta interação é realizada através de uma única API, sendo definida independentemente do número e características dos sistemas externos a integrar.

Assim é possível concluir que a *Mobile ePatient, WhiteBoard e ePatient DB* ficam inalterados e, portanto, não serão novamente abordados na descrição da solução proposta.

Posteriormente, nas próximas secções os componentes *ePatient* e *WSO2 Suite* serão abordados mais ao detalhe.

5.2 ePatient

Em termo de *ePatient* existe uma diferença bastante relevante, entre a arquitetura inicial, Figura 2, e a nova arquitetura apresentada, Figura 19. A base de dados referente ao *ePatient*

passa a estar fora do componente *ePatient* e todas as comunicações entre o *ePatient* e a sua base de dados é feito através do WSO2.

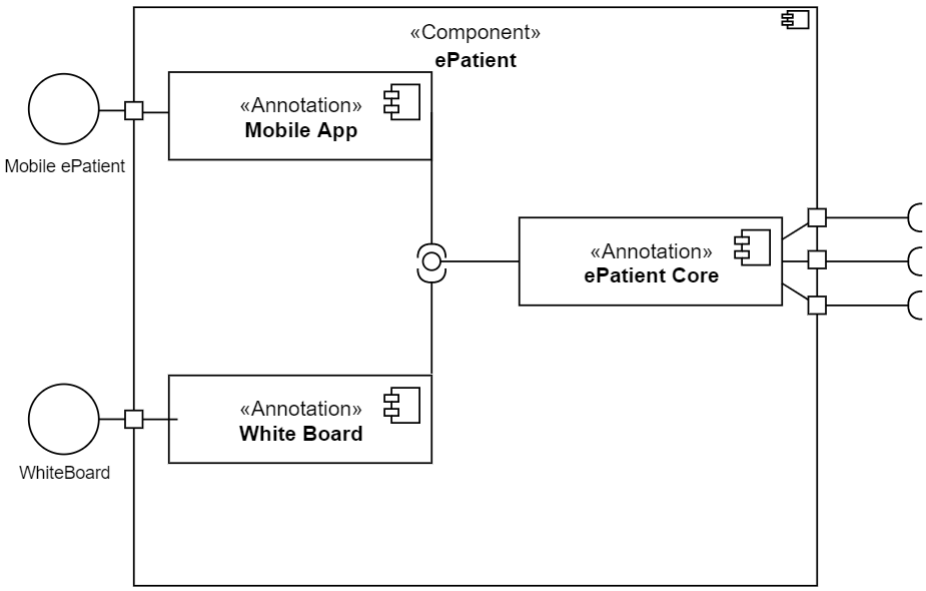

Figura 19 – Representação detalhada nova solução ePatient

5.2.1 ePatient Core

Estando, neste presente capítulo, a efetuar um exercício de comparação e evolução, a seguinte Figura 20 demonstra a consequência natural da arquitetura, relativa à inicialmente existente, representada na Figura 5.

É então possível verificar, na Figura 20, que já não existe qualquer componente de *External Integration* pertencente ao *ePatient Core*, este passa a ser desnecessário pois as suas responsabilidades passam a ser desempenhadas pelo componente WSO2 Suite. O componente WSO2 tem agora a responsabilidade de comunicar com serviços externos (representados na imagem por AP1, AP2 e APN).

Assim existe uma transparência/simplicidade para cada lógica de negócio (Core) que está abstraído de onde vem a informação solicitada/requerida para o processo de negócio em causa.

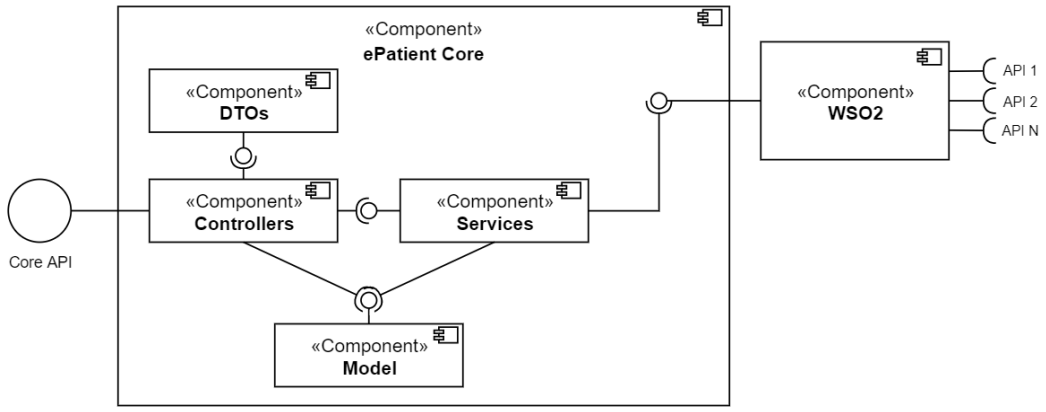

Figura 20 – Nova Arquitetura

5.2.2 Considerações finais

Neste ponto já existem alguns pontos positivos a retirar da nova solução apresentada, como por exemplo o retirar de lógica contida no componente *ePatient Core* que faz com que haja uma baixa acoplação de serviços num só componente.

A existência de um componente intermediário faz com que seja possível haver um certo controlo nas chamadas, assim como uma abstração do tipo de sistemas que comunicam entre si.

5.3 Especificações no WSO2

Em relação ao WSO2, como referido no ponto acima, este é responsável por comunicar com o mundo exterior, sendo um dos componentes mais importantes neste desenvolvimento, devido às funções que exerce. Quando se fala no WSO2 deve referir-se e especificar quais as configurações necessárias para que este consiga exercer as suas funções. São configurações que exigem a maior das atenções dados que esta camada de integração tem de ser capaz de comunicar e transmitir de forma correta, sem que ocorra qualquer falha, todos os dados necessários.

Com isto, e dada a comunicação entre a aplicação e vários sistemas externos, é essencial contruir uma solução que consiga lidar com sobrecarga de serviços. Para que isto seja possível, é então necessário dividir os pedidos em certos módulos. Por observação, tanto das integrações existentes como das previsíveis de ocorrer, foram identificadas diferentes responsabilidades que podem ser agrupadas em módulos ou área de atuação:

- Infraestruturas - pedidos referentes a quartos do hospital;
- Cirurgias - pedidos referentes a cirurgias;

- Profissionais de saúde - pedidos referentes a profissionais de saúde;
- Camas de hospital - pedidos referentes a camas de hospital;
- Exames - pedidos referentes a exames;
- Registo de atividade - pedidos referentes a atividades efetuadas pelo paciente;

Identificados os pontos de análise às chamadas necessárias da API, o próximo passo é naturalmente o desenho de arquitetura, tendo sempre em atenção toda a análise.

Para isto foi necessário adaptar todos os módulos encontrados, tendo sido usadas várias APIs para a sua representação. Esta divisão causou impacto no WSO2 IE. A divisão por APIs criou a necessidade de as desenvolver, contendo cada uma os diversos *resources*.

Para a adaptação dos módulos é então necessário a utilização de duas camadas de APIs. São aplicadas as camadas de *Process* e *System*.

A camada *Process* corresponde à camada intermedia. É nesta camada onde a lógica das diversas verificações e das chamadas secundárias, que sejam necessárias efetuar, é aplicada.

A camada *System* funciona como uma camada de adaptação, isto é, em terminologia WSO2 corresponde aos *endpoints* de saída. Esta camada é responsável pela lógica de integração e pela ligação à base de dados. É esta base de dados que alimenta a plataforma *ePatient*. Estas duas camadas, e as suas ligações, encontram-se representadas na Figura 21.

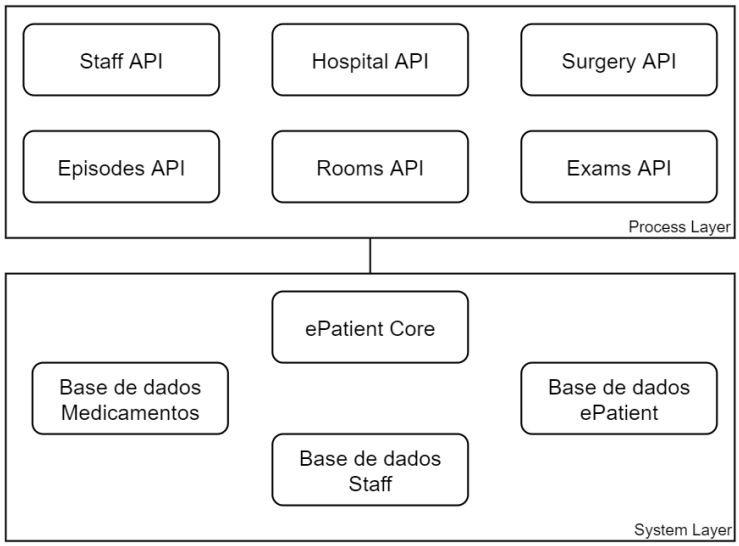

Figura 21 – Representação da comunicação entre APIs

Na camada *Process*, camada inferior, estão representadas as mais variadas APIs, com as suas diversas responsabilidades:

- *Hospital* API – responsável por conter os tipos de serviços e especialidades que o hospital oferece aos pacientes;

- *Surgery* API – responsável por toda a parte referente a uma cirurgia, como o médico associado e os tipos de cirurgia existentes no hospital;
- *Staff* API – responsável por gerir o Staff do hospital, como médicos, enfermeiros e respetivas especializações dos enfermeiros;
- *Rooms* API – responsável pela gestão de camas e quartos do hospital;
- *Exams* API – responsável por todos os tipos de tratamentos e exames que o paciente poderá fazer, assim como também o registo dos mesmos.
- *Episode* API – responsável pelo registo de todos os tipos de acontecimentos que um paciente possa ter durante o seu internamento no hospital.

Na camada *System* estão representados todos os recursos aos quais o WSO2 tem acesso para comunicar e efetuar a troca de informação.

5.3.1 WSO2 API Manager

WSO2 *API Manager* é responsável por expor as diversas APIs que foram mencionadas no ponto anterior 5.3. Além de conseguir expor as APIs, este componente é também responsável por monitorizar, aplicar políticas de segurança, caso existam no futuro, e por analisar as estatísticas de acesso e erros.

5.3.2 API Login

Por questões de segurança, a aplicação *ePatient* utiliza políticas de segurança, essa política de segurança é aplicada através da utilização de *Bearer Authentication* [45].

Bearer Authentication é uma autenticação HTTP que envolve *tokens* de segurança designados por *Bearer Tokens. Bearer Tokens* permite que os pedidos REST utilizem autenticação através de uma chave de acesso, como é o exemplo do *JSON WEB TOKEN* (JWT).

Neste caso, o *token* não é nada mais do que uma *string* incluída no *header* do pedido. Este *token* não tendo validade vitalícia, sendo necessário renovar o mesmo. A validade definida dos *tokens* é definia pelos desenvolvedores.

Nesse sentido, as APIs expostas pelo WSO2 deveram suportar este método de autenticação. Para tal, dará origem a uma API nova denominada API Login. Esta mesma API será responsável por armazenar o *token* resultante do pedido ao *ePatient* e que ficará em *cache* na API pelo mesmo tempo que a vida útil do *token* para fazer chamadas ao *ePatient*.

5.4 Realização de Casos de Uso

Neste projeto são levantados 38 casos de uso, divididos pelas APIs respetivas. Dado o número considerável de casos de uso é necessário sintetizar a descrição de cada um eles, sendo apenas apresentados e analisados alguns considerados necessários para a perceção dos mesmos, casos que diferem em certos pormenores importantes, estando os restantes representados no Anexo B.

Para que esta sintetização seja eficaz utilizou-se um modelo de diagrama de sequência que, para todos os casos de uso, ajuda na visualização de todo o processo. Este modelo pode ser visto na Figura 22.

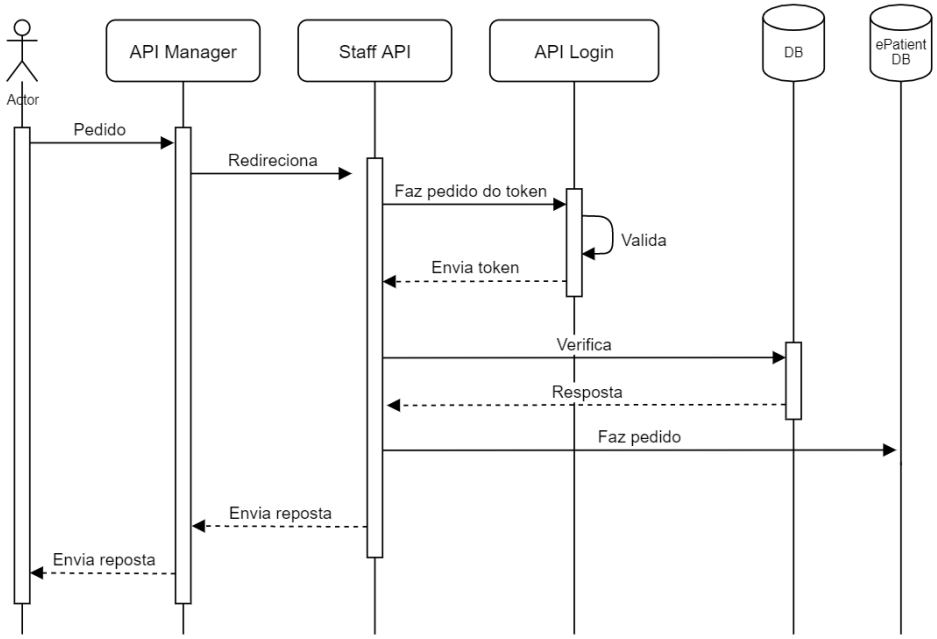

Figura 22 – Vista de cenário modelo de um caso de uso

Na Figura 22 é possível verificar que todos os pedidos comunicam sempre com WSO2, componente API Manager, não tendo qualquer relevância que tipo de autor despoleta o pedido. Depois de efetuadas as verificações no WSO2, este comunica com o componente a que se dirige o pedido.

É importante realçar que, para que isto aconteça, isto é, para que o WSO2 consiga comunicar com o *ePatient*, é necessário um *token*. Este *token* é disponibilizado pela API Login aquando efetuado o pedido.

Após o pedido ser construído é então necessário efetuar verificações à base de dados por parte do WSO2, para depois, por exemplo, verificar a existência de médicos disponíveis no caso de um novo internamento. Depois da verificação ser concluída com sucesso é então enviado o pedido ao *ePatient* para que este fique com o registo da operação e para que mais tarde seja

possível verificar essa mesma operação. Por fim é enviada uma mensagem com a informação de sucesso e a resposta, caso seja um pedido que retorne algum tipo de resposta no *payload.*

O modelo de caso de uso apresentado acima serve apenas de exemplo, para que depois, consoante cada caso de uso, possa ser adaptado ao mesmo.

Após uma rápida análise, é possível verificar que grande parte dos casos de uso são de adicionar/atualizar ou de procura. Seguindo o raciocínio apresentado no inicio deste subcapítulo foi então decidido abordar com maior detalhe "*Put Episode Open*" e "*Put Surgery*".

5.4.1 Alternativa ao Modelo de Caso de uso

Em todos os momentos, aquando do desenvolvimento de um design de uma solução, existem dúvidas relativas à aplicação das melhores práticas de engenharia de software.

Neste desenvolvimento umas das dúvidas que surgiu é relativa ao *login*. A dúvida colocada é se o *login* deve ser feito antes ou depois da chamada à API. Existem duas alternativas em resposta a esta questão.

A primeira alternativa é desenvolver consoante o diagrama apresentado na Figura 22, em que não seria uma chamada tão direta, dado que o API Manager redireciona para a API destino, sendo só aí efetuado o login necessário. A vantagem desta solução é a não exposição de dados sensíveis nas comunicações entre APIs.

A segunda alternativa, apresentada na Figura 23, é realizar uma chamada mais direta, estando o API Manager a chamar o API Login e só daí ser redirecionado para a API destino. Esta alternativa, apesar de direta, expõe informação sensível, dado que atravessa uma API a mais, para que esta adicione a restante informação necessária (*token*).

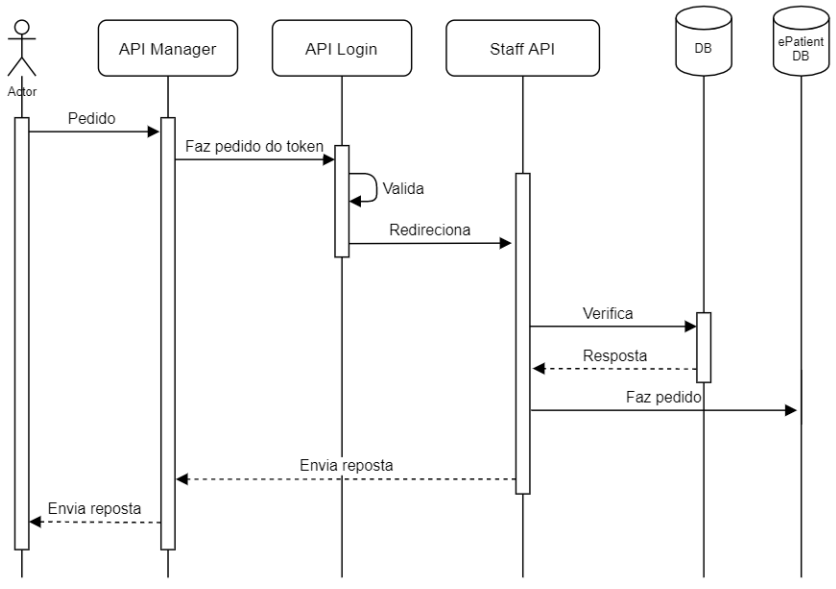

Figura 23 – Alternativa Modelo de Caso de uso

Depois de analisados os prós e contras das alternativas possíveis, ficou claro que o modelo apresentado na Figura 22 é o mais viável, pois permite que cada API tenha acesso apenas ao que a ela diz respeito. Com isto, o API Login apenas é chamado por cada API da camada *process* e devolve um *token* que depois é usado para efetuar o pedido ao *ePatient*.

5.4.2 WSO2

Depois de uma análise geral ao modelo de casos de uso, é importante realizar uma análise aos componentes representados na Figura 22 pertencentes ao WSO2. Estes componentes são o API Manager, API Login e um das APIs representadas na camada *Process Layer* na Figura 21. A título de exemplo foi usada a *Staff* API.

Como é possível verificar na Figura 24, todos os pedidos chegam à camada de WSO2 através do *ePatient*. Já no WSO2 *API Manager* é possível realizar diversas verificações e aplicar certos filtros.

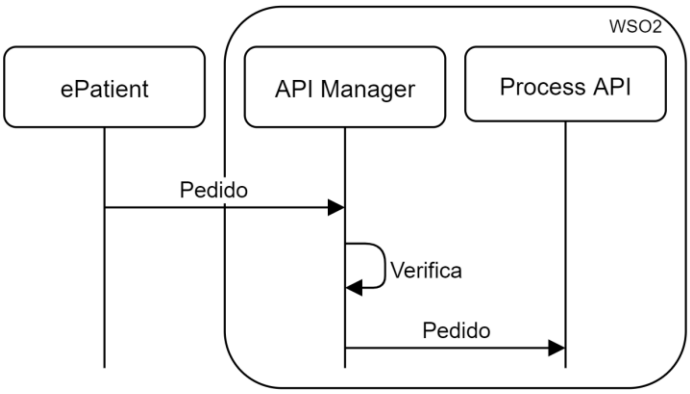

Figura 24 – Análise mais aprofundada ao WSO2

WSO2 *API Manager* é um dos componentes mais importantes a nível da camada de integração, pois é ele o responsável por toda a comunicação com o exterior. Neste ponto todas as configurações têm de estar bem definidas e todos os casos de uso associados a uma API. É importante que a ligação entre as APIs dos casos de uso a API Manager esteja bem definida.

5.4.3 Put Surgery

Como foi referido anteriormente, no subcapítulo 5.4 , existem alguns casos de uso que serão analisados mais ao pormenor. O caso de uso *Put Surgery* é um desses, dado que apresenta algumas diferenças dos restantes.

Neste caso de uso existe uma série de validações a serem realizadas antes do registo de uma nova cirurgia, como descrito na Figura 25. Estas verificações são relativas a pessoal médico, se existe pessoal médico disponível para realizar a cirurgia, e às salas de operações, se existem salas disponíveis para que a cirurgia possa ser realizada.

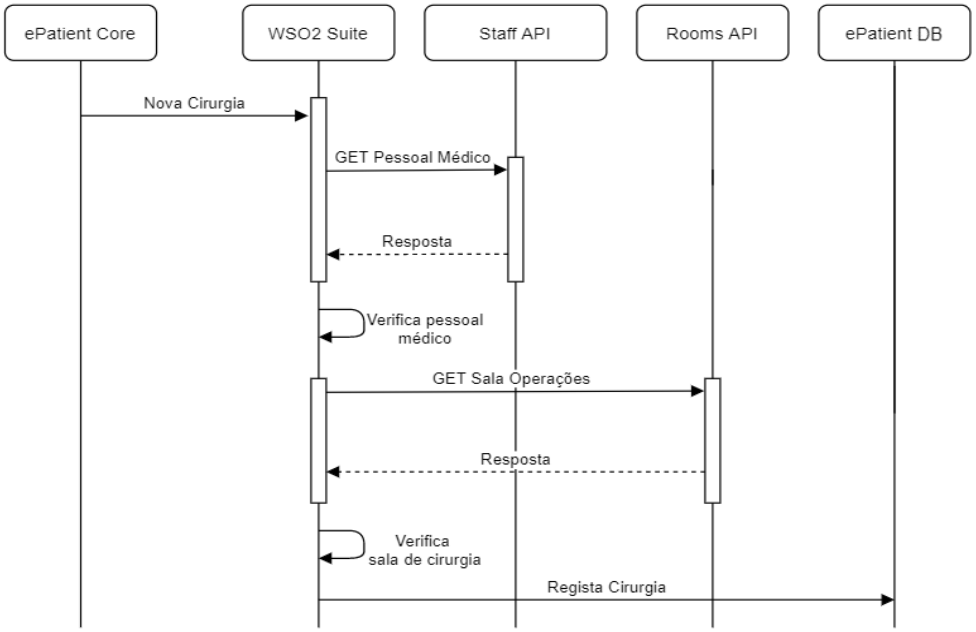

Figura 25 – Diagrama de sequência introdução de uma cirurgia

Este diagrama segue o modelo apresentado no início do capítulo 5.4, na Figura 22. Este segue o modelo de primeiro comunicar com a bases de dados, para recolher a informação necessária sobre a componente, e, de seguida, efetuar a validação dos mesmos dados no WSO2 Suite, este engloba o componente API Manager e WSO2 EI.

O diagrama representativo de todo o processo a nível de integração é um pouco diferente. Esta diferença advém do facto de ser necessário mais detalhe, para isto foi desenvolvido um diagrama EIP (sigla do inglês – *Enterprise Integration Patterns*). Para que fosse possível a implementação de uma arquitetura melhorada recorreu-se ao uso de padrões de integração. Um exemplo deste tipo de diagramas é o da Figura 26.

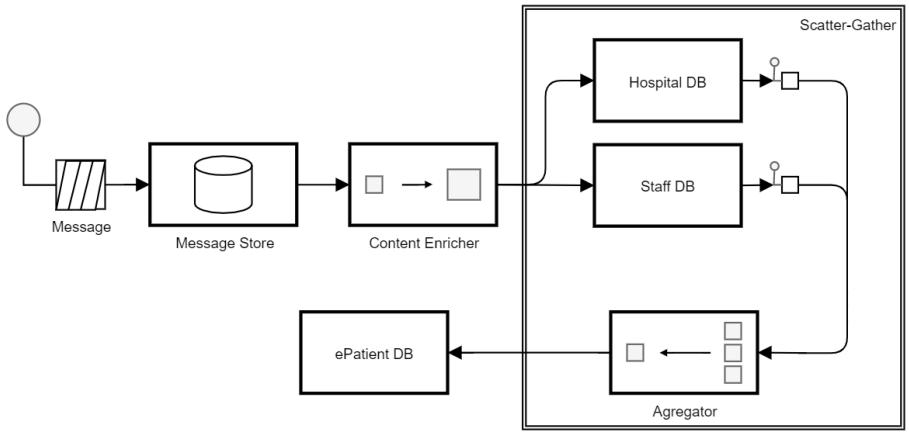

Figura 26 – Diagrama EIP nova Cirurgia

No diagrama da Figura 26, estão representados os vários processos e validações que o WSO2 necessita de fazer para depois poder comunicar com o *ePatient*. Inicialmente é necessário ir

buscar o token, este permite fazer as várias chamadas necessárias aos serviços da aplicação *ePatient*, representado na *Message Store* [46].

Depois de construir a mensagem com o *token*, utilizando o *Content Enricher*[47], é necessário fazer duas chamadas para reunir a informação toda essencial para efetuar o pedido. Estas chamadas são feitas utilizando o padrão de integração *Scatter-Gather* [48]. Este diz que o pedido deve ser divido em diversos pedidos mais pequenos, que executam todos ao mesmo tempo, para no final toda a informação ser agrupada apenas num lugar, esta responsabilidade recaí no *Aggregator* [49].

5.4.4 Put Episode Open

Como referido no ponto 5.4, existem alguns casos de uso que apresentam algumas diferenças relevantes em relação aos restantes, e que necessitam de ser apresentados mais ao detalhe.

Neste caso de uso, representado na Figura 27, existem certas validações a serem realizadas para que seja possível o registo de um novo episódio, por outras palavras, um novo paciente admitido para internamento. As validações passam por verificar se existe pessoal médico capaz de atender e responder às necessidades do paciente, se existem medicamentos disponíveis caso o paciente necessite, ou até mesmo para uma eventual emergência, e, por fim, a ocupação atual do hospital para verificar que o paciente pode efetivamente dar entrada. De uma forma geral, este é o procedimento que deve ocorrer para que seja atingido o sucesso neste caso de uso.

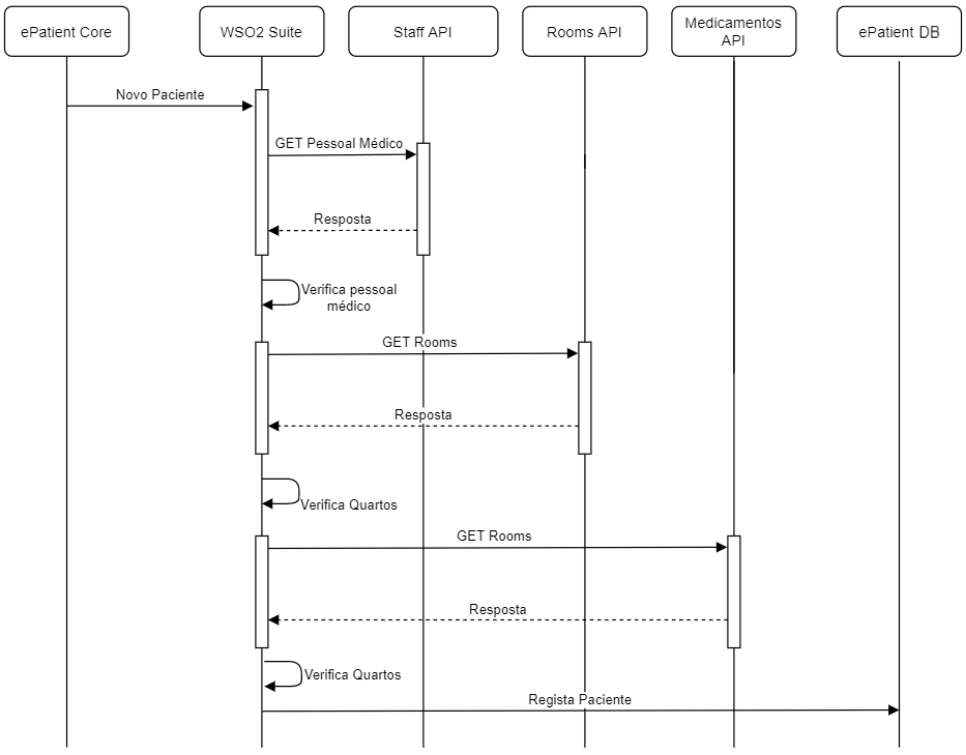

Figura 27 – Diagrama de sequência novo paciente

Neste contexto, é também possível a representação deste processo através de um diagrama IEP, representado na Figura 28, isto para que fosse possível a implementação de uma arquitetura melhorada.

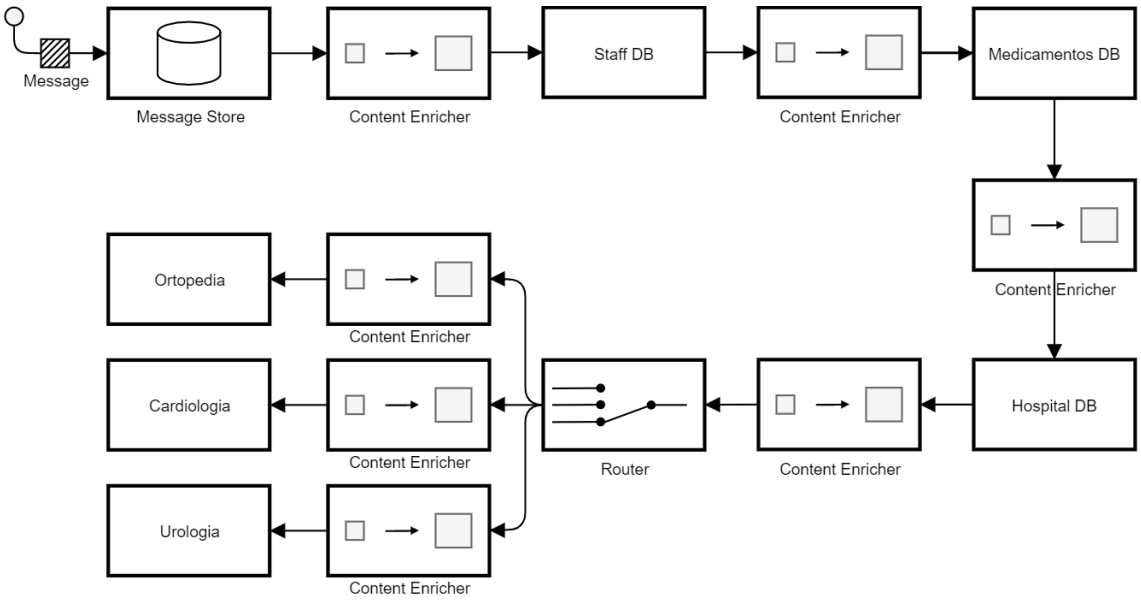

Figura 28 – Diagrama EIP novo paciente

No diagrama da Figura 28, estão representados os vários processos e validações que o WSO2 necessita de fazer para depois poder comunicar com o *ePatient*. Ao longo destas validações, o WSO2 vai construindo uma mensagem com toda a informação necessária para ser enviada para o *ePatient*, e neste caso, seja registado o novo episódio.

Inicialmente é necessário ir buscar o token, este permite fazer as várias chamadas necessárias aos serviços da aplicação *ePatient*, representado na *Message Store* [46].

Depois de construir a mensagem com o *token*, utilizando o *Content Enricher* [47]. Após a mensagem construída, é necessário aceder a pessoal médico disponível, aos medicamentos em stock e às camas livres no hospital para registar onde o paciente irá ficar internado. Durante todo este processo, e a cada acesso às bases de dados necessárias, a mensagem vai sendo construída progressivamente, através do *Content Enricher*.

Por fim, antes de ser registado o episódio, é necessário conduzir a mensagem até à especialidade introduzida na mensagem inicial. Para isto é utilizado o *Message Router* [50], este envia mensagens para todos os canais possíveis, isto é, segundo os parâmetros referenciados no *payload*. Neste caso, e para efeitos de demonstração, assume-se que o hospital apenas tem três especialidades (ortopedia, cardiologia e urologia).

5.5 Sumário

Após finalizar todo o desenho da arquitetura, e a sua descrição, é possível retirar alguns pontos positivos que a abordagem desta arquitetura traz:

- Reutilização: o facto de existirem diversas chamadas em que apenas se alteram certas variáveis, faz com que seja possível o uso de trabalho anteriormente desenvolvido;
- Produtividade: referindo aqui o ponto anterior, se existe a possibilidade de reutilizar então a produtividade aumenta, dado que não é necessário repetir os mesmos processos de desenvolvimento várias vezes;
- Flexibilidade: como existe uma camada externa dedicada exclusivamente a comunicações é possível isolar a estrutura de um serviço, fazendo com que as alterações necessárias sejam feitas com mais facilidade;
- Manutenção: como referido no ponto anterior, o baixo acoplamento, facilita a manutenção dos serviços;
- Interoperabilidade: visto ser um projeto com intenções de expandir, é certo que irão aparecer sistemas adicionais que necessitam de ser ligados, é neste ponto que a camada de integração traz as suas vantagens, sendo esta capaz de disponibilizar serviços independentemente da plataforma e tecnologia a integrar;
- Gestão: com a utilização do *API Manager*, é possível acompanhar todo o fluxo de tráfego que ocorre com as comunicações entre as APIs;
- Padronizado: o desenvolvimento é baseado no uso de padrões;
- Abstração: este ponto resume um pouco alguns dos pontos referidos anteriormente, o serviço passa a estar totalmente abstrato da sua implementação.

Capítulo 6

Implementação

O foco do presente capítulo é a exposição do processo de desenvolvimento da solução desenhada. É feita uma descrição da implementação e descritas todas as etapas da implementação e obstáculos encontrados aquando o desenvolvimento. Neste capítulo são esclarecidos alguns termos técnicos referentes a WSO2.

Como foi referido aquando a exposição dos casos de uso, no ponto 5.4, dado o elevado número de casos de uso foi decidido apresentar apenas aquele que tinham um funcionamento relativamente parecido à maioria, como também um que apresenta algumas variações no seu funcionamento. No capítulo anterior, foram apresentados, ao detalhe, os casos de uso "*Put Surgery*" e "*Put EpisodeOpen*", no ponto 5.4.3 e 5.4.4, respetivamente, sendo então estes dois a seres analisados neste ponto.

Para o desenvolvimento da solução é necessário utilizar a ferramenta de desenvolvimento de WSO2 disponibilizada para Windows, denominada de *Integrator*, esta tem por base o Eclipse. Para além do *software* referido, é também necessário instalar a versão para Windows do WSO2 *API Manager* 2.6 e WSO2 *Enterprise Integrator* 7.0.

6.1 Especificações no WSO2 ESB

Após a instalação de todos os *softwares* necessários, para o correto funcionamento do projeto, o passo a seguir é a implementação da solução.

6.1.1 API's

Começa-se pela especificação das API definidas no capítulo anterior, *Staff* API, *Hospital* API, *Surgery* API, *Room* API, *Exams* API e *Episode* API. Na Figura 29 é possível verificar as APIs definidas, como devidamente configuradas, dentro da pasta "api". Todos os *resources* necessários para atender os vários pedidos são definidos e construídos nestes mesmos ficheiros.

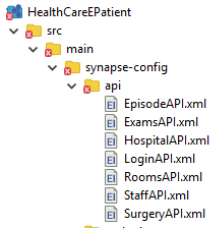

Figura 29 – Implementação das API em WSO2

Cada r*esource* é referente a um pedido específico na sua API. Isto significa que cada *resource* tem um determinado mapeamento único. Este mapeamento permite restringir que tipo de pedidos HTTP podem ser processados.

Para além das APIs, os *endpoints*, responsáveis pela comunicação com o *ePatient*, também devem ser definidos inicialmente. Esta definição deve ser inicial, e não uma codificação posterior diretamente no código, para garantir a fácil manutenção do código em questão.

Nesta solução existem inúmeras ligações ao *ePatiente*, fazendo com que seja necessário a configuração de inúmeros *endpoints* em WSO2 EI como *AsmissionTypesEP*, *AllergyEP*, *AttitudeEP*, entre outros. Estes *endpoints* são definidos com recurso aos *endpoints* presentes em cada serviço externo incluindo o *ePatient*. Na Figura 30, a nível de exemplo, encontram-se representados alguns dos *endpoints* definidos para a solução.

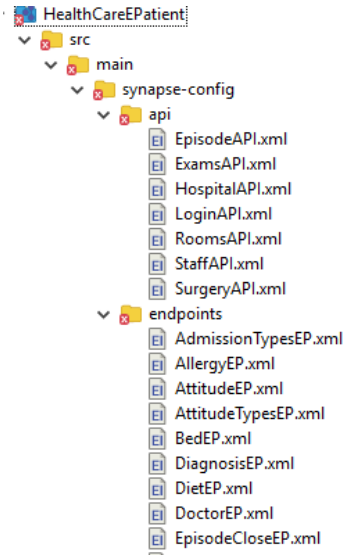

Figura 30 – *Endpoints* de ligação ao ePatient

Nesta figura é possível ver a representação das duas camadas definidas como necessárias para a adaptação dos vários módulos existentes, definidas no ponto 5.3. Estão então representadas a camada *Process*, representada na pasta "api", e a camada *System*, representada pela pasta "endepoints".

6.1.2 Endpoints

Fazendo uso da Figura 30, cada *endpoint* deve ser configurado anteriormente ao desenvolvimento das APIs em si, para que estes não sejam repetidamente desenvolvidos para cada APIs que use o mesmo *endpoint* repita informação do mesmo. Os *endpoints* quando configurados exigem o preenchimento de certas definições e campos, sendo os mais importantes, a definição do endereço e do método que representa (GET, POST, etc.). Na Figura 31, a título de exemplo, é possível ver a configuração do *endpoint PostLoginEP*.

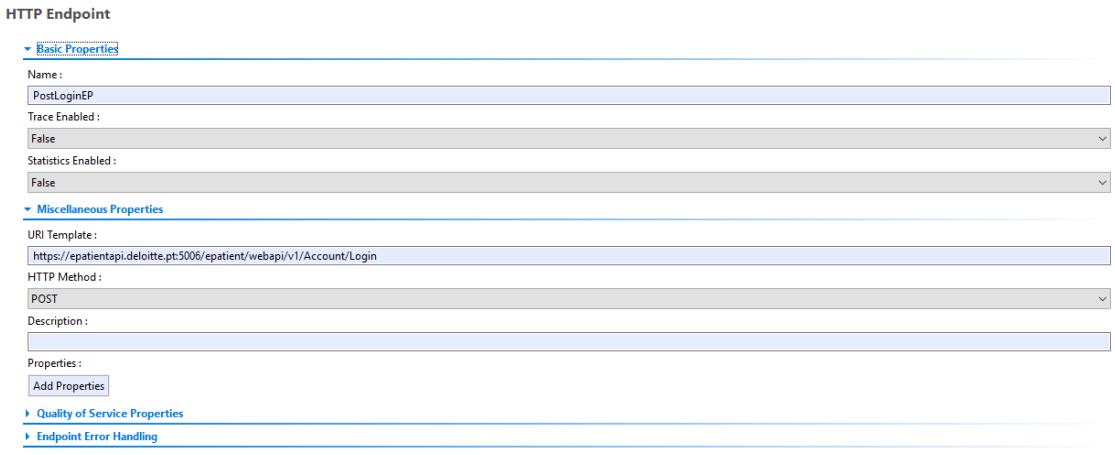

Figura 31 – Configurações do *Endpoint*

Nesta configuração é atribuído um nome ao *endpoint*, para que possa ser referenciado nos *resources*, é atribuído um "*URI Template*", este é o endereço a que se deve ligar o *endpoint*, e, por fim, é atribuído o método a executar, neste caso é um POST.

Após a definição de todos os *endpoints* e das APIs definidas, é possível a implementação correta dos casos de uso.

6.1.3 Sequências

Nos próximos subcapítulos será demonstrado a implementação de três *resources* em WSO2, para além da API Login, assim como a reutilização de certos componentes e como é feito o login para aceder aos serviços aos serviços do ePatient Core. Os restantes *resources* seguiram a mesma linha de desenvolvimento.

6.1.3.1 Login

A API responsável por providenciar um *token* de acesso aos pedidos diretos do *ePatient*, designa-se pela *APILogin*. Esta API funciona de uma maneira particular, como foi mencionado no capítulo anterior de *design*.

Como boa prática, é necessário encontrar uma maneira de não haver um pedido sistemático ao sistema por novos *tokens*, como quando os *tokens* gerados anteriormente estão dentro do prazo.

Existe, em WSO2, uma função capaz de armazenar em memoria aquilo que o programador pretende, pelo tempo pretendido atribuído pelo programador. Esta função resolve o problema apresentado em cima, com a utilização desta, são evitados os pedidos sistemáticos ao sistema que ainda se encontram dentro do prazo. Com isto, foi então decidido a sua implementação no código da solução.

Na Figura 32 é então possível verificar a forma de como o requisito do *token* funciona. Como o mesmo tem um prazo de validade é necessário verificar que este vá sendo atualizando constantemente.

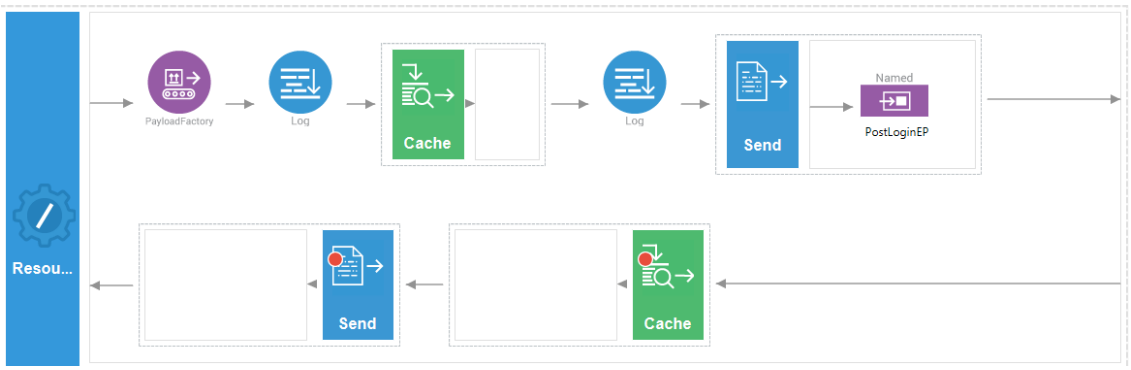

Figura 32 – Implementação login

O primeiro icon representativo da *cache* é responsável por verificar se existe alguma informação em memoria capaz de satisfazer o pedido. Quando não existe, ou até mesmo o que existe já passou a validade, faz o pedido em questão ao *ePatient*, icon *Send*, requisitando um novo *token*. No caso de existir em *cache*, passa logo para a função de retorno, segunda linha da ilustração.

No retorno existem duas funções importantes, novamente a de *cache*, tem como função guardar o novo valor caso se tenha recorrido à chamada de um novo valor para o *token*, e, posteriormente, existe o envio da resposta que, mais tarde, em cada *resource* irá ser trabalhado e contruído o pedido consoante o respetivo *token*.

Existe outra maneira de representar o pedido de um novo *token*, apresentado na Figura 33. Nesta a sequencia é iniciada no *payloadFactory*, esta tem como função criar a mensagem que será enviada. No passo seguinte é então iniciada a função *cache* que irá armazenar o *token* gerado pelo pedido. Neste exemplo o *token* tem um período de vida de 1200 milissegundos. É de referir que caso exista um *token* guardado com as mesmas propriedades em *cache* é então redirecionado logo para a resposta, como referido no paragrafo anterior. Por fim, caso não exista qualquer *token* guardado em *chache*, é feito o pedido.

```
<inSequence>
    <payloadFactory media-type="json">
        <format key="conf:myresources/LoginConfig.json"/>
        <args/>
    </payloadFactory>
    <log level="full"/>
    <cache collector="false" maxMessageSize="2000" timeout="1200">
        <onCacheHit/>
        <protocol type="HTTP">
            <methods>*</methods>
            <headersToExcludeInHash/>
            <responseCodes>.*</responseCodes>
            <enableCacheControl>false</enableCacheControl>
            <includeAgeHeader>false</includeAgeHeader>
            <hashGenerator>org.wso2.carbon.mediator.cache.digest.REQUESTHASHGenerator</hashGenerator>
        </protocol>
        <implementation maxSize="1000"/>
    </cache>
    <log level="full"/>
    <send>
        <endpoint key="PostLoginEP"/>
    </send>
</inSequence>
```

Figura 33 – Pedido API Login

Quando enviado o pedido, a resposta é construída, representado na Figura 34. A resposta é guardada no *outSequence*, contendo o novo *token* e a informação de que a resposta foi enviada.

```
<outSequence>
    <cache collector="true"/>
    <send/>
</outSequence>
```

Figura 34 – Resposta API Login

6.1.3.2 Put Surgery

O desenvolvimento deste caso de uso será na sua API respetiva, neste caso *Surgery* API. Este caso de uso é responsável pela criação de uma nova cirurgia. Na Figura 35 é possível observar esta implementação em WSO2.

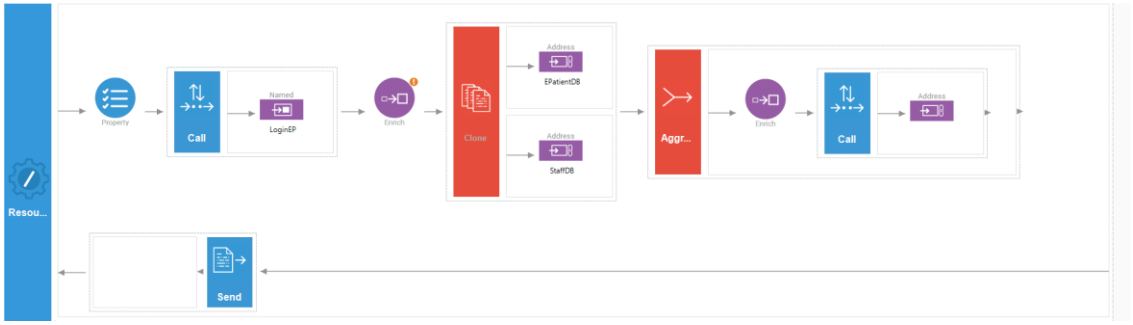

Figura 35 – Implementação Put Surgery

Para o desenvolvimento deste caso de uso, inicialmente é necessário o *token* que permite a comunicação com o *ePatient*, representado na figura como a primeira *Call* efetuada. Depois de o pedido possuir as informações do *token*, são então realizadas as várias chamadas às APIs que contêm a informação necessária, neste caso à *Rooms* API e Staff API para recolher a informação. Quando recolhida a informação necessária, a mesma é agregada numa mensagem apenas para que o pedido chegue ao *ePatient* através do *endpoint* definido no início, como representado na Figura 30.

6.1.3.3 Put Episode Open

O desenvolvimento deste caso de uso recai numa API diferente da anterior, este caso de uso trata do registo de um novo episodio, por outras palavras, do internamento de um novo paciente. Este caso de uso será então desenvolvido na *Episode* API. Na Figura 36 é possível observar esta implementação em WSO2.

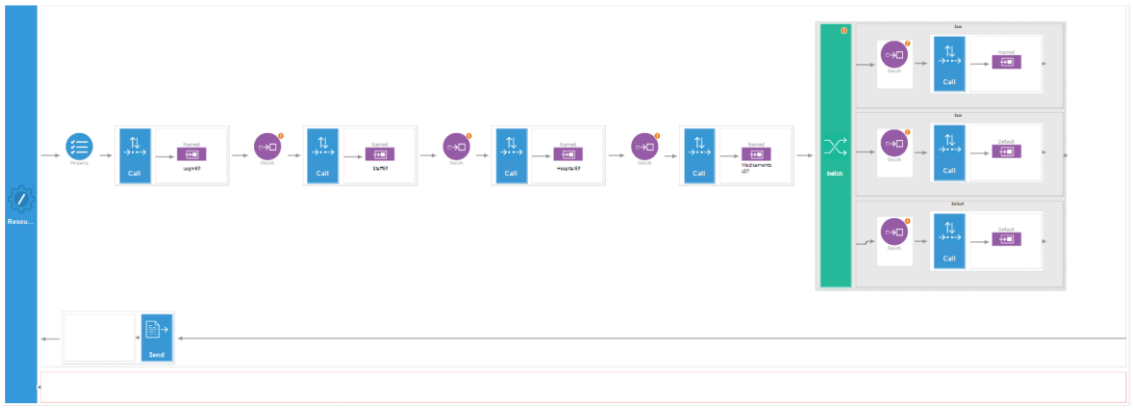

Figura 36 – Implementação Put Episode Open

Como no caso de uso anterior, inicialmente é feita a chamada para receber o *token*. Neste caso, também é necessário ir buscar informação a várias APIs diferentes, sendo que neste foi utilizado um desenvolvimento diferente.

Quando a mensagem já possui o *token* necessário para estabelecer comunicação com o *ePatient*, são necessárias mais três chamadas diferentes, *Staff* API, *Rooms* API e Medicamentos API, para verificar a disponibilidade dos vários serviços, isto é, se se encontram disponíveis para suportar mais um paciente.

Por fim, e assumindo que o hospital tem capacidade para internar o paciente, é então implementado um *switch*. Este *switch* redireciona o paciente para a especialidade que vem inicialmente na mensagem para registar um novo episódio. A título de exemplo, e dando continuidade ao que foi apresentado na descrição deste caso de uso, são três especialidades.

6.2 Especificações no WSO2 API Manager

Quando terminado o desenvolvimento de cada API, é necessário configurar o *API Manager*. Como referido no ponto 5.3.1, é *o API Manager* que trata da gestão das várias APIs. Esta gestão passa pelo controlo de versões, permitir/bloquear acessos e chamadas às mesmas e, mais importante, a análise do tráfego de dados da solução.

De início foi instalado o WSO2 *API Manager* 2.6.0 com a configuração de origem, o que permitiu a configuração de cada uma das APIs. Na Figura 37 é possível ver a interface gráfica do WSO2 API Manager 2.6.0 com todas as APIs que foram descritas no ponto 6.2. Todas foram configuradas e conectadas com as soluções desenvolvidas no WSO2 ESB.

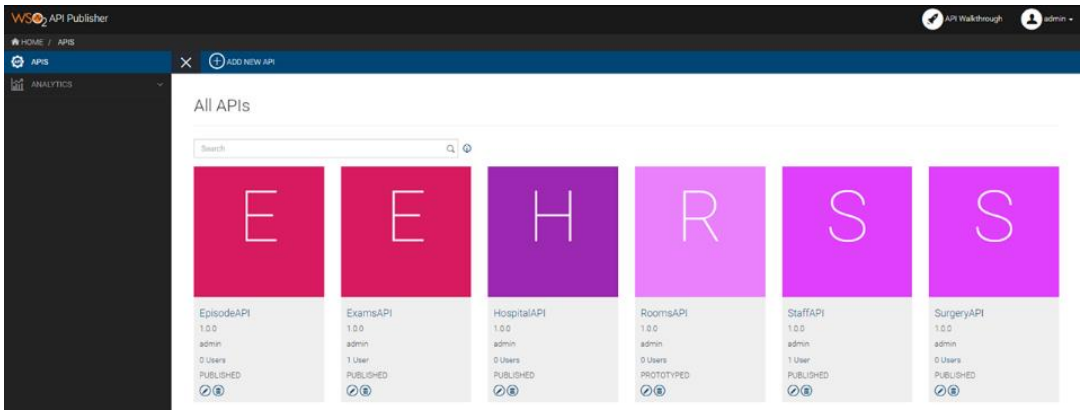

Figura 37 – WSO2 API Manager

Quando selecionamos, por exemplo, a *EpisodeAPI*, são apresentados os detalhes relacionados com esta API e todos os endereços que tornam possível o acesso a esta, estes detalhes são possíveis ver na Figura 38. Relembrando, esta API é responsável pela gestão de internamentos e dados relacionados com o paciente.

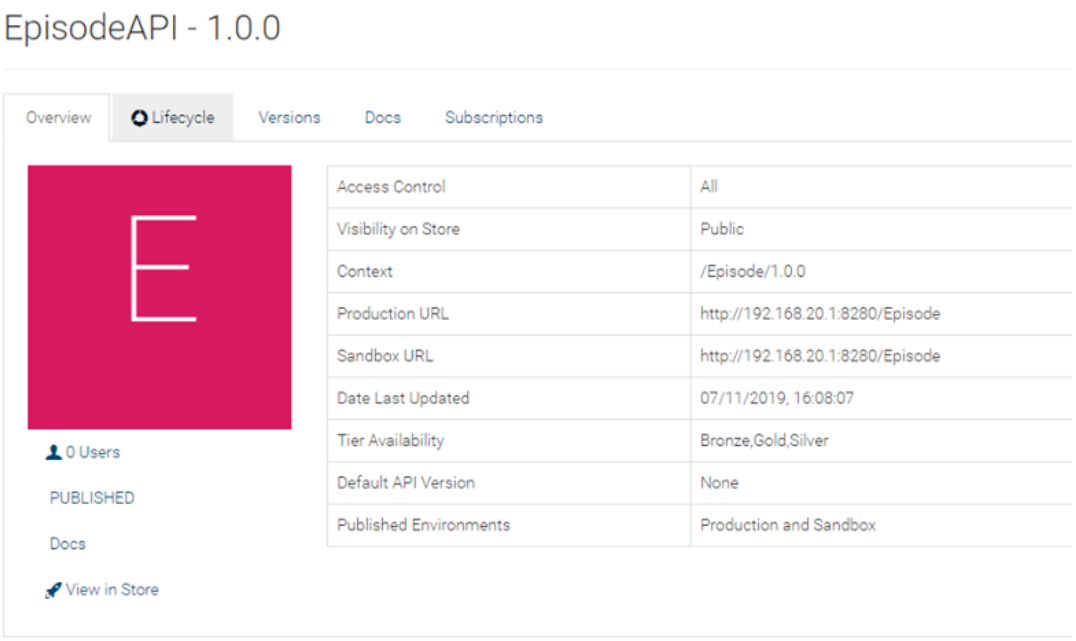

Figura 38 – Configuração de uma API

É também possível ter acesso às análises realizadas às APIs. Nesta análise é possível verificar todos os dados referentes à mesma e visualizar todos os subscritores que a mesma tem. Na Figura 39 é possível ver parte desta análise, onde são apresentados os subscritores, ao longo do tempo, de todas as APIs.

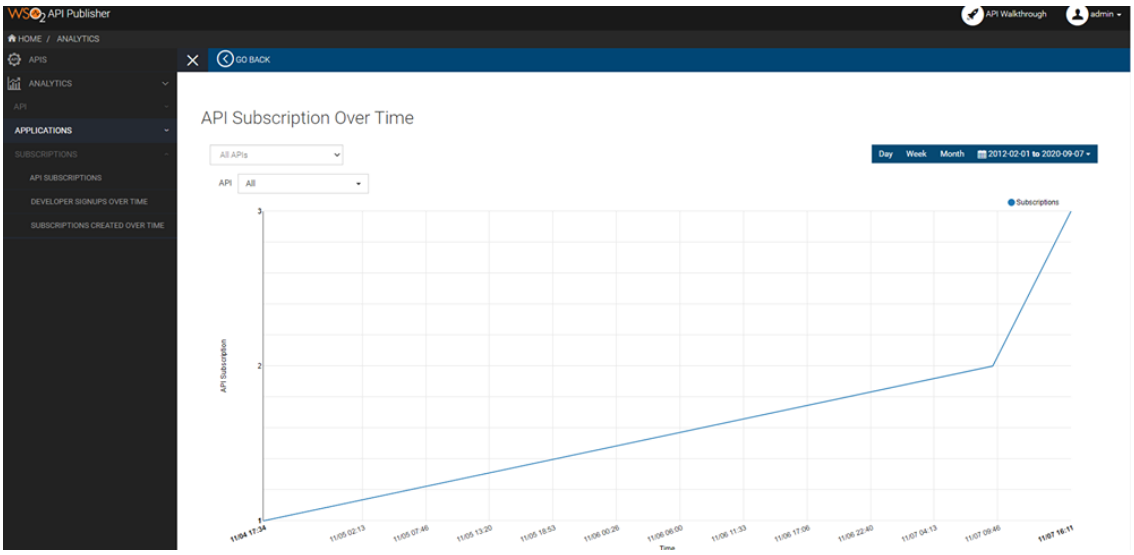

Figura 39 – Análise da API

6.3 ePatient Core

São também necessárias algumas alterações ao componente *ePatient Core*. Estas alterações recaem principalmente no componente *Services*. Este componente comunicava anteriormente com o *External Integration*, passando a comunicar com o WSO2 Suite, como representado na Figura 20, tendo sido necessária a alteração da lógica do mesmo.

Concluindo assim o fim da lógica acoplada dentro do componente *ePatient Core* havendo uma maior clareza nas responsabilidades que cada componente contém.

Capítulo 7

Experiências e Avaliação

Neste capítulo serão especificados, descritos e efetuados um conjunto de testes à solução desenvolvida. Estes testes têm o objetivo de aferir a qualidade da mesma em relação aos objetivos traçados.

São então apresentados quais os testes a serem efetuados, uma pequena descrição de como esses mesmos testes são realizados, os resultados obtidos e, por fim, uma pequena reflexão sobre os resultados obtidos.

7.1 Objetivos

Para avaliar o desempenho e qualidade da solução desenvolvida, tendo sempre como objetivo a resolução do problema inicial desta dissertação, decidiu-se realizar dois tipos diferentes de testes, capazes de validar o valor da solução:

- Testes de *performance*: estes testes visam avaliar o desempenho em termos de tempo despendido no processo de integração por comparação entre o sistema anterior e o novo sistema desenvolvido em WSO2;

- Teste de rapidez de evolução/adaptação do sistema: visou o desenvolvimento de uma nova integração, em que um grupo de pessoas testam o tempo necessário para adicionar um novo sistema externo com o qual o *ePatient* tem de comunicar e comparar esse mesmo tempo de desenvolvimento;

7.2 Testes De Performance

O teste de *performance* neste caso específico, tem como principal objetivo comparar as duas soluções existentes, antiga e nova, e concluir qual a mais rápida a executar os pedidos.

Uma das opções para realizar estes testes é optando pelo software disponibilizado pela *Postman*, através da sua aplicação onde é possível simular todo o tipo de chamadas e pedidos.

Os seguintes testes realizados no *Postman* nada mais são que chamadas ao serviço desenvolvido. Quando realizada essa chamada, é devolvida a resposta em conjunto com o tempo de demora dessa resposta. É este tempo de resposta utilizado para comparação das duas soluções, a existente anteriormente e a nova solução desenvolvida.

7.2.1 Preparação e Configuração

É utilizada então a aplicação *Postman* 7.31.0 para a realização deste teste. Com as soluções alojadas numa máquina virtual Windows server 2012 com 16gb de RAM, os pedidos efetuados foram aos seguintes *endpoints*:

- Get Nurse;
- Get Speciality;
- Get Surgery;
- Get Room;
- Get Exams;
- Get Episode History;

A escolha dos pedidos teve como base a sua variada complexidade. Foram escolhidos pedidos simples (*Get Nurse*), pedidos com alguma lógica (*Get Surgery*) e pedidos mais complexos (*Get Episode History*). Recorreu-se especialmente a pedidos *Get*, devido à comodidade e para não estar a criar registos novos desnecessários na base de dados, acrescentada informação dispensável e inútil na mesma.

Para o teste, cada um dos pedidos foi executado cinco vezes cada uma das soluções, para cada uma das soluções a testar. O tempo, medido em segundos, a considerar para cada pedido é, como referido em acima, aquele apresentado pelo Postman quando a resposta à chamada é apresentada.

7.2.2 Resultados Obtidos

Os resultados obtidos foram recolhidos e tratados na plataforma *excel* para posteriormente ser de análise mais fácil, por exemplo, na análise de gráficos de comparação.

Inicialmente foi testada a nova solução e registados os tempos dos cinco pedidos, para cada chamada a realizar, numa tabela e calculado a média de tempo de cada chamada, representado na Tabela 3.

Tabela 3 – Tabela de resultados em segundos para a nova solução

	GET/Nurse	GET/Speciality	GET/Surgery	GET/Room	GET/Exams	GET/EpisodeHistory
1º pedido	1,172	1,472	1,521	1,382	1,714	2,102
2º pedido	1,176	1,411	1,538	1,425	1,731	1,951
3º pedido	1,18	1,424	1,552	1,328	1,728	1,972
4º pedido	1,232	1,382	1,47	1,394	1,693	1,893
5º pedido	1,032	1,391	1,516	1,382	1,702	2,215
Média:	1,153	1,416	1,532	1,382	1,714	2,027

Na chamada GET/Nurse, o tempo mínimo registado do pedido foi 1,032 segundos, o tempo máximo registado do pedido foi 1,18 segundos, apresentando uma média de tempo de resposta de 1,153 segundos.

Na chamada GET/Speciality, o tempo mínimo registado do pedido foi 1,382 segundos, o tempo máximo registado do pedido foi 1,472 segundos, apresentando uma média de tempo de resposta de 1,416 segundos.

Na chamada GET/Surgery, o tempo mínimo registado do pedido foi 1,47 segundos, o tempo máximo registado do pedido foi 1,538 segundos, apresentando uma média de tempo de resposta de 1,532 segundos.

Na chamada GET/Room, o tempo mínimo registado do pedido foi 1,328 segundos, o tempo máximo registado do pedido foi 1,425 segundos, apresentando uma média de tempo de resposta de 1,382 segundos.

Na chamada GET/Exams, o tempo mínimo registado do pedido foi 1,693 segundos, o tempo máximo registado do pedido foi 1,728 segundos, apresentando uma média de tempo de resposta de 1,714 segundos.

Por fim, na chamada GET/EpisodeHistory, o tempo mínimo registado do pedido foi 1,893 segundos, o tempo máximo registado do pedido foi 2,215 segundos, apresentando uma média de tempo de resposta de 2,027segundos.

Com estes resultados, foi elaborado um gráfico com as médias, Figura 40, onde é possível observar a diferença, em segundos, do tempo de resposta média às várias chamadas, sendo que as chamadas que contêm mais lógica são as que mais demoram no tempo de resposta.

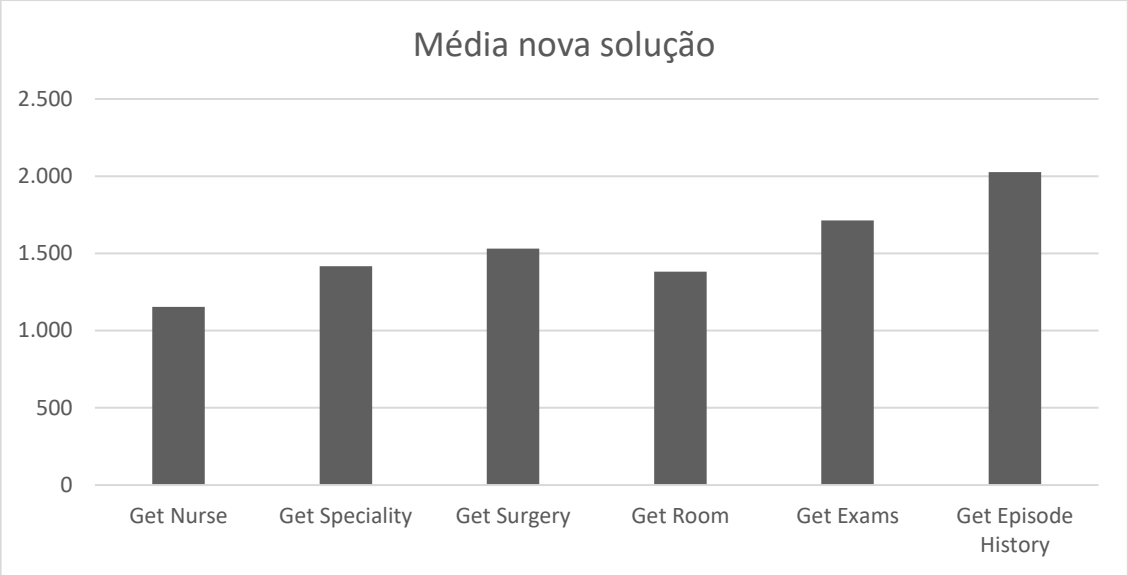

Figura 40 – Gráfico da média de resultados referentes aos pedidos à nova solução

Foi testada, depois, a antiga solução e registados os tempos dos cinco pedidos, para cada chamada a realizar, com recurso ao mesmo método de recolha de tempos que foi utilizado para o teste da solução nova numa tabela e calculado a média de tempo de cada chamada, representado na Tabela 4.

Tabela 4 – Tabela de resultados para a antiga solução (em seg.)

	GET/Nurse	GET/Speciality	GET/Surgery	GET/Room	GET/Exams	GET/EpisodeHistory
1º pedido	1,342	1,372	1,527	1,473	1,921	2,503
2º pedido	1,276	1,479	1,593	1,432	1,972	2,493
3º pedido	1,172	1,512	1,602	1,482	1,881	2,205
4º pedido	1,532	1,502	1,572	1,407	2,043	2,259
5º pedido	1,526	1,439	1,562	1,521	1,852	2,839
Média:	1,370	1,461	1,571	1,463	1,934	2,460

Na chamada GET/Nurse, o tempo mínimo registado do pedido foi 1,172 segundos, o tempo máximo registado do pedido foi 1,532 segundos, apresentando uma média de tempo de resposta de 1,370 segundos.

Na chamada GET/Speciality, o tempo mínimo registado do pedido foi 1,372 segundos, o tempo máximo registado do pedido foi 1,502 segundos, apresentando uma média de tempo de resposta de 1,461 segundos.

Na chamada GET/Surgery, o tempo mínimo registado do pedido foi 1,527 segundos, o tempo máximo registado do pedido foi 1,602 segundos, apresentando uma média de tempo de resposta de 1,571 segundos.

Na chamada GET/Room, o tempo mínimo registado do pedido foi 1,407 segundos, o tempo máximo registado do pedido foi 1,521 segundos, apresentando uma média de tempo de resposta de 1,463 segundos.

Na chamada GET/Exams, o tempo mínimo registado do pedido foi 1,852 segundos, o tempo máximo registado do pedido foi 2,043 segundos, apresentando uma média de tempo de resposta de 1,934 segundos.

Por fim, na chamada GET/EpisodeHistory, o tempo mínimo registado do pedido foi 2,205 segundos, o tempo máximo registado do pedido foi 2,839 segundos, apresentando uma média de tempo de resposta de 2,460 segundos.

Com estes resultados, foi elaborado um gráfico com as médias, Figura 41, onde é possível observar a diferença, em segundos, do tempo de resposta média às várias chamadas, sendo que as chamadas que contêm mais lógica são as que mais demoram no tempo de resposta.

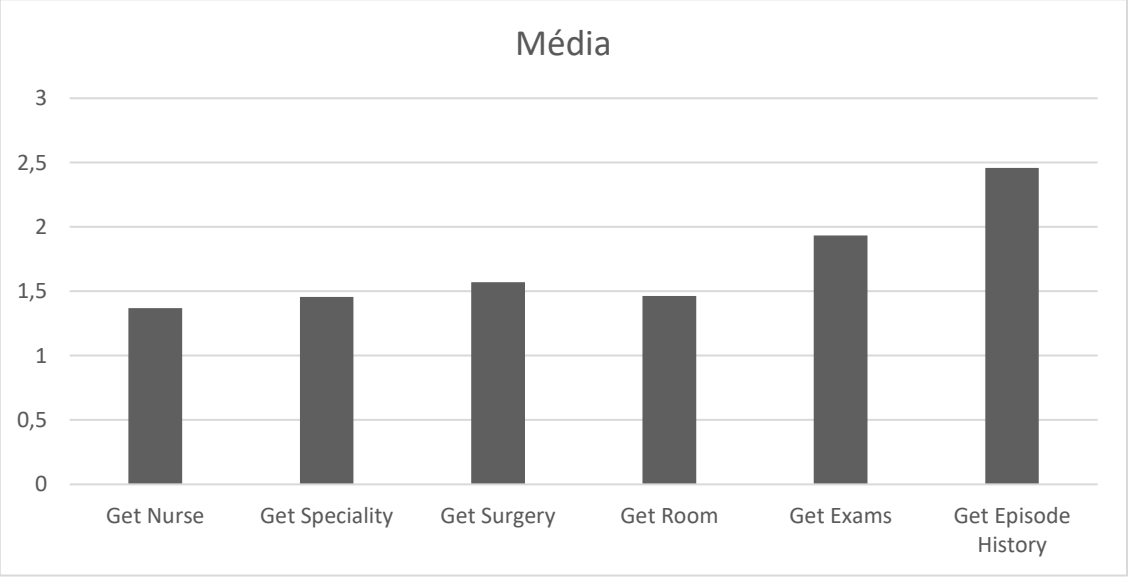

Figura 41 – Representação gráfica dos resultados referentes aos pedidos à antiga solução

7.2.3 Análise dos Resultados

Com os resultados obtidos anteriormente, é possível verificar uma discrepância de resultados entre as duas soluções construídas.

Para que fosse possível uma melhor análise comparativa dos tempos obtidos relativos às várias chamadas, foi construído um gráfico de barras, Figura 43, que apresenta a média de tempo, em segundos, de resposta das várias chamadas em teste.

Analisando ao pormenor pedido GET/Nurse, na Figura 42 é possível verificar que, apesar de pouco significativa, existe diferença rapidez de resposta. Em quatro dos cinco pedidos, apenas um da solução antiga foi capaz de responder mais rápido.

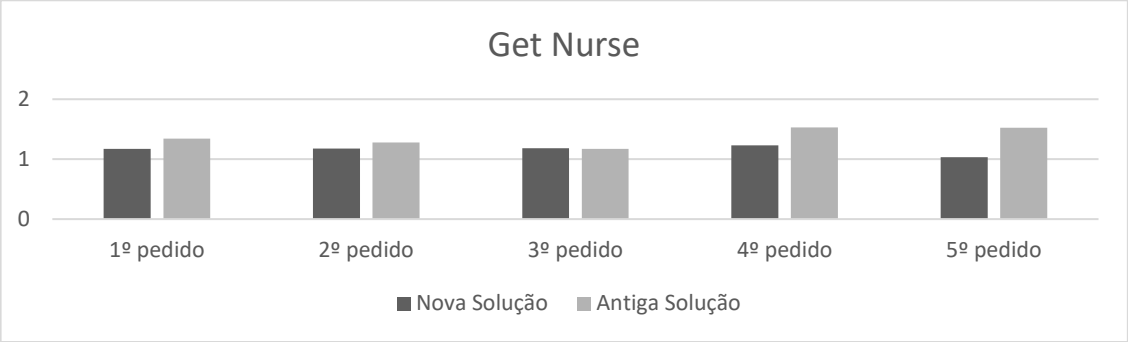

Figura 42 – Representação gráfica dos pedidos *Get Nurse*

Tanto como no pedido GET/*Nurse*, apresentado acima, como os restantes pedidos, apresentados no gráfico da Figura 43, é possível verificar que, em todos os pedidos, a média do tempo de resposta é mais rápido na nova solução.

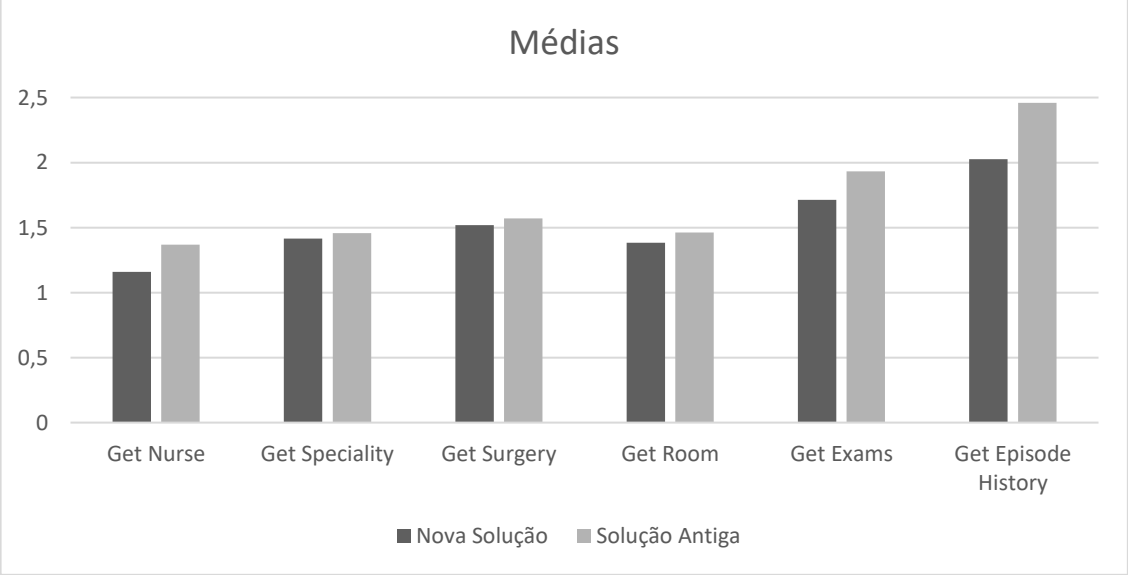

Figura 43 – Médias de resultados das soluções desenvolvidas

Como referido para a análise do pedido GET/*Nurse*, apesar das diferenças em alguns tempos de resposta serem quase nulos, é de extrema importância que na área em questão, área da saúde, seja sempre possível trabalhar com menores tempos de resposta possíveis.

7.3 Teste de rapidez de evolução/adaptação

Neste teste o objetivo é adicionar às soluções existentes, antiga e nova, um novo conjunto de serviços que deverão comunicar com as já existentes. No final do desenvolvimento o tempo de cada programador será registado e feito um resumo a comparar os resultados obtidos.

7.3.1 Preparação e Configuração

Para o desenvolvimento desta experiência foi necessário criar um novo serviço, neste caso um serviço de fisioterapia, capaz de comunicar com o *ePatient*, recebendo e enviando pedidos, como representado Figura 44.

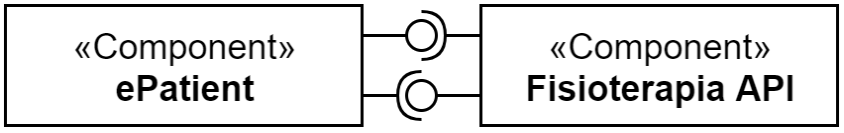

Figura 44 – Representação gráfica comunicação entre os dois sistemas

O componente Fisioterapia API necessitava de informação relativa a um paciente para que seja possível efetuar uma marcação, informação como nome e problema a tratar, para isto é necessário que o componente *ePatient* disponibilize tal funcionalidade.

Por sua vez, o componente *ePatient* tem de ser capaz de verificar as marcações de fisioterapia existes de um determinado paciente, para isto é necessário que o componente Fisioterapia API disponibilize um serviço capaz de atender a esta necessidade.

Por outras palavras, a Figura 44 demonstra através do diagrama, a existência de duas chamadas, um POST, com a responsabilidade de marcar uma nova sessão e um GET, com a responsabilidade de ir buscar todas as sessões de fisioterapia, de um dado paciente, existentes no sistema.

Na Figura 45 é possível ver a sequencia de ações despoletadas pela marcação de uma sessão de fisioterapia, para um dado paciente, por um médico. O médico solicita a marcação de uma sessão de fisioterapia através da aplicação *WhiteBoard*, esta comunica com o *ePatient*. Já no *ePatient*, este envia os dados necessário do paciente e a sessão que pretende marcar, a Fisioterapia API verifica os dados do paciente, se é um novo paciente, se já tem algumas sessões marcadas, e se a sessão que pretende está livre. Em caso de sucesso, a sessão é marcada e guardada na base de dados da Fisioterapia.

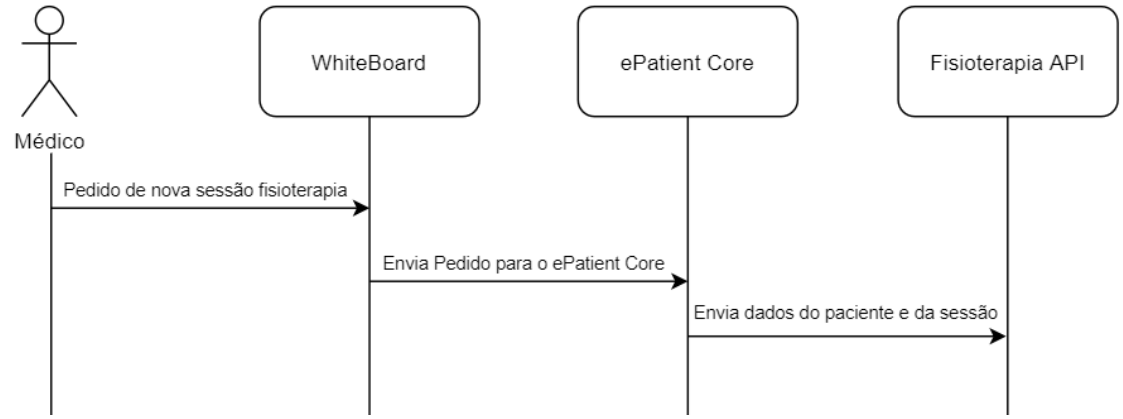

Figura 45 – Diagrama de sequência pedido de marcação de fisioterapia

A Figura 45 representa um fluxo é de muito alto nível, mas devido à sua baixa complexidade é possível desenvolver os testes através destes diagramas.

Já na Figura 46, é possível ver a sequencia de ações despoletadas pelo pedido de listagem de sessões, de um dado paciente, por um médico. O médico solicita a listagem através da aplicação *WhiteBoard*, esta comunica com o *ePatient*. Já no *ePatient*, este envia os dados necessário do paciente, a Fisioterapia API pesquisa os dados do paciente, se existem sessões realizadas, se existem sessões por realizar, se não existe qualquer sessão marcada. Em caso de sucesso, a listagem é retornada no *WhiteBoard* para que o médico consiga ter acesso à mesma.

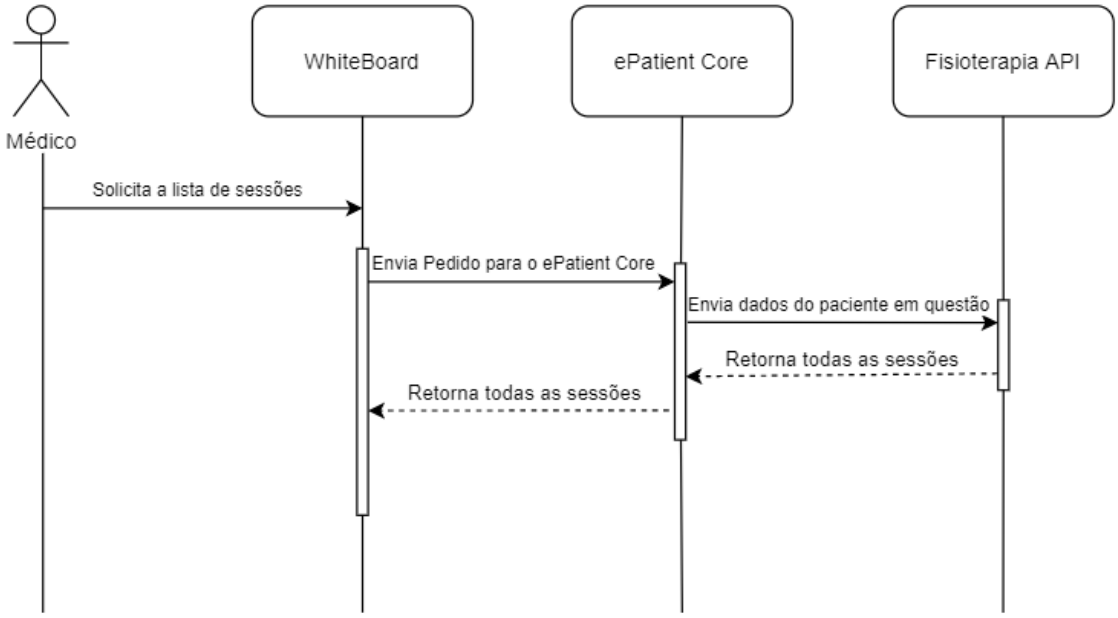

Figura 46 – Diagrama de sequência listar sessões

Como foi referido na Figura 45, na Figura 46 também se trata de um pedido simples para listar as sessões marcadas no sistema da Fisioterapia, novamente é uma chamada simples apenas para efeitos de teste e comparativo entre soluções desenvolvidas.

Para realizar este teste e os desenvolvimentos necessários foram escolhidas pessoas com experiencia em WSO2 (superior a um ano) e com base de programação em .NET, a linguagem em que o *ePatient* foi desenvolvido.

Para além dos requisitos mencionados, houve também uma pequena introdução às duas soluções desenvolvidas para que os resultados das experiências não ficassem dependentes da compreensão de cada desenvolvedor. Neste caso foram escolhidos 3 voluntários, colaboradores da Deloitte, que preenchiam estes mesmos requisitos.

Os três voluntários encontram-se entre os 25 e os 31 anos, todos do sexo masculino. O primeiro voluntário de 31 anos tendo começado recentemente a trabalhar com projetos de integração, e consequentemente com ferramentas de integração. O segundo voluntário de 27 anos trabalhou desde o início da sua carreira com ferramentas de integração. Por último, o terceiro voluntario de 25 anos também ele apenas trabalhou com projeto de integração desde o início da sua carreira, há cerca de 2 anos.

Por questões de segurança, os envolvidos nestes testes serão tratados por sujeito A, sujeito B e sujeito C, respetivamente.

7.3.2 Resultados Obtidos

Após uma breve explicação sobre a maneira como ambas as soluções estavam desenvolvidas era tempo de os voluntários fazerem os seus desenvolvimentos para no final comparar o tempo que cada um levou até completar a tarefa em ambas as soluções, todos os voluntários tiveram de começar a desenvolver na solução antiga antes de passar para os desenvolvimentos da nova solução. Os resultados obtidos são medidos em minutos e registados numa tabela, Tabela 5, para uma melhor análise dos mesmos.

Tabela 5 – Resultados do teste de rapidez de evolução/adaptação

	Nova Solução	Solução Antiga
Sujeito A	120	240
Sujeito B	90	165
Sujeito C	150	170

Por uma breve análise à Tabela 5, é possível verificar que integrar um novo componente à nova solução é mais rápido do que integrar na solução antiga. Para uma melhor comparação elaborou-se o gráfico da Figura 47.

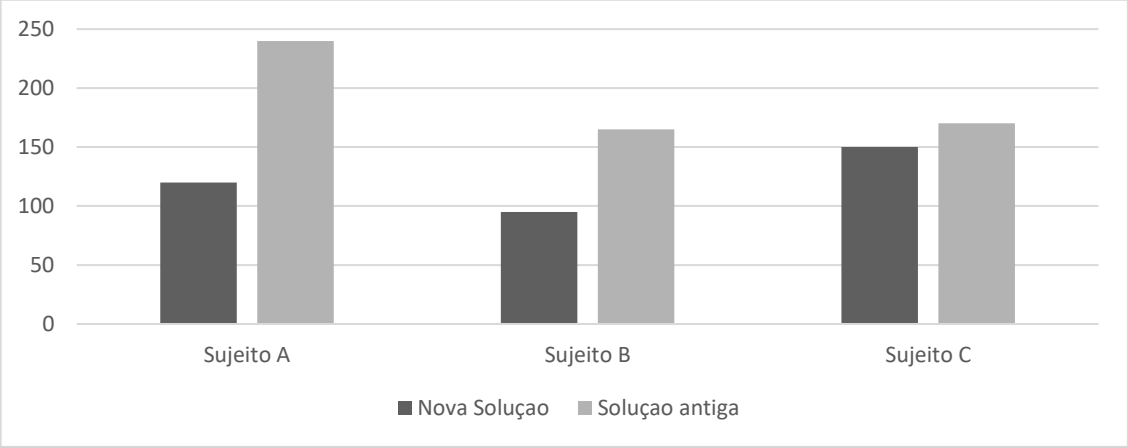

Figura 47 – Representação gráfica de os resultados teste de escalabilidade

Na Figura 47, nas barras do lado esquerdo, tom mais escuro, estão representados os tempos necessários para integrar o novo componente na nova solução. Nas barras do lado direto, tom mais claro, estão representados os tempos necessários para integrar o novo componente na solução antiga.

7.3.3 Análise dos Resultados

Após os desenvolvimentos efetuados pelos voluntários e apresentados no subcapítulo anterior, é possível ver uma diferença ainda significativa do tempo de desenvolvimento consumido em cada uma das soluções.

O novo sistema foi mais rápido de implementar na solução desenvolvida em WSO2, em comparação com a desenvolvida em .NET. Observando o sujeito A, torna-se mais claro o que foi referido. Quando observado o sujeito C, não se nota uma diferença tao significativa, isto acontece, pois, o sujeito A, apesar de possuir conhecimentos em .NET, já não trabalha com isso há algum tempo, ao contrario do sujeito C que neste momento integra um projeto que inclui integração e programação em .NET.

Com isto, e dados os diversos casos apresentados de teste, torna-se claro que a solução em WSO2 é mais benéfica no que toca à facilidade de evolução/adaptação do sistema.

É possível retirar destes testes que uma solução em WSO2 torna-se, financeiramente, melhor dado que o desenvolvimento das suas atualizações requer menos tempo e trabalho.

Capítulo 8

Conclusão

No desenvolvimento desta dissertação foi abordado, estudado e analisado um problema crítico no desenvolvimento de sistemas que necessitam de comunicação com aplicações externas. No mundo atual é cada vez mais indispensável a existência de sistemas de integração. O cenário atual de pandemia (COVID-19) veio demonstrar a existência de um problema crítico na comunicação entre sistemas independentes e heterogéneos e a necessidade de rapidez de integração.

Por fim, neste último capítulo serão apresentadas as conclusões do estudo efetuado nesta dissertação. Será apresentada uma contextualização do problema e, de seguida, uma avaliação do trabalho efetuado relativamente aos objetivos definidos no início do trabalho. Os pontos que não foram, até ao momento, realizados serão deliberados, com o intuito de os incorporar em desenvolvimentos futuros. Por fim, é feita a apreciação final sobre o trabalho desenvolvido.

8.1 Resumo

O trabalho aqui desenvolvido pode ser dividido em duas partes. Uma primeira parte que consistiu no estudo sobre os conceitos, as ferramentas e as vantagens de realizar integração de sistemas recorrendo a soluções baseadas em intermediação. O resultado desta primeira parte

trouxe então a segunda parte. A segunda parte consistiu no desenvolvimento de uma nova solução, numa tecnologia nova, e com uma arquitetura melhor estruturada e com facilidade de escalabilidade.

Já nos tempos pré-pandemia era possível notar uma falta de capacidade e conhecimento das empresas na disponibilização de interfaces capazes de comunicar com serviços externos. Com isto, também não era uma preocupação fazer uma monitorização mais detalhada desses mesmo serviços externos.

O desenvolvimento desta dissertação veio ao encontro disto mesmo, tornar a aplicação mais maneável, mais ajustável a qualquer sistema externo, mais ainda, fazendo sempre monitorização dos fluxos de dados da aplicação.

Inicialmente, a escolha da tecnologia foi um dos passos mais difíceis a tomar dado a grande diversidade das mesmas. Esta escolha teria de ser cuidada e segundo alguns critérios definidos inicialmente capaz de filtrar este número enorme de tecnologias existentes no mercado. Estes critérios recaíram sobre o problema, e o que se adapta melhor à solução, como ao desenvolvedor, dado que este terá de aprender, desenvolver a solução e, por fim, desenvolver esta dissertação num período estabelecido.

Apos a escolha da tecnologia a utilizar, WSO2, tudo se tornou mais simples, dado que se trata de uma tecnologia simples de aprender e fácil de desenvolver. Para juntar a isto, toda a informação e pedidos necessários do *ePatient* já se encontravam desenvolvidos, necessitando apenas de serem reencaminhados para a nova camada responsável pela comunicação.

Por fim, e após várias testagens e comparações finais, chegou-se à conclusão referida desde o início, a integração ponto a ponto é rudimentar e arquitecturalmente confusa. Isto acontece devido ao facto de a integração de sistemas ser capaz de mapear e esconder logica do desenvolvedor que a integração ponto a ponto não é capaz de o fazer. Nos dois testes realizados no capítulo anterior é possível ver os passos dados para chegar a esta conclusão.

8.2 Satisfação dos Objetivos

Os requisitos definidos para esta dissertação foram o desenvolvimento de uma camada intermédia de integração e a configuração de uma ferramenta de monitorização de APIs. Foram estudadas as diferentes hipóteses referentes a tecnologias de integração de sistemas. No final WSO2 foi escolhido e, a partir desta decisão, todos os requisitos foram desenvolvidos, começando por todos os pedidos que foram descritos no capítulo 0, e, no final, a configuração do WSO2 API Manager, capaz de fazer a monitorização do fluxo de dados.

Por fim, com o capítulo de testes, foi provado os benefícios que a nova solução traz para o desenvolvimento e instalação do *ePatient* em novos hospitais.

8.3 Limitações e Trabalho Futuro

Identificou-se que o *ePatient* não comunica com os sistemas externos recorrendo a *standards* internacionais como o HL7 e outros. Tal pode ser visto como uma limitação do sistema atual que a nova arquitetura até permite endereçar de uma forma mais facilitada o uso destes padrões, e sendo assim, poderá vir a ser adaptado à solução.

Uma segunda limitação do trabalho desenvolvido foi a falta de um tratamento de erros para os casos no qual existe erro no tratamento de dados.

Para trabalho futuro existe alguns pontos a trabalhar, em primeiro lugar o desenvolvimento e aplicação dos padrões internacionais referentes à saúde devido ao facto de muitos hospitais e sistemas que este tem, usarem estes mesmo padrões. Outro ponto que pode também ser melhorado é a configuração do WSO2 API Manager, de forma a torná-lo mais seguro e restrito em termos de quem tem acesso ao que dentro de uma determinada API.

8.4 Considerações Finais

A dissertação desenvolvida foi uma mais-valia para o seu autor, que teve a oportunidade de desenvolver um projeto sozinho desde a sua fase inicial, com as dificuldades aí inerentes, onde o objetivo seria a aplicação de conceitos, análise e investigação, assim como muitas das matérias aprendidas durante o mestrado em engenharia de software, e as matérias aprendidas durante o próprio desenvolvimento do projeto.

O projeto, desde o início, se tornou um desafio, considerando as diversas tecnologias e ferramentas disponíveis no mercado, contudo, com a devida análise a cada uma das tecnologias, foi possível aprender rapidamente, que de facto obrigou a alguma investigação acrescida, que reduziu o tempo disponível para o desenvolvimento. Apesar das dificuldades ao longo do desenvolvimento do projeto, foi com muito gosto e dedicação que este foi desenvolvido.

Capítulo 9

Referências Bibliográficas

[1] R. Land and I. Crnkovic, “Existing Approaches to Software Integration-and a Challenge for the Future.” Accessed: Oct. 12, 2020. [Online]. Available: http://www.idt.mdh.se/%7B~rld,~icc%7D.

[2] “Web Services Architecture.” https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest (accessed Oct. 12, 2020).

[3] C. Tiernan, “Why Point-to-Point Integrations Are Evil | Chris Tiernan’s Blog,” Nov. 24, 2014. http://www.christiernan.com/why-point-to-point-integrations-are-evil/ (accessed Oct. 12, 2020).

[4] Glintt, “GlobalCare.” https://www.glintt.com/en/WHAT-WE-DO/Product-Services/SoftwareSolutions/Pages/GlobalCare.aspx.

[5] Globalcare | Powered by Glintt, “Hospital Management System - Globalcare.” https://globalcare.glintt.com/hospital-management-system/.

[6] Globalcare | Powered by Glintt, “Clinical - Globalcare.” https://globalcare.glintt.com/clinical/.

[7] F3M, “F3M Information Systems, S.A.” https://www.f3m.pt/pt/home.

[8] I. Equipamentos, “IBERDATA - SOFTWARE DE GESTÃO - iberdata.” http://www.iberdata.pt/p62-software-de-gestao-pt.

[9] S. A. ALERT Life Sciences Computing, “ALERT® PAPER FREE HOSPITAL (PFH) | ALERT® ONLINE - Português.” https://www.alert-online.com/pt/pfh.

[10] a S. company MuleSoft LLC, “Mule Message Structure | MuleSoft Documentation.” https://docs.mulesoft.com/mule-runtime/3.9/mule-message-structure (accessed Oct. 11, 2020).

[11] P. Fletcher and M. Waterhouse, *Web Services Business Strategies and Architectures - Mike Clark, Peter Fletcher, Jeffrey J. Hanson, Romin Irani, Mark Waterhouse, Jorgen Thelin - Google Livros*. EXPERT, 2002.

[12] a S. company MuleSoft LLC, “Reusing integration design patterns | MuleSoft,” 2020. https://www.mulesoft.com/resources/api/reusing-integration-design-patterns (accessed Oct. 11, 2020).

[13] J. Morcos, Z. Abedjan, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker, “DataXFormer: An Interactive Data Transformation Tool,” doi: 10.1145/2723372.2735366.

[14] P. Aubrecht and Z. Kouba, “Metadata Driven Data Transformation.” Accessed: Sep. 29, 2020. [Online]. Available: http://labe.felk.cvut.cz/~aubrech/bin/Sumatra.pdf.

[15] Thor Olavsrud, “Agile Comes to Data Integration,” *CIO*, Feb. 14, 2014. https://www.cio.com/article/2378615/agile-comes-to-data-integration.html (accessed Sep. 29, 2020).

[16] L. Chen, “Integrating Cloud Computing Services Using Enterprise Service Bus (ESB),” 2012.

[17] B. De, *API Management: An Architect’s Guide to Developing and Managing APIs for ... - Brajesh De - Google Livros*. Apress.

[18] a S. company MuleSoft LLC, “What is API Management? | MuleSoft.” https://www.mulesoft.com/resources/api/what-is-api-management.

[19] RESTfulAPI.net, “REST API Versioning Guide – REST API Tutorial.” https://restfulapi.net/versioning/.

[20] I. Amazon Web Services, “Throttle API requests for better throughput - Amazon API Gateway,” 2020.

https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-request-throttling.html (accessed Oct. 11, 2020).

[21] APIFriends.com by Axway, “API Policies: Why you Need Separate API Policies.” https://apifriends.com/api-management/need-separate-api-policies/.

[22] Jon Pierce, “Integration Frameworks and Enterprise Integration Patterns - Credera,” *Credera*, 2014. https://www.credera.com/blog/technology-insights/java/integration-frameworks-enterprise-integration-patterns/.

[23] I. Forrester Research, “Research · Forrester.” https://go.forrester.com/research/ (accessed May 19, 2020).

[24] Axway, “About Us | Digital Transformation | About Axway.” https://www.axway.com/en/company/about-axway.

[25] B. Inc, “API Full Lifecycle Management.” https://www.broadcom.com/products/software/api-management.

[26] T. Allen, “Apigee vs Mulesoft: What’s the difference? - Delta powered by Computing,” *delta Powered by: computing*, 2019. https://www.computingdelta.com/2019/10/23/apigee-vs-mulesoft/.

[27] Software AG, “API Management Platform & Integration | Software AG,” 2020. https://www.softwareag.com/en_corporate/platform/integration-apis/api-management.html (accessed Oct. 11, 2020).

[28] a S. company MuleSoft LLC, “About | MuleSoft.” https://www.mulesoft.com/about.

[29] TIBCO Software Inc., “API-led Integration | TIBCO Software,” 2020. https://www.tibco.com/api-led-integration (accessed Oct. 11, 2020).

[30] WSO2, “About WSO2 - Simplify API-First Integration.” https://wso2.com/about/.

[31] WSO2, “WSO2 API Manager Documentation 3.1.0.” https://apim.docs.wso2.com/en/latest/.

[32] WSO2, “Introduction - WSO2 Enterprise Integrator Documentation.” https://ei.docs.wso2.com/en/latest/micro-integrator/overview/introduction/.

[33] Oracle, “Oracle API Manager.” https://www.oracle.com/middleware/technologies/api-manager.html.

[34] IBM® Knowledge Center, “Configuring and managing your server environment.”

https://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.cmc.doc/con_cmc_overview.html.

[35] WSO2, “WSO2 Enterprise Service Bus - The Only 100% Open Source ESB.” https://wso2.com/products/enterprise-service-bus/.

[36] WSO2 Inc, “Developer Portal - API Manager 2.1.0 - WSO2 Documentation,” 2020. https://docs.wso2.com/display/AM210/Developer+Portal (accessed Oct. 11, 2020).

[37] a S. company MuleSoft LLC, “Core Components | MuleSoft Documentation.” https://docs.mulesoft.com/mule-runtime/4.2/about-components.

[38] a S. company MuleSoft LLC, “What is an ESB? | MuleSoft.” https://www.mulesoft.com/resources/esb/what-esb.

[39] Domenico Bellifemine, “Good Job Tibco BW | TrustRadius,” *TrustRadius*, 2017. https://www.trustradius.com/reviews/tibco-businessworks-2017-10-26-02-19-34.

[40] JavaTpoint, “JMS Tutorial - javatpoint,” 2018. https://www.javatpoint.com/jms-tutorial (accessed Oct. 14, 2020).

[41] I. Gartner, “Compare MuleSoft vs. TIBCO Software vs. WSO2 in Full Life Cycle API Management | Gartner Peer Insights.” https://www.gartner.com/reviews/market/full-life-cycle-api-management/compare/mulesoft-vs-tibco-vs-wso2inc.

[42] I. Forrester Research, “The Forrester Wave Methodology Guide · Forrester.” https://go.forrester.com/policies/forrester-wave-methodology/ (accessed Sep. 30, 2020).

[43] R. Heffner and K. Takeaways, “The 15 Providers That Matter Most And How They Stack Up,” API Management Solutions, 2018. [Online]. Available: https://vmark.eu/wp-content/uploads/2019/04/The-Forrester-Wave_-API-Management-Solutions-Q4-2018.pdf.

[44] R. Heffner, C. Mine, A. Livingston, and K. Hartig, “The Forrester Wave: API Management Solutions,” 2020, [Online]. Available: https://www.forrester.com/report/The+Forrester+Wave+API+Management+Solutions+Q3+2020/-/E-RES159081.

[45] SmartBear Software, “Bearer Authentication.” https://swagger.io/docs/specification/authentication/bearer-authentication/ (accessed Oct. 02, 2020).

[46] G. Hohpe, “Enterprise Integration Patterns - Message Store,” 2020. https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageStore.ht

ml (accessed Oct. 11, 2020).

[47] G. Hohpe, “Enterprise Integration Patterns - Content Enricher,” 2020. https://www.enterpriseintegrationpatterns.com/patterns/messaging/DataEnricher.html (accessed Oct. 11, 2020).

[48] G. Hohpe, “Enterprise Integration Patterns - Scatter-Gather,” 2020. https://www.enterpriseintegrationpatterns.com/patterns/messaging/BroadcastAggregate.html (accessed Oct. 11, 2020).

[49] G. Hohpe, “Enterprise Integration Patterns - Aggregator,” 2020. https://www.enterpriseintegrationpatterns.com/patterns/messaging/Aggregator.html (accessed Oct. 11, 2020).

[50] G. Hohpe, “Enterprise Integration Patterns - Message Router,” 2020. https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageRouter.html (accessed Oct. 11, 2020).

[51] A. De Brittos Valdati, R. Fernandes, and J. Artur De Souza, “FERRAMENTAS PARA AUXILIO NA IDENTIFICAÇÃO DE OPORTUNIDADES NO FRONT END DA INOVAÇÃO Área temática: Gestão do Conhecimento Organizacional Gertrudes Aparecida Dandolini,” 2016.

[52] D. Pereira, “Canvas da Proposta de Valor - O Analista de Modelos de Negócios,” 2019. https://analistamodelosdenegocios.com.br/canvas-da-proposta-de-valor/ (accessed May 19, 2020).

Anexo A

Análise de Valor

Como foi referido anteriormente, existem consequências referentes à não existência de uma camada de integração para uma aplicação que envolve a comunicação com tantos softwares externos.

Nesta secção é descrita a aplicação de *New Concept Development* (NCD) a este projeto, detalhando o valor gerado e por fim apresentado o modelo de negócio deste projeto.

A.1 Modelo NCD

O modelo NDC nasceu com o intuito de definir uma linguagem comum entre as componentes do *Front End of Innovation* (FEI). Este modelo é composto por três partes principais, como está descrito na Figura 48.

Uma das partes é o motor, parte que dirige os cinco elementos, sendo este abastecido pela liderança, cultura e estratégia da organização. A segunda parte são os fatores de influência, que consistem nas capacidades da empresa, da estratégia de negócios da mesma, o mundo externo (desde os clientes, à concorrência no mercado) e a ciência e tecnologia. A terceira parte são os cinco elementos chave em que as ideias e os conceitos iteram, sendo estas a identificação de

oportunidade, análise de oportunidade, geração de ideias, seleção de ideias e por fim a definição de conceito.

A sua forma circular demostra que ideias e oportunidades estão ligadas entre si, pois, o surgimento de uma oportunidade é uma situação para criar uma ideia ou vice-versa, uma ideia pode levar a que surja uma nova oportunidade. Sendo assim, este modelo possui duas entradas possíveis, tanto uma ideia nova ou uma nova oportunidade que surgiu. Este modelo possui também apenas uma saída, esta só acontece no fim do conceito estar bem definido, sendo este o ponto de ligação com o processo de desenvolvimento [51].

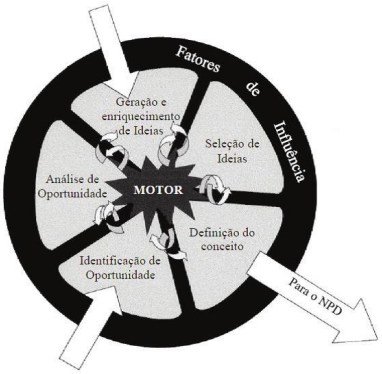

Figura 48 – Modelo NCD – Adaptado de Koen et al., 2002.

Identificação de Oportunidade

A identificação de oportunidades é a procura de falhas ou faltas que possam existir no negócio ou mesmo na tecnologia usada. Esta identificação permite que a empresa ou organização se mantenha competitiva no mercado, sendo que esta pode ser impulsionada pelas metas ou por um potencial competidor que exista no mercado.

Neste caso, e como mencionado no problema (cf. secção XXX), a falta de uma camada de integração faz com que seja difícil a sua adaptação a novos hospitais e consequentemente os custos irão ser maiores. Após esta análise surge então uma nova oportunidade de negócio.

É então identificada uma nova oportunidade de negócio que consiste em desenvolver uma *framework* capaz de realizar uma integração mais eficaz do que a solução existente. Esta necessidade surge também da grande competição existente no mercado e da necessidade de automatizar a integração do software, dado que até a momento era feita uma integração ponto a ponto, sendo esta desatualizada e muito suscetível a falhas.

Análise de Oportunidade

Após a identificação de oportunidades é importante que haja uma análise das mesmas, onde se estuda a probabilidade de sucesso destas tanto a nível de mercado como a nível tecnológico.

Este estudo é feito através do planeamento e avaliação das vantagens que existem, tendo sempre em conta os recursos que a empresa ou organização possuem.

Já tendo sido feita a identificação de oportunidade, sendo então o desenvolvimento de uma nova *framework* de integração é necessário fazer análise de oportunidade. Esta solução exigirá um grande esforço inicial para a sua construção, sendo que tem de ser bem desenvolvida e em termos arquiteturais bem definida.

Uma outra oportunidade é o facto de, seguindo os objetivos definidos no

Geração de ideias

Dado então que tanto a oportunidade encontrada e a solução pensada são vantajosas é necessário passar para a geração de ideias para o desenvolvimento real da solução. Este momento, conhecido também por momento de metamorfose, consiste na transformação das oportunidades em novas ideias de tecnologias. Estas novas ideias vão sempre de encontro à necessidade dos clientes, tendo sempre em conta a oportunidade de adquirir novos clientes, e às possibilidades da empresa.

A geração de ideias, no caso do desenvolvimento de uma *framework* de integração com certas funcionalidades de auditoria aos dados processados como também a transformação da solução para uma mais modular, ocorreu após uma análise aprofundada do problema bem como dos recursos disponíveis, tendo chegado a um leque variado de soluções e tecnologias que podiam ser adaptadas, respondendo tanto às necessidades do cliente bem como às da empresa.

Seleção de ideias

A seleção de ideias, tal como o nome indica, depois de identificadas várias ideias, soluções para a resolução do problema, terão então de ser selecionadas aquela, ou aquelas, que vão ser desenvolvidas. Quando se passa para a fase de desenvolvimento há um acréscimo de responsabilidades, desde o investimento inicial como futuramente a integração da ideia em novos clientes. Dado isto é necessário que haja uma avaliação para garantir retorno, sendo também por vezes necessário uma análise de possíveis concorrentes.

Esta seleção foi efetuada tendo em conta diversos fatores tais como os benefícios diretos, indiretos de uma aplicação mais modular, os requisitos e os custos de implementação. Esta atividade permitiu filtrar de uma forma eficaz as ideias com maior valor associado e que justifiquem a implementação do protótipo.

Os principais fatores da seleção de ideias são a adaptabilidade da solução, o custo de desenvolvimento da mesma e a possibilidade de conseguir seguir certos standards internacionais.

Mas o fator principal para o desenvolvimento deste projeto é então o fator monetário, o facto de ser uma plataforma para gestão de hospitais nacionais implica um forte entrave ao investimento financeiro da empresa em soluções que de um certo modo poderiam aumentar o numero de funcionalidades da aplicação, mas nada que se altera os principais objetivos deste trabalho, tem de se olhar sempre para o facto do custo das licenças da mesma e o que cada licença permite fazer no software de integração.

Esta solução trouxe então a introdução de uma nova tecnologia para a empresa. Tecnologia esta capaz de responder tanto às necessidades dos clientes, uma integração rápida e eficaz, como da empresa, fácil de desenvolver e económica.

Definição de conceito

A definição de conceito é apresentada formalmente onde é evidente as vantagens do produto. É avaliada a viabilidade de introduzir a ideia no processo de produção e quais os requisitos técnicos para o mesmo. Este ocorre quando a ideia se encontra totalmente trabalhada.

Depois de bem definida a ideia esta entra para produção com o desenvolvimento de um protótipo da *framework* de integração. Pretendendo-se que a instalação do software *ePatient* seja mais fácil e rápida.

Valor

O valor de um produto é associado ao custo de fabrico e ao preço que é vendido de forma a haver lucro. Corresponde ao valor sem expectativas exteriores, tanto do comprador como do vendedor. É ainda caracterizado pela razão entre os benefícios e os respetivos custos/sacrifícios.

Tendo em conta o que foi referido no paragrafo acima, o principal valor no desenvolvimento deste projeto é o facto de haver um custo e tempo que é reduzido na implementação da solução no cliente (hospital) como também no desenvolvimento de novas funcionalidades na aplicação.

A.2 Valor percecionado

A criação de valor é fundamental para qualquer negócio, e qualquer atividade comercial é sobe troca de alguns bens ou serviços tangíveis e/ou intangíveis e ter o seu valor aceite e recompensado por clientes, dentro da empresa, da rede colaborativa ou fora desta.

Com esta definição percebe-se que o conceito de valor é algo subjetivo, dado a sua relação com a componente de cada ser humano. No entanto por ser entendido pelo rácio existente entre os sacrifícios e os benefícios existentes na aquisição de algo tangível ou não tangível, como um bem ou um serviço.

O enquadramento com este projeto, a Framework demonstra que o rácio poderá ser positivo dado ao retorno do investimento feito inicialmente para a implementação da mesma.

Tabela 6 – Relação Custo / Benefício

	Produto	Relação
Benefícios	Desenvolvimento de uma solução de integração capaz de integrar sistemas já existentes com o software ePatient	Aumento da satisfação do cliente dado que a implementação do novo software é rápida e prática fazendo com que haja também uma diminuição de erros e perda de dados.
Sacrifícios	Custo de implementação.	É necessário saber à priori com que softwares o ePatient se irá ligar e isso muitas vezes não está bem explícito num hospital devido ao baixo investimento em tecnologia.

A.3 Proposta de Valor

Este projeto promove uma nova forma de vender o produto, já desenvolvido, mais eficientemente, rápida e mais segura.

Esta nova abordagem proporciona ainda, poupança de recursos e sobretudo resultados em benefícios económicos.

A.4 Modelo de Negócio (CANVAS)

Na figura 2 está representado o modelo de negócio utilizando a estrutura Business Model Canvas proposta por Alexander Osterwalde[52].

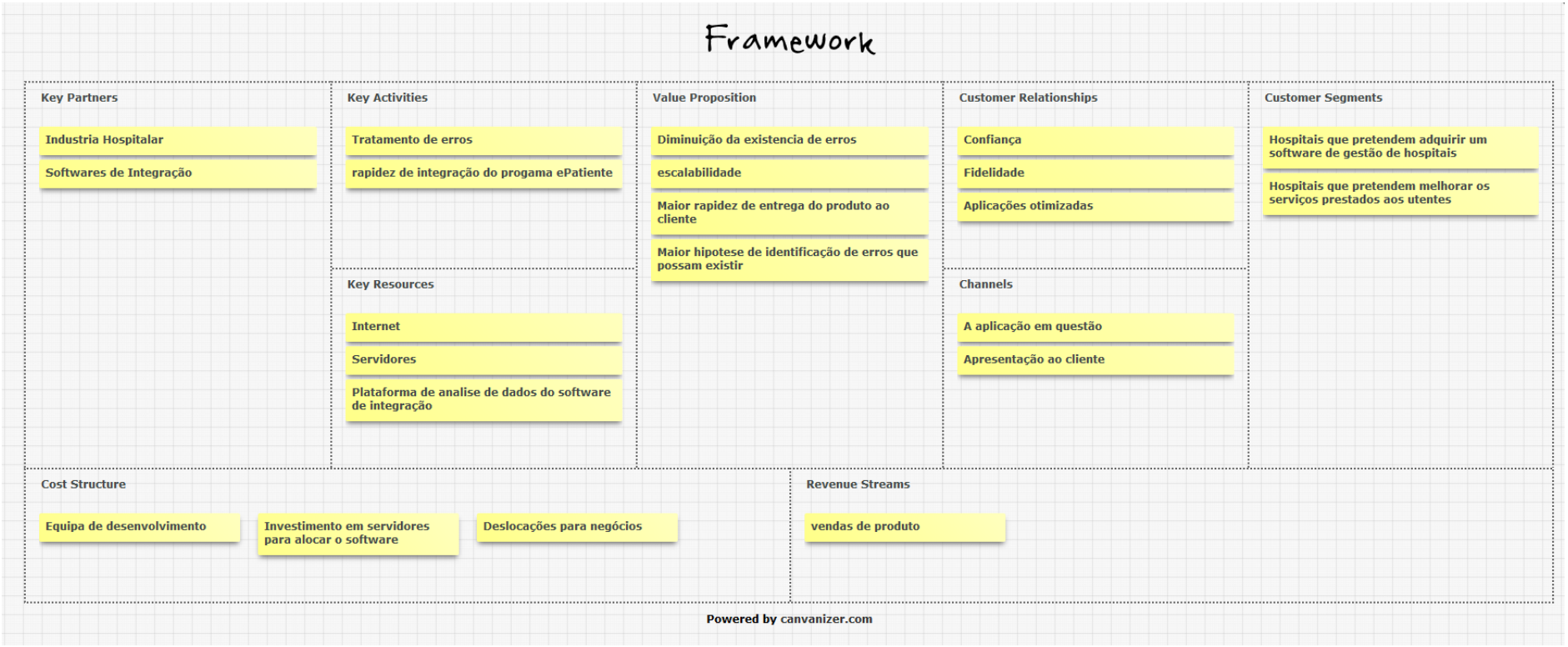

Figura 49 – Modelo Canvas

Anexo B

Web Services

B.4 Staff API

Staff API, como foi descrito anteriormente, ponto 5.3, é a API responsável por coordenar todos os profissionais de saúde, médicos e enfermeiros, e alocá-los a um paciente. É também responsável pelo registo das especializações referentes a cada enfermeiro. Na Tabela 7 estão descritos todos os pedidos possíveis a esta API.

Tabela 7 – Lista de Pedidos Staff API

	Pedido	Descrição
5.4.1	GET/Nurse	Web Service responsável por retornar informação de enfermeiros
5.4.2	PUT/Nurse	Web Service responsável por criar/atualizar registos de enfermeiros
5.4.3	GET/Doctor	Web Service responsável por retornar informação de médicos
5.4.4	PUT/Doctor	Web Service responsável por criar/atualizar registos de médicos
5.4.5	PUT/AttitudeNurse	Web Service responsável por criar/atualizar registos de especialidade de enfermeiros

B.2 Hospital API

Esta API, também referida no ponto 5.3, tem como objetivo tratar das comunicações relativas a assuntos gerais do Hospital, como as especialidades e serviços presentes no hospital. Na Tabela 8 estão descritos todos os pedidos possíveis a esta API.

Tabela 8 – Lista de pedidos Hospital API

	Pedido	Descrição
5.5.1	GET/Speciality	Web Service responsável por retornar informação de especialidades do hospital
5.5.2	PUT/Speciality	Web Service responsável por criar/atualizar especialidades do hospital
5.5.3	GET/Service	Web Service responsável por retornar os serviços do hospital
5.5.4	PUT/Service	Web Service responsável por criar/atualizar serviços do hospital

B.3 Surgery API

Esta API, referida no ponto 5.3, contém todos os serviços necessários para o registo de novas cirurgias no ecossistema ePatient. Esta API é também responsável pela atribuição dos médicos às cirurgias a realizar. Na Tabela 9 estão descritos todos os pedidos possíveis a esta API.

Tabela 9 – Lista de pedidos Surgery API

	Pedido	Descrição
5.6.1	PUT/Surgery	Web Service responsável por criar/atualizar uma determinada cirurgia
5.6.2	GET/Surgery	Web Service responsável por retornar informação de uma determinada cirurgia
5.6.3	PUT/SurgeryDoctor	Web Service responsável por associar um determinado médico a uma cirurgia

B.4 Room API

Esta API, referida no ponto 5.3, tem como objetivo gerir toda a parte de infraestruturas dos hospitais, tanto quartos como camas disponíveis e ocupadas. Na Tabela 10 estão descritos todos os pedidos possíveis a esta API.

Tabela 10 – Lista de pedidos Room API

	Pedido	Descrição
5.7.1	GET/Room	Web Service responsável por retornar informação relativa a quartos do hospital
5.7.2	PUT/Room	Web Service responsável por criar/atualizar um determinado quarto do hospital
5.7.3	GET/Bed	Web Service responsável por retornar informação relativa a camas do hospital
5.7.4	PUT/Bed	Web Service responsável por criar/atualizar uma determinada cama do hospital

B.5 Exams API

Esta API, referida no ponto 5.3, é responsável por conter todos os serviços necessários de avaliação de um paciente internado no hospital, isto é, todos os exames existentes no hospital, como todos os exames realizados por um paciente.

Para além disto, também inclui o registo das medições de parâmetros vitais de um paciente registadas, por norma, pelos enfermeiros. Na Tabela 11 estão descritos todos os pedidos possíveis a esta API.

Tabela 11 – Lista de pedidos Exams API

	Pedido	Descrição
5.8.1	GET/Exams	Web Service responsável por retornar informação relativa a exames
5.8.2	PUT/Exams	Web Service responsável por criar/atualizar exames
5.8.3	GET/Measurements	Web Service responsável por retornar informação relativas à medição de parâmetros vitais de um paciente
5.8.4	PUT/Measurements	Web Service responsável por registar medições de parâmetros vitais de um paciente
5.8.5	GET/MeasurementsTypes	Web Service responsável por retornar informação relativa ao tipos de parâmetros vitais de um paciente
5.8.6	GET/ExamsTypes	Web Service responsável por retornar informação relativa ao tipos de exames possíveis de realizar naquele hospital

B.6 Episode API

Esta API, referida também no ponto 5.3, é talvez a API mais importante de toda a arquitetura desenhada. *Episode* API é responsável por todas as funcionalidades referentes a questões relacionadas com o paciente internado no hospital.

Episode neste contexto é o mesmo que um episódio, isto é, regista todos os acontecimentos desde a entrada à saída do paciente. Acontecimentos como as alergias de um paciente, sejam

estas anteriores ou descobertas no tempo de internamento, a especialidade em que se encontra internado o paciente, todos os diagnósticos apresentados e associados ao paciente, a dieta em que se encontra o paciente, todos os registos administrativos do paciente e se este se encontra ainda internado.

É nesta API que reside a responsabilidade de introduzir um novo paciente que necessite de internamento, como também de atribuir alta ao paciente que não necessite mais de cuidados médicos. É possível verificar se um paciente ainda se mantém internado, como também aceder ao histórico de todo o internamento de um paciente.

Esta API contem também toda a informação relativa aos tipos de admissões suportados pela aplicação e todas as áreas contidas pelo hospital capaz de suportar pacientes. Na Tabela 12 estão descritos todos os pedidos possíveis a esta API.

Tabela 12 – Lista de pedidos Episode API

	Pedido	Descrição
5.9.1	GET/Allergy	Web service responsável por retornar as informações alérgicas referentes a um paciente.
5.9.2	PUT/Allergy	Web service responsável por adicionar/atualizar alergias referentes a um paciente.
5.9.3	GET/AdmissionTypes	Web service responsável por retornar os diferentes tipos de admissões suportados pelo ePatient.
5.9.4	GET/Attitude	Web service responsável por retornar a que especialidade está associado o paciente.
5.9.5	PUT/Attitude	Web service responsável por adicionar/alterar a área referente a um paciente.
5.9.6	GET/AttitudeTypes	Web Service responsável por retornar todas as áreas suportadas pelo hospital referente aos pacientes
5.9.7	GET/Diagnosis	Web Service responsável por devolver informação sobre todos os diagnósticos referentes ao paciente.
5.9.8	PUT/Diagnosis	Web Service responsável por adicionar/atualizar diagnósticos referentes a um paciente.
5.9.8	GET/Diet	Web Service responsável por retornar a dieta associado ao paciente no pedido
5.9.10	PUT/Diet	Web Service responsável por adicionar/atualizar a dieta de um paciente
5.9.11	GET/EpisodeOpenExists	Web Service responsável por informar quem o requisita se um episódio continua aberto ou não, por outras palavras, se o paciente continua internado ou não.
5.9.12	GET/EpisodeHistory	Web Service responsável por devolver informação referente a um determinado episodio de um paciente.
5.9.13	PUT/EpisodeClose	Web Service responsável por atualizar o estado do episódio para fechado, o que significa que um determinado paciente já não se encontra internado no hospital.
5.9.14	PUT/EpisodeOpen	Web Service responsável por introduzir um novo paciente no hospital
5.9.15	GET/Process	Web Service responsável por retornar toda a informação administrativa do paciente
5.9.16	PUT/Process	Web service responsável por adicionar/alterar a informação administrativa do paciente