INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Arquiteturas baseadas em mensagens

BRUNA MOREIRA TEIXEIRA
Julho de 2020

POLITÉCNICO
DO PORTO

Arquiteturas baseadas em mensagens

Bruna Moreira Teixeira

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

Orientador: Nuno Silva
Co-Orientador: António Coelho

Júri:
Presidente:
Alexandre Bragança

Vogais:
Isabel Azevedo

Porto, 5 de Julho de 2020

Dedicatória

A todos os que cruzaram o meu caminho e contribuíram para o que sou hoje.

Resumo

Neste documento é apresentado o trabalho realizado no âmbito da disciplina TMDEI do Mestrado em Engenharia Informática (MEI) do Departamento de Engenharia Informática (DEI) do Instituto Superior de Engenharia do Porto (ISEP) do Politécnico do Porto (P.Porto).

O trabalho descrito neste relatório foi realizado na empresa DevScope e tem como intuito fornecer um conjunto de informações que auxilie os projetos nos quais se pretenda implementar sistemas com arquiteturas baseadas em mensagens.

Ao longo do projeto foi efetuada pesquisa no contexto de *messaging*, incluindo padrões e tecnologias associados.

Para além disso, também se realizou uma prova de conceito do produto SmartDocumentor no âmbito da passagem de sistemas com base de dados partilhada e *polling* para sistemas orientados a mensagens.

Assim sendo, a arquitetura do SmartDocumentor foi redesenhada e generalizada para facilitar o seu uso noutros projetos. De seguida, implementou-se uma solução que contém um componente de comunicação recorrendo a *message bus* genérico que pode ser reutilizado em diversos sistemas de *software*.

Tendo por base os testes de desempenho realizados ao sistema inicial e ao final, conclui-se que o segundo é superior ao primeiro. Apesar de as mensagens oferecerem diversas vantagens, acrescentam complexidade e a sua aplicação deve ser ponderada aquando da alteração de um sistema já existente.

Palavras-chave: Engenharia de Software, Arquitetura, *Event-driven*, *Messaging*

Abstract

At this document is presented the work performed in the scope of the subject TMDEI of the Master in Informatics Engineering (MEI) of the Informatics Engineering Department (DEI) at Porto School of Engineering (ISEP) of the Porto Polytechnic (P.Porto).

The work described at this document was performed at DevScope company and aims to provide a series of information that supports projects in which it is intended to implement systems with message-based architectures.

A research in the scope of messaging was carried throughout the project, including patterns and associated technologies.

Furthermore, it was also done a proof of concept for the product SmartDocumentor in the scope of transitioning from a system with shared database and associated polling to a message oriented system.

Therefore, the SmartDocumentor architecture was redesigned and generalized to ease its application within other projects. This was followed by the implementation of a solution that contains a generic communication component using message bus that can be reused at several other software systems.

Based on the created performance tests of the initial and final systems, it was concluded that the second outstands the first. The use of message offers various advantages, but increases complexity and its application must be conscious when reengineering an existing system.

Keywords: Software Engineering, Architecture, Event-driven, Messaging

Agradecimentos

Agradeço em primeiro lugar à minha família materna, principalmente à minha mãe, aos meus avós e ao meu padrinho, que desde sempre me ajudaram em tudo o que puderam e esforçaram-se para que fosse sempre o mais feliz possível.

Um agradecimento especial a todos os que conheci nesta instituição e que tenho o prazer de chamar de amigo(a), alguns deles permanecerão para sempre no meu coração.

Agradeço também às amizades que fiz ao longo dos anos e que estiveram ao meu lado independentemente da distância e/ou do tempo sem comunicarmos.

Agradeço imenso às pessoas que encontrei na DevScope, empresa na qual tive o gosto de desenvolver este projeto, particularmente ao meu supervisor, António Coelho, ao João Sousa, à Catarina Ricca e a toda a equipa de produto.

Quero agradecer ao meu orientador, Nuno Silva, por toda a paciência e esclarecimento de dúvidas durante o desenvolvimento deste documento.

Agradeço aos restantes que acompanharam o meu percurso académico ao longo da licenciatura e do mestrado, nomeadamente colegas e professores.

Por fim, agradeço a todos os que cruzaram o meu caminho e que direta ou indiretamente influenciaram o meu trajeto.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

μ média

Lista de Acrónimos e Siglas

AHP	Analytic Hierarchy Process.
AV	Análise de valor.
BPO	Business Process Outgoing.
CD	Continuous Delivery.
CI	Continuous Integration.
CRUD	*Create*, *Read*, *Update*, *Delete*.
DEI	Departamento de Engenharia Informática.
FFE	Modelo Fuzzy Front End.
IDE	Ambiente de Desenvolvimento Integrado.
ISEP	Instituto Superior de Engenharia do Porto.
MEI	Mestrado em Engenharia Informática.
NCD	Modelo de desenvolvimento de novo conceito.
NPD	Desenvolvimento de novo produto.
RC	Razão de Consistência.
SaaS	Software as a Service.
SGBD	Sistema de Gestão de Base de Dados.
SOA	Service-oriented architecture.
UI	User Interface.

Capítulo 1

Introdução

No presente capítulo apresenta-se o contexto, problema, objetivo, abordagem preconizada e estrutura do documento.

1.1 Contexto

O SmartDocumentor [1] é uma solução de gestão documental e arquivo com digitalização, captura e reconhecimento inteligente de dados que permite a classificação automática de documentos e interligação com outros sistemas externos.

A partir da digitalização de documentos físicos ou digitais efetua-se:

- O reconhecimento de dados através de diversos processos:
  - OCR (Optical Character Recognition de caracteres impressos);
  - ICR (reconhecimento ótico de texto manuscrito);
  - OMR (reconhecimento ótico de questionários ou inquéritos).
- A validação e integração dos dados (conteúdos) em sistemas externos/terceiros.

O SmartDocumentor promove a recolha de dados de documentos com validação e integração nos sistemas de gestão empresarial (ERP), gestão documental (ECM), gestão de clientes (CRM) ou qualquer outro sistema, através do seu modelo de extensibilidade.

O atual design do sistema é representável pelo diagrama de componentes da figura 1.1.

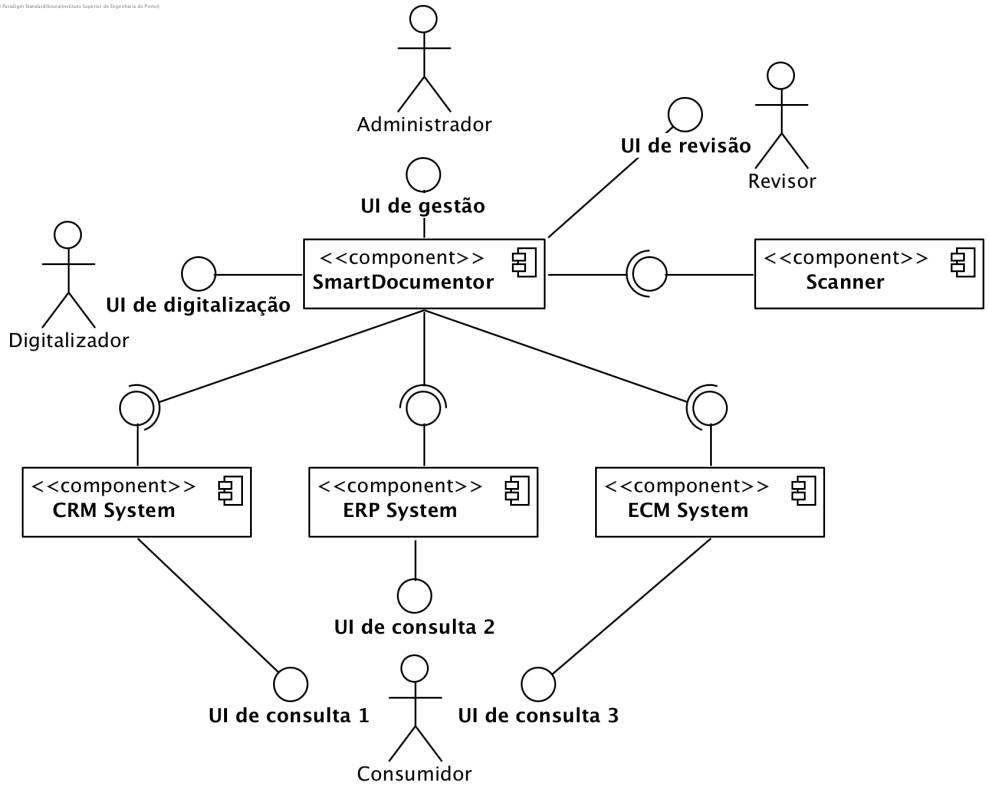

Figura 1.1: Diagrama de componentes do SmartDocumentor

1.2 Problema

Na solução atual do SmartDocumentor é aplicado o padrão *shared database*. Apesar de este padrão ser em muitas circunstâncias parte da melhor solução, a sua utilização pode ser conjugada com estratégias referidas no livro "Enterprise Integration Patterns" [2].

No *software* é utilizado o método baseado em *polling*[1] no qual os componentes contactam a base de dados de 30 em 30 segundos de forma a verificar se existem dados com o estado pretendido quer haja ou não alterações na mesma ou itens que correspondam à pesquisa. No pior dos casos, cada um dos serviços pode atrasar as suas operações em 30 segundos, o que atualmente não é considerado um problema de desempenho com implicações no negócio. Caso fosse necessário, isto poderia ser resolvido diminuindo o valor do intervalo definido, mas com a eventual necessidade de escalabilidade do sistema, a quantidade de dados trocados aumenta e, por conseguinte, o *software* pode tornar-se mais lento e desperdiçar recursos.

Esta reengenharia também advém do facto de a empresa saber que a arquitetura atual do sistema não é a mais adequada devido ao uso de base de dados partilhada com *polling*, nomeadamente dado que a quantidade de tarefas de digitalização e revisão varia muito ao longo do tempo e entre clientes.

Efetivamente, embora alguns sistemas de *software* sejam projetados de modo a haver a existência de respostas imediatas e permitirem escalabilidade, por vezes estes requisitos não são cumpridos do modo mais adequado. De facto, o desenho destes sistemas de forma

[1] Técnica em que o componente interroga continuamente outro acerca de dados a ser partilhados [3].

eficiente e escalável pode ser um desafio, dependendo da complexidade da comunicação dos serviços que compõem o *software*.

1.3 Objetivo

O objetivo consiste em passar um sistema de *software* em que existe comunicação indireta entre os vários componentes através da base de dados para uma solução orientada a mensagens.

Assim sendo, o propósito traduz-se por realizar uma reengenharia, redesenhando a arquitetura atual do sistema.

Para tal, devem ser seguidos os passos:

- Fazer levantamento e análise de modos de interligação de sistemas baseados em mensagens, nomeadamente:
  - Conceitos relevantes para o projeto associados a integração de sistemas e respetiva comunicação entre componentes;
  - Tecnologias relacionadas com a solução a desenvolver no âmbito de *messaging*.
- Análise da implantação de soluções distribuídas, particularmente:
  - Diferentes tipos de implantação;
  - Ferramentas utilizadas na implantação de soluções distribuídas.
- Desenvolver caso concreto sob a forma de prova de conceito, adicionando ao SmartDocumentor comunicação entre componentes através de técnicas de *messaging*. Nesta prova de conceito deve existir a possibilidade de alterar a escolha tecnológica. Para além disso, a integração e entrega contínuas do sistema devem ser asseguradas e as exceções que possam ocorrer tratadas.
- Efetuar comparação:
  - Entre as escolhas tecnológicas;
  - Entre os sistemas inicial e final.

No processo de reengenharia do sistema de *software*, deve-se ainda considerar outros requisitos:

- Monitorização, para verificar o fluxo de mensagens na comunicação entre componentes;
- Implantação na nuvem, nomeadamente a possibilidade de diferentes implantações adaptadas à instalação;
- Desempenho, no sentido em que o *software* deve ser otimizado em termos de tempo de resposta e tempo de trabalho;
- Flexibilidade, o sistema deve ser adaptável a alterações de requisitos sem abranger modificações a nível de estrutura;
- Escalabilidade, pois o *software* deve ter a capacidade de crescer conforme as necessidades do utilizador;

- Modularidade do sistema, dado que o mesmo deve estar dividido em unidades distintas e independentes responsáveis por determinado aspeto do programa, promovendo o desacoplamento e coesão das partes e consequente manutenibilidade.

Deste modo, o caso concreto do SmartDocumentor da DevScope servirá como prova de conceito no âmbito da construção de arquiteturas orientadas a mensagens, dado que o intuito da empresa traduz-se em aplicar esta solução em diversos sistemas.

1.4 Abordagem preconizada

Existem vários tipos de arquitetura que podem ser adotados de forma a cumprir os requisitos de uma solução como a preconizada na secção anterior. Assim sendo, serão pesquisados e analisados abordagens, estilos e padrões aplicáveis ao produto e à dimensão do mesmo.

Atualmente, os modelos de *messaging* desempenham um papel importante no desenvolvimento de soluções de *software*, uma vez que correspondem a uma forma de comunicação direta entre sistemas, permitindo o aumento da confiabilidade. Os padrões de *messaging* correspondem também a uma boa forma de diminuir o acoplamento que, por sua vez, fornece benefícios como flexibilidade e mais fácil escalabilidade [2].

De facto, é frequente uma aplicação baseada na nuvem encontrar-se dividida em serviços mais pequenos que trabalham juntos de forma a processar uma transação de negócio do início ao fim. Isto é realizado com o intuito de diminuir o acoplamento entre os serviços, ficando cada um dos mesmos responsável por uma única operação de negócio. Posto isto, é possível desenvolver aplicações mais rápido e como a base do código é mais pequena a sua escalabilidade também se torna mais simples [4].

Todavia, a adoção desta arquitetura apresenta a desvantagem de ser mais complexa, dado o elevado grau de distribuição e baixo acoplamento. Assim, caso não se tenha um conhecimento pleno do sistema, há a possibilidade de, no momento de desenvolvimento/alteração do mesmo, que a adição de um único evento desencadeie uma sequência imprevista de eventos [5].

Após a realização da prova de conceito espera-se que a solução funcione enquanto inspiração e/ou diretriz para a adoção de arquiteturas orientadas a mensagens para comunicação entre componentes noutros sistemas, existentes ou futuros.

1.5 Estrutura do documento

No presente capítulo, **Introdução**, é exposto o necessário à compreensão inicial do documento, nomeadamente a respetiva estrutura.

O segundo capítulo, **Estado da arte**, tem como conteúdo os conceitos e o estado da arte em tecnologia relevante.

No capítulo 3, **Análise de negócio**, encontram-se apresentados os detalhes necessários ao entendimento do tema do projeto, os processos e intervenientes e as restrições existentes.

No quarto capítulo, **Análise de valor**, foi realizada a análise de valor do projeto descrito neste relatório tendo em conta o processo respetivo.

No que diz respeito ao capítulo 5, **Desenho da solução**, o seu intuito centra-se na denotação das abordagens alternativas e do desenho arquitetural, incluindo as vistas mais relevantes e a sua generalização.

O sexto capítulo, **Construção da solução**, foca-se na apresentação da construção da solução para o problema descrito, incluindo os detalhes da implementação e da prova de conceito.

No capítulo 7, **Experiências e Avaliação**, realiza-se a comparação entre diferentes soluções segundo as fontes de informação definidas.

O oitavo capítulo, **Conclusões**, tem como conteúdo os objetivos alcançados, as limitações no decorrer do projeto e respetivo trabalho futuro.

Por fim, são apresentadas as **Referências** e **Anexos** referentes ao documento presente.

Capítulo 2

Estado da arte

Neste capítulo são apresentados os conceitos, assim como o estado da arte em tecnologia relevante.

2.1 Conceitos

Aquando na presença de uma aplicação que consiste em diversos componentes a correr em diferentes computadores, servidores ou dispositivos móveis, a comunicação entre os mesmos pode tornar-se difícil e de baixa confiabilidade [6].

Atualmente, um número considerável de sistemas são compostos por múltiplas aplicações construídas de modo independente recorrendo a diferentes linguagens e plataformas cuja informação necessita de ser trocada eficaz e eficientemente. Ainda que na presença do mesmo servidor ou *data center*, as arquiteturas de baixo acoplamento requerem mecanismos que permitam aos vários componentes comunicar, sendo que a confiabilidade deste *messaging* é frequentemente um problema crítico [6]. Assim sendo, deve ser construída uma infraestrutura robusta de comunicação de forma a assegurar que a aplicação possui confiabilidade e escalabilidade.

2.1.1 Estilos de integração de sistemas

De seguida serão apresentados os dois principais estilos adotados para este género de problemas. O primeiro corresponde ao utilizado atualmente em aplicações como o SmartDocumentor e o segundo ao que é pretendido alcançar.

2.1.1.1 Shared database

No caso do padrão *shared database* os componentes são integrados através do uso de uma base de dados partilhada na qual todos os componentes armazenam os dados respetivos, tal como se pode observar na figura 2.1 [2], [7].

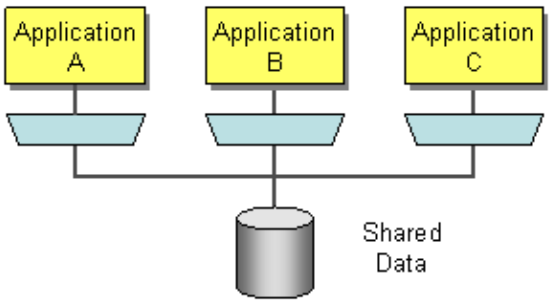

Figura 2.1: *Shared database* (EIP)
Fonte: [2]

2.1.1.2 Messaging

O *messaging* é utilizado para transferir pacotes de dados de um modo imediato, confiável e assíncrono. Desta forma, o envio de uma mensagem não requere que os componentes estejam funcionais e preparados ao mesmo tempo, havendo, assim, a possibilidade de aumentar a coesão e diminuir a adesão. Na figura 2.2 encontra-se representado o exemplo de uma forma de utilizar *messaging* [2].

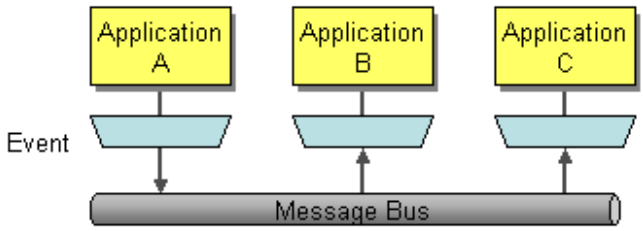

Figura 2.2: *Messaging* (EIP)
Fonte: [2]

Uma mensagem corresponde a um pacote de dados que são transmitidos do *sender* para o *receiver*, tal como se pode observar na figura 2.3 [2].

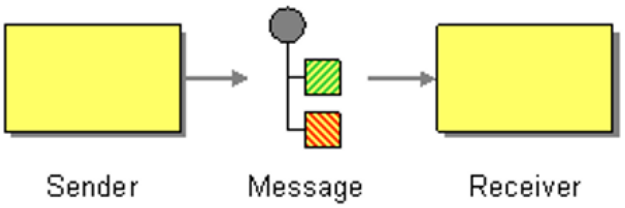

Figura 2.3: Mensagem (EIP)
Fonte: [2]

Ambos, comandos e eventos, são enviados por meio de mensagens. No entanto, diferem um do outro, dado que o primeiro corresponde ao que se pretende que outra aplicação realize e o segundo ao que aconteceu aos dados da aplicação que o emite [8].

Os componentes podem ser distinguidos entre [5]:

- *Sender/Publisher* - componente que produz e envia a mensagem/envia um evento;
- *Receiver/Subscriber* - componente à escuta que processa o conteúdo enviado/subscreve aos eventos com os quais deseja comunicar.

As mensagens podem ser transmitidas de forma [5]:

- Direta - pelo uso do protocolo HTTP, entre outros;
- Indireta - através de sistemas de intermediação, nomeadamente *message brokers*.

Por outro lado, tanto uma mensagem direta como indireta, dependendo da forma como o emissor pretende lidar com a resposta, pode ser: síncrona (fica à espera) ou assíncrona (não fica à espera) [5].

No mundo real, uma única aplicação pode utilizar comandos para umas tarefas e eventos para as demais. De forma a escolher a arquitetura mais adequada, os casos de uso da aplicação devem ser previamente analisados para que sejam identificadas todas as ocorrências nas quais os componentes necessitam de comunicar entre si [9].

O *messaging* envolve uma série de definições básicas, correspondentes a padrões EIP, que uma vez percebidos facilitam a construção da arquitetura do sistema baseado em mensagens.

As aplicações que utilizam arquiteturas baseadas em mensagens transmitem os dados através de um *message channel* que liga o *sender* ao *receiver* do modo que se encontra representado na figura 2.4 [2].

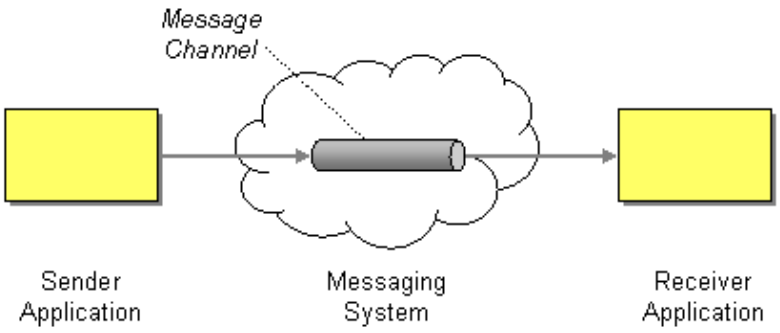

Figura 2.4: Message channel (EIP)
Fonte: [2]

Existem diferentes tipos de *message channels* [2]:

- `Message Bus` – estrutura o *middleware* entre aplicações, possibilitando que trabalhem juntas com recurso ao *messaging*;
- `Point-to-Point channel` – assegura que apenas um *receiver* consome dada mensagem, mesmo que o *channel* tenha vários *receivers*. Neste âmbito há a possibilidade de empregar, nomeadamente, os padrões:
  - Consumidores concorrentes – No cenário deste padrão, o *sender* envia mensagens que serão respetivamente consumidas. No entanto, neste caso as mensagens são

enviadas para um único *channel* e apenas um *receiver* vai consumir dada mensagem. Por conseguinte, os *consumers* podem processar múltiplas mensagens de forma concorrente [10], [11].

Aquando do uso do padrão consumidores concorrentes, pode ser aplicada uma orquestração ou uma coreografia.

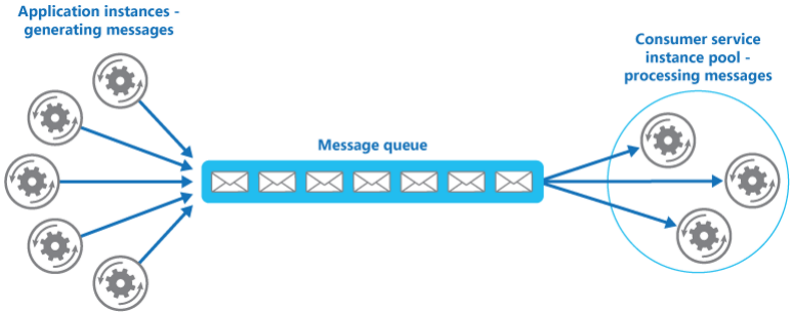

Figura 2.5: Consumidores concorrentes
Fonte: [11]

– Queue-Based Load Leveling Pattern – Este padrão é utilizado aquando na presença de aplicações compostas por serviços que podem ser sobrecarregados. No entanto, o mesmo não é recomendado caso a aplicação espere respostas do serviço com o mínimo de latência [12].

As arquiteturas baseadas neste padrão possuem uma *message queue*, para a qual as tarefas são enviadas e consequentemente armazenadas, de forma a que de seguida sejam consumidas por dado serviço [12]. Um exemplo de como este padrão funciona pode ser observado na figura 2.6.

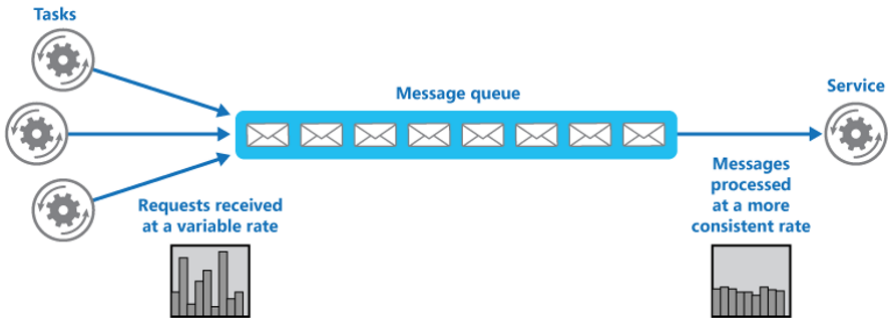

Figura 2.6: Queue-Based Load Leveling Pattern
Fonte: [12]

- `Publish-Subscribe channel` – entrega uma cópia do evento a cada um dos *receivers*. No cenário deste padrão, o *publisher* envia uma cópia de dada mensagem a cada um dos *subscribers*. Assim sendo, difere do padrão consumidores concorrentes pelo facto de todos os *receivers* receberem determinada mensagem e não apenas um dos mesmos. Aquando da implementação deste género de solução, a mensagem passa pelo *message channel* e é distribuída pelos diversos *output subscribers*, tal como apresentado na figura 2.7 [13].

Quando se utiliza este padrão, decide-se empregar uma orquestração ou uma coreografia.

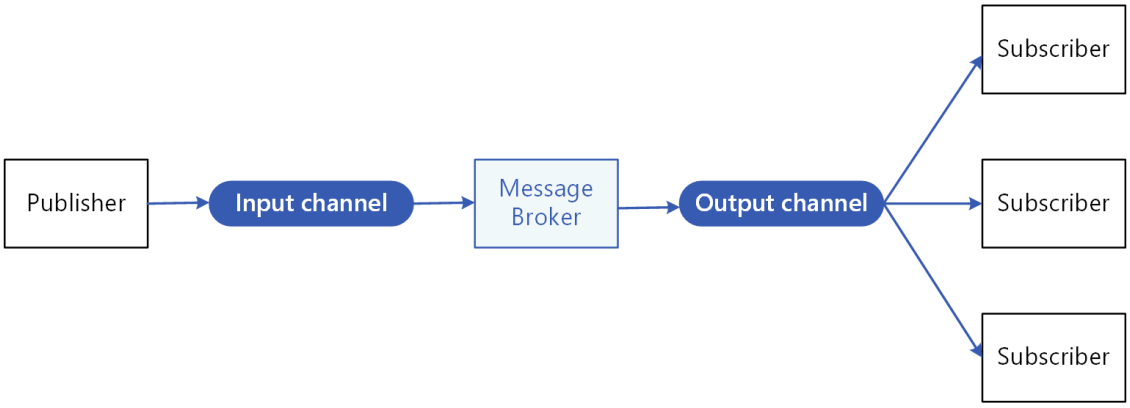

Figura 2.7: Publisher/Subscriber
Fonte: [13]

- `Guaranteed Delivery` – garante que as mensagens são persistidas para que não sejam perdidas mesmo que o sistema de *messaging* falhe.

2.1.1.2.1 Estilos de controlo distribuído

De modo a auxiliar aplicações na nuvem que se encontram divididas em vários serviços menores existem padrões, nomeadamente os apresentados nesta subsecção, que auxiliam o desenho do sistema e conseguinte implementação.

2.1.1.2.1.1 Orquestração

No cenário de uma orquestração há um serviço que controla a sequência de operação das várias partes para a realização das tarefas de negócio. Desta forma, do mesmo modo que se pode observar na figura 2.8, o orquestrador recebe todos os pedidos e reencaminha para o componente respetivo, gerindo todo o processo de negócio do sistema, ou seja, cada um dos serviços não tem conhecimento do *workflow* geral. A desvantagem deste padrão é que caso seja necessário acrescentar ou remover serviços, a lógica do modelo de negócio tem de ser repensada e implementada no orquestrador, o que pode ser complexo e difícil de manter. Todavia, existe a vantagem de esta alteração acontecer apenas no orquestrador [4], [14].

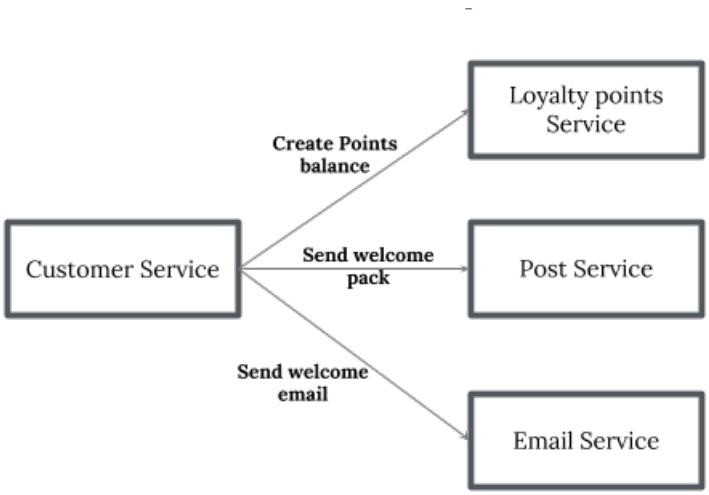

Figura 2.8: Orquestração
Fonte: [15]

2.1.1.2.1.2 Coreografia

Numa coreografia, por oposto a uma orquestração, os serviços têm a possibilidade de comunicar diretamente uns com os outros, existindo descentralização. Consequentemente, cada serviço pode decidir quando e como a operação de negócio é processada, em detrimento de depender de um orquestrador, descrito na figura 2.9. Podem ser utilizados vários padrões neste sentido, particularmente o que inclui o recurso a *message bus*.

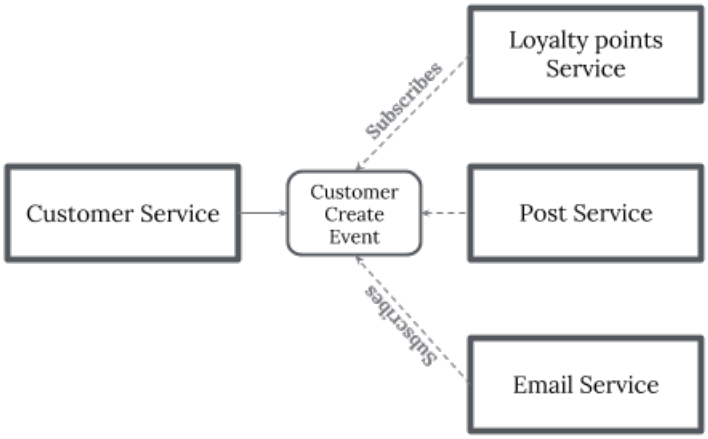

Figura 2.9: Coreografia
Fonte: [15]

2.1.1.2.2 Estilos de arquitetura

Seguidamente, são apresentados vários estilos de arquitetura que podem, respetivamente, adotar um conjunto ou subconjunto de padrões. De facto, estes padrões podem ser utilizados simultaneamente numa mesma arquitetura.

2.1.1.2.2.1 Event-driven

Nesta arquitetura, existem componentes que geram os dados e outros que ficam à escuta dos eventos. Deste modo, uma aplicação pode consumir mensagens assim que ficam disponíveis no *channel* [16]. A arquitetura *event-driven* pode ser utilizada juntamente com microsserviços, descrito na figura 2.10.

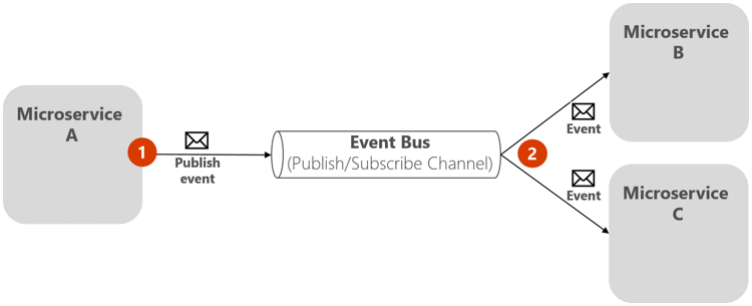

Figura 2.10: Exemplo de uso de *event-driven*
Fonte: [16]

Um evento é uma notificação que indica que algo acontece, sendo transformados em mensagens que são enviadas entre partes do sistema. A forma mais comum de lidar com eventos é através do uso da arquitetura publish-subscribe (pub/sub). Neste cenário a subscrição é gerida por um intermediário que aquando do envio de um evento por parte de um *publisher* reencaminha o mesmo para os *subscribers* interessados. Um evento pode estar ou não relacionado com outros, sendo que no primeiro caso poderá pertencer a uma sequência ordenada, tal como representado na figura 2.11 [9].

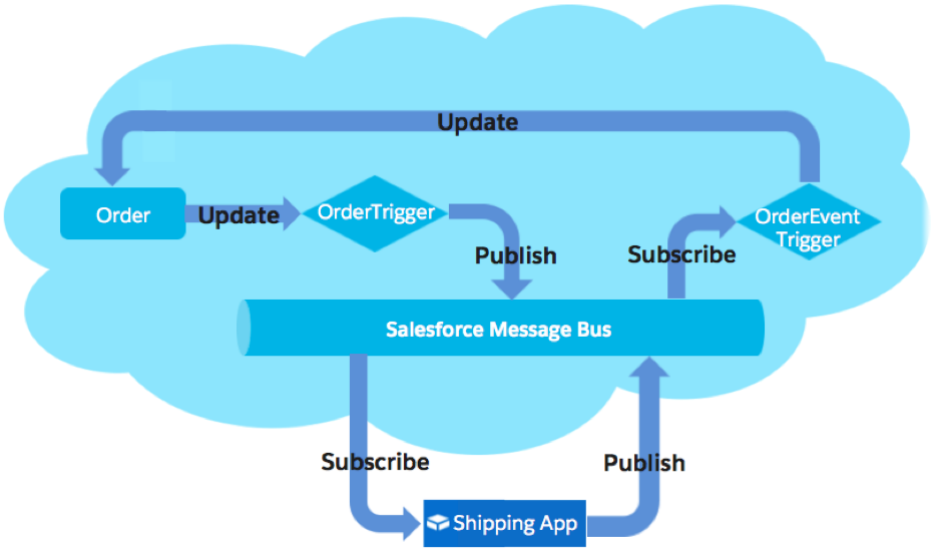

Figura 2.11: Sequência ordenada de eventos
Fonte: [17]

2.1.1.2.2.2 Microsserviços

Microsserviços é um estilo arquitetural que estrutura uma aplicação como uma coleção de pequenos serviços autónomos e modelados em torno de um domínio de negócio. Neste tipo de arquitetura, cada serviço é independente e implementa um único recurso comercial e respetiva base de dados. Desta forma, o serviço correspondente manipula os seus próprios dados e executa diferentes funcionalidades.

O objetivo do uso de microsserviços é acelerar o desenvolvimento de *software*, permitindo o *continuous delivery/deployment*. Este estilo arquitetural decompõe uma aplicação inteira num conjunto de serviços que podem ser implementados e dimensionados de forma independente [18], [19].

Neste cenário, pode-se implementar, a título de exemplo, comunicação assíncrona orientada a eventos, nomeadamente com o auxílio de um *message broker*, o que se encontra representado na figura 2.12.

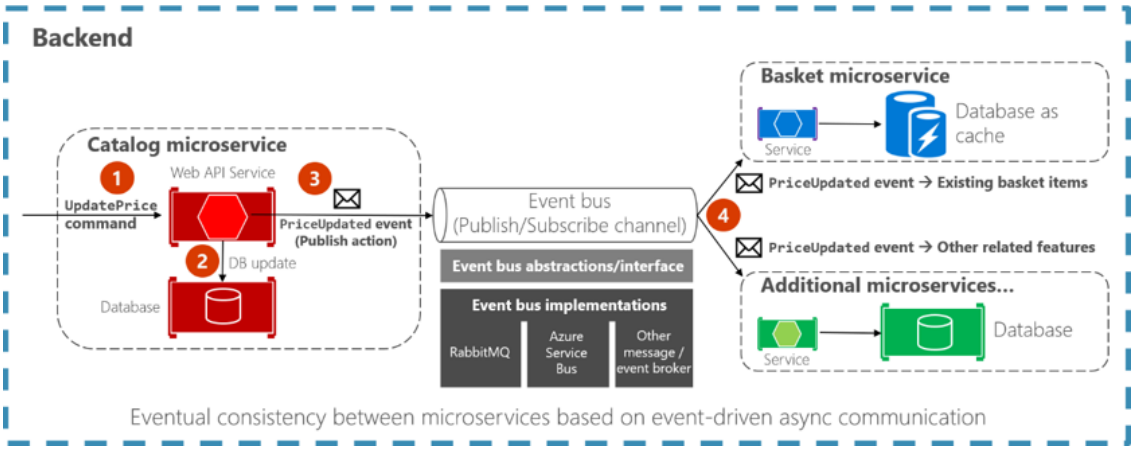

Figura 2.12: Exemplo do uso de microsserviços
Fonte: [18]

2.1.1.2.2.3 Arquitetura em camadas

O padrão de arquitetura em camadas é um dos mais utilizados no cenário de desenvolvimento de *software*.

Nesta arquitetura, cada parte independente de *software* possui determinada função no sistema [20].

Apesar deste padrão não possuir uma quantidade de tipos específicos de camadas, um exemplo é apresentado na figura 2.13. Portanto, podem existir mais ou menos camadas consoante a complexidade da aplicação [20].

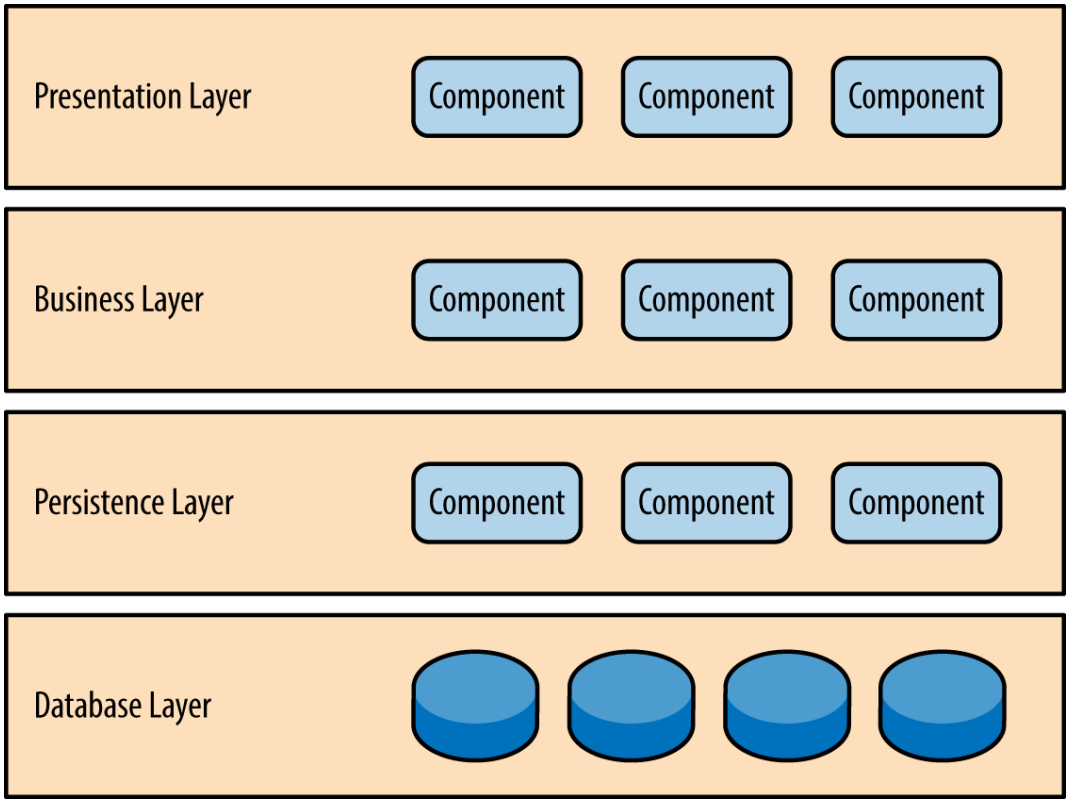

Figura 2.13: Exemplo de arquitetura em camadas
Fonte: [20]

2.1.2 Implantação de soluções distribuídas

Atualmente, a arquitetura de aplicações é importante, uma vez que facilita o conhecimento do sistema em questão e permite que o mesmo seja maximizado no sentido de escalabilidade, resiliência e disponibilidade.

A lógica *serverless* consiste em correr o código da aplicação sem a necessidade de criar, configurar e manter um servidor. O cenário principal corresponde à divisão da aplicação em várias funções que são desencadeadas aquando de um acontecimento [21].

Atualmente, existem diferentes tipos de soluções de integração de sistemas:

- **Onpremises** – fazem uso de servidores *onpremises* de modo a interligar várias aplicações;
- **Nuvem** – usam serviços disponíveis na nuvem com recurso, por exemplo, ao Azure, ao WSO2 Cloud e ao IBM Cloud.
- **Híbridas** – género de solução que utiliza ambas as anteriores [22].

De facto, a nuvem está a alterar o modo como as aplicações são desenhadas, facilitando a posterior manutenibilidade, pois em vez de aplicações monolíticas as mesmas são decompostas por menores serviços descentralizados, isto é, o acoplamento é diminuído. Assim sendo, estes serviços têm a necessidade de comunicar de modo assíncrono através de mensagens ou eventos. Por conseguinte, e tendo em conta a existência de escalabilidade horizontal, novas instâncias podem ser criadas conforme sejam precisas.

No entanto, existem aspetos que devem ser ponderados aquando do uso da nuvem, de entre estes destacam-se: a aplicação ser distribuída, as operações serem realizadas em paralelo e de modo assíncrono, o sistema ter de ser resiliente em caso de falha, as implantações serem automatizadas e previsíveis e monitorizar de modo a ter uma visão geral do sistema [23].

Uma comparação mais específica entre os tipos de soluções de integração de sistemas encontra-se apresentada na subsecção A.1.1 do anexo A.

2.2 Estado da arte em tecnologia relevante

Nesta secção serão apresentadas várias tecnologias utilizadas para *messaging*, sendo que podem ser utilizadas em soluções *onpremise*, na nuvem ou híbridas.

2.2.1 Protocolos de messaging

Atualmente, há dois géneros de mecanismos utilizados para a troca de mensagens: síncronos e assíncronos. No entanto, com o crescente uso de microsserviços, o modelo assíncrono é o mais utilizado para o auxílio da construção, desempenhando um papel fulcral na comunicação entre os mesmos [24].

Existem diversos protocolos que preenchem as necessidades básicas do *messaging*, porém de entre os mesmos destacam-se o AMQP e o MQTT. Ambos os protocolos são *open-source*, utilizados em *asynchronous message queueing* e de simples configuração [25].

No entanto, cada um dos protocolos é melhor em determinados aspetos do que o outro, tal como se pode observar na tabela 2.1 com os critérios [24]:

1. Ligação pub/sub, sendo possível a implementação do padrão publisher-subscriber;
2. Outras formas de *messaging*, incluindo a utilização do padrão consumidores concorrentes;
3. Persistência, permitindo que um servidor redirecione para um mútuo, a nível de toda a ligação ou apenas relativamente dada queue;
4. Transações, possibilitando a coordenação do resultado das diversas transferências de informação;
5. Maior facilidade de implementação, considerando a complexidade do protocolo.

Tabela 2.1: AMQP vs MQTT

Protocolo	Critério 1	Critério 2	Critério 3	Critério 4	Critério 5
AMQP	✔	✔	✔	✔	✖
MQTT	✔	✖	✖	✖	✔

Ambos os protocolos são compatíveis com a maioria do *software*, podendo ser utilizados nos principais sistemas operativos e plataformas. Contudo, o AMQP é superior em termos de adequação a diferentes casos, segurança, confiabilidade e aproveitamento de recursos [24].

Posto isto e tendo em conta a informação sistematizada na tabela anterior, pode-se concluir que o protocolo AMQP é superior ao MQTT.

2.2.2 Tecnologias para comunicação entre componentes

Nesta subseção são inumeradas algumas das tecnologias utilizadas em sistemas distribuídos que suportam o conceito de *messaging*. Neste âmbito, foram consideradas tecnologias que utilizam os padrões consumidores concorrentes, publish-subscribe ou ambos.

2.2.2.1 Amazon Simple Queue Service (SQS)

O Amazon Simple Queue Service (SQS) é um serviço de *message queuing* que auxilia o desacoplamento e escalabilidade de microsserviços, sistemas distribuídos e aplicações *serverless*. Esta tecnologia possui as vantagens referidas, uma vez que permite o envio, o armazenamento e a receção de mensagens entre componentes de *software* [26].

2.2.2.2 Apache ActiveMQ

O Apache ActiveMQ é um servidor de *messaging open-source* baseado em Java que suporta diversos protocolos para ser utilizado em diversas situações [27].

Esta tecnologia utiliza a API do Java Message Service (JMS). O JMS separa os tipos de arquitetura de mensagens em dois: Point-to-Point (P2P) *messaging* e Publish-and-Subscribe (Pub/Sub) *messaging*. No primeiro caso, é utilizada uma queue para a qual o *publisher* envia as mensagens e um dos consumidores à espera retira a mesma da queue, ou seja, o princípio corresponde ao padrão de consumidores concorrentes. No segundo cenário, o *broker* possui tópicos e caso a mensagem seja envia para um dos mesmos todos os consumidores à escuta irão receber a respetiva informação [28].

Apesar de o ActiveMQ ser um *message broker* implementado *onpremise*, pode ser configurado e operado na nuvem recorrendo ao serviço Amazon MQ.

2.2.2.3 Apache Kafka

O Kafka [29] é uma plataforma de transmissão distribuída e possui as seguintes funcionalidades:

- `Publish-subscribe` – leitura e escrita de fluxos de dados, tal como um sistema de mensagens;
- `Processamento` – escrita de fluxo de aplicações de processamento escalável que reage a eventos em tempo real;
- `Armazenamento` – guarda fluxos de dados de um modo seguro num *cluster* distribuído, replicável e tolerante a falhas.

A tipologia baseada em *logs* do Apache Kafka difere das que utilizam queues.

Aquando do uso de *logs*, os tópicos são estruturas de dados físicas partilhadas entre *producers* e *consumers*. Isto é, *producers* e *consumers* ficam acoplados a dado tópico, o que torna a construção de tópicos difícil de alterar após a sua implementação. Por outro lado, esta tecnologia permite uma melhor escalabilidade do sistema, assim como a garantia de ordenação de eventos [30].

2.2.2.4 Azure Queue Storage

O uso de Azure Queue Storage é adequado aquando na presença do padrão de consumidores concorrentes. No entanto, as aplicações que pretendem receber os dados não se encontram à escuta, simplesmente recorrem ao *polling* de forma a verificar a queue respetiva [31].

2.2.2.5 Azure Service Bus

O Azure Service Bus é um message bus que foi criado para ser utilizado em aplicações empresariais e pode recorrer ao uso de queues (local para guardar mensagens temporárias), tópicos (semelhante a uma queue, mas pode ter múltiplos subscritores) ou *relays* (objeto que realiza comunicação assíncrona entre aplicações) [32].

2.2.2.6 Azure Event Grid

O Azure Event Grid distribui eventos de diferentes origens e foi criado para facilitar o desenvolvimento de aplicações *serverless* e/ou baseadas em eventos. Assim sendo, pode ser utilizado como serviço intermediário nos casos em que existem *publishers* e *subscribers*, uma vez que ajuda relativamente à notificação de que algo aconteceu ou que algum objeto foi modificado [33].

2.2.2.7 Azure Event Hubs

Esta tecnologia é adequada a aplicações que produzem uma grande quantidade de eventos de imensas fontes, sendo associado às situações que requerem o uso de estruturas específicas para as manusear. A ferramenta Azure Event Hubs pode ser usada para implementar o padrão publish-subscribe [34].

2.2.2.8 Google Cloud Pub/Sub

Tal como indica o nome, esta tecnologia tem por base o padrão publish-subscribe, podendo ser utilizada para aumentar a flexibilidade e confiabilidade do *middleware* orientado a mensagens situado na nuvem. O Google Cloud Pub/Sub permite o uso de *messaging* nas aplicações pretendidas e ajuda na diminuição do acoplamento entre *senders* e *receivers* [35].

2.2.2.9 RabbitMQ

O RabbitMQ [36] corresponde ao message broker *open-source* mais utilizado devido aos seguintes motivos:

- Leveza e facilidade de implantação onpremise ou na nuvem;
- Suporta múltiplos protocolos de *messaging*;
- Pode ser implantado num sistema distribuído e configurado de modo a ser adequado a requisitos de grande escala e disponibilidade;
- Corre em diferentes sistemas operativos e plataformas na nuvem pois fornece uma grande quantidade de ferramentas para as linguagens mais populares.

2.2.2.10 Redis

Redis [37] é uma estrutura que pode ser usada como base de dados, *cache* ou *message broker*.

Assim sendo, há a possibilidade de implementar o padrão publish-subscribe recorrendo a esta tecnologia que fornece os seguintes comandos:

- `Publish` – publica para determinado *channel*;
- `Subscribe` – subscreve a dado *channel*;
- `Unsubscribe` – anula a subscrição ao *channel*;
- `Psubscrive` – subscreve ao(s) *channel*(s) que obedece(m) ao parâmetro definido;
- `Punsubscribe` – cancela a subscrição ao(s) *channel*(s) cujo padrão corresponde ao estipulado [38].

O Redis pode ser utilizado *onpremise* ou na nuvem e é adequado aos cenários em que se pretende ter um sistema em que se utiliza *messaging*, permitindo a interação entre diversas aplicações.

2.2.2.11 SignalR

O SignalR é normalmente usado para interligar componentes no sentido em que quando um emite dada alteração o(s) restante(s) recebe(m) respetiva notificação.

Esta solução é adequada no caso de sistemas que utilizam *polling*, uma vez que é possível efetuar uma configuração na qual o SignalR notifica os clientes aquando da alteração de dado componente.

No entanto, perante o cenário de um sistema que utiliza eventos de integração para haver consistência, apesar de ser possível, não se deve utilizar uma abordagem que simule os mesmos, tal como o caso do SignalR ou de sistemas de *polling* por parte do cliente. Assim sendo, o SignalR não é adequado à comunicação assíncrona de eventos com múltiplos *receivers* [39].

2.2.2.12 ZeroMQ

A tecnologia ZeroMQ, ao contrário da maioria das apresentadas, não corresponde a um servidor central, apenas a uma biblioteca que pode ser utilizada nas aplicações. Esta biblioteca comporta-se como uma *framework* de concorrência, pois fornece *sockets*[1] que transportam as mensagens [41].

[1] *Socket* corresponde a um endpoint de comunicação bidirecional entre dois programas que correm em dada rede. A uma *socket* está associado um número de porta de modo a que a camada TCP identifique a aplicação para a qual os dados devem ser enviados [40].

A comparação das diversas tecnologias encontra-se representada na tabela 2.2, tendo por base os critérios:

1. Possível uso do padrão consumidores concorrentes com recurso a uma queue
2. Possível uso do padrão Pub/Sub, podendo um *publisher* emitir um evento para todos os *receivers*
3. *Open-source*, não havendo custos relacionados com implantação nos casos mais básicos
4. Adequada a sistemas empresariais integrados na nuvem
5. *Onpremise*, sendo possível o seu uso num ambiente sem acesso à nuvem

Tabela 2.2: Comparação entre tecnologias

Tecnologia	Critério 1	Critério 2	Critério 3	Critério 4	Critério 5
AmazonSQS	✔	✔	✖	✔	✖
ActiveMQ	✔	✔	✔	✖[a]	✔
Apache Kafka	✔	✔	✔	✔[b]	✔
Azure Queue Storage	✔	✖	✖	✔	✖
Azure Service Bus	✔	✔	✖	✔	✖
Azure Event Grid	✖	✔	✖	✔	✖
Azure Event Hubs	✖	✔	✖	✔	✖
Google Cloud Pub/Sub	✔	✔	✖	✔	✖
RabbitMQ	✔	✔	✔	✔	✔
Redis	✔	✔	✔	✔[b]	✔
SignalR	✖	✔	✔	✔[b]	✖
ZeroMQ	✔	✔	✔	✔[b]	✔

[a] Apesar de não se conseguir configurar na nuvem sozinho, pode-se usar a ferramenta AmazonMQ (não é *open-source*) como serviço para a implantação do *message broker* ActiveMQ.

[b] Pode ser usado implantado na nuvem, usando plataformas como o AWS ou o Azure, porém é necessário possuir subscrição.

2.2.3 Ferramentas utilizadas na implantação de soluções distribuídas

2.2.3.1 Plataformas serverless

Um sistema pode ser *onpremise* ou situar-se na nuvem. Existem várias plataformas para desenvolver aplicações *serverless*, por exemplo: Google Cloud Platform [42], Microsoft Azure [43], Amazon Web Services (AWS) [44], IBM Cloud [45] e Alibaba Cloud [46].

2.2.3.2 Kubernetes

"Kubernetes é uma plataforma *open-source* que gere cargas de trabalho e serviços, facilitando a sua configuração declarativa e automação. Desta forma, facilita a implantação de aplicações, para além de os seus serviços, suporte e ferramentas serem abundantes"[47].

2.2.3.3 KEDA

"As arquiteturas orientadas a eventos correspondem a uma evolução natural dos microsserviços, que proporcionando um desenho flexível de desacoplado estão progressivamente a ser adotados por clientes empresariais. Serviços *serverless* como Azure Functions possuem um *design* orientado a eventos, mas os clientes lamentavam as lacunas destas capacidades em soluções baseadas em Kubernetes. O *scaling* em Kubernetes correlaciona-se com a memória do CPU [2] de um *container* [3]"[50].

Por outro lado, existem serviços que são conscientes das fontes de eventos e consequentemente conseguem escalar baseando-se em sinais que vêm diretamente da origem ainda antes de existir impacto em termos de CPU ou memória. Desta forma, KEDA - Kubernetes-based event-driven autoscaling - tem como foco unir os benefícios de arquiteturas orientadas a eventos/mensagens aos de Kubernetes [50].

Dado que o KEDA lida com eventos, existem diversas tecnologias que são utilizadas para os desencadear. Na realidade, há um conjunto denominado "event handlers" que permitem auxiliar a lógica de negócio na nuvem. Numa parte destes cenários, é criado dado *trigger* que, aquando de um evento, é acionado e desencadeia a respetiva resposta. Algumas das ferramentas que têm funções deste género são: AWS Lambda, Azure Functions, Azure Logic Apps e Google Cloud Functions [41].

2.2.3.4 Dapr (Distributed Application Runtime)

"A ferramenta *open-source* Dapr auxilia os programadores aquando do uso de qualquer linguagem ou *framework* para construir facilmente aplicações baseadas em microsserviços, tanto nos casos da escrita de novo código como nos de migração"[51].

Um exemplo do uso desta ferramenta com uma tecnologia para o uso de *messaging*, neste caso o Azure Service Bus, pode ser observado na figura 2.14.

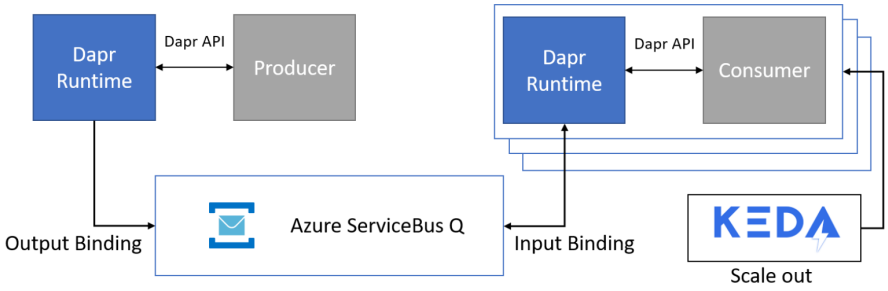

Figura 2.14: Dapr
Fonte: [52]

[2] CPU - *Central Process Unit* - realiza a maioria do processamento de um dispositivo [48].
[3] Um *container* é uma unidade padrão de *software* que integra o código e as suas dependências para que a aplicação corra de forma rápida e confiável de um ambiente computacional para outro [49].

2.3 Resumo do capítulo

Na primeira secção deste capítulo foram expostos os conceitos referentes aos estilos de integração de sistemas, estilos de controlo distribuído, estilos de arquitetura e implantação de soluções distribuídas.

Por sua vez, na secção referente ao estado da arte em tecnologia relevante são referios os protocolos de *messaging*, tecnologias para comunicação entre componentes, incluindo respetiva comparação, e algumas ferramentas utilizadas na implantação de soluções distribuídas.

Desta forma, foram reunidos os estilos arquiteturais, formas de interligar os componentes e tecnologias relevantes para o projeto descrito no presente documento.

Capítulo 3

Análise de negócio

Neste capítulo apresenta-se a análise de negócio, incluindo os detalhes sobre contexto e problema, processos e intervenientes, simplificação e generalização e restrições existentes.

3.1 Detalhes sobre contexto e problema

Esta secção especifica o contexto e o problema do trabalho no âmbito do SmartDocumentor.

A necessidade de os clientes possuírem o SmartDocumentor advém do facto de o sistema permitir a automatização de processos. A automatização na inserção de metadados em sistemas de gestão ou arquivo permite poupar tempo em atividades de pouco valor numa entidade. Desta forma, é possível focar cada vez mais as tarefas na análise dos dados e nos processos a jusante dos resultados dessa mesma análise.

O SmartDocumentor funciona de uma forma semelhante à da figura 3.1, sendo a Review opcional dependendo das necessidades da organização que utiliza o produto.

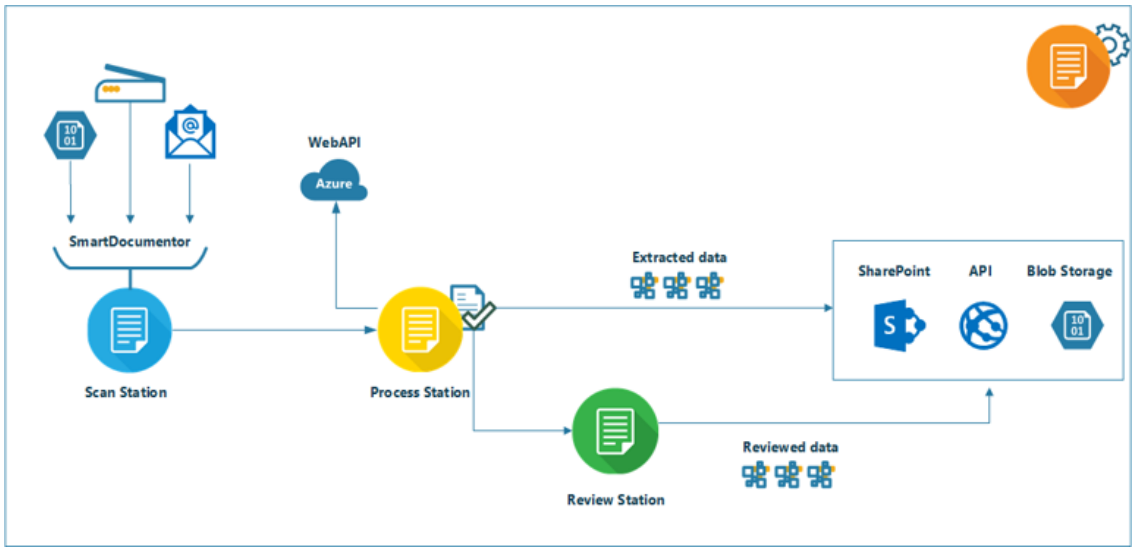

Figura 3.1: Fluxo de trabalho do SmartDocumentor

Relativamente ao número de clientes, o SmartDocumentor possui 40 instalações nacionais e 4 internacionais, estando a quantidade de digitalizações dependente da organização.

Existem dois tipos de serviço de revisão:

- Instalação em cliente - é instalada a estação de revisão no cliente;
- Business Process Outgoing (BPO) - os clientes em BPO são cada vez mais uma realidade nacional e internacional devido à crescente necessidade de desmaterialização, tanto por razões de espaço como por imposição de normas e regulamentos europeus. As empresas com diversos pontos de entrada são os principais clientes deste tipo de serviço, não sendo necessário a instalação da estação de revisão na respetiva organização.

Tal como já foi referido, pode até não ser necessário instalar a estação de revisão, dependendo do género de documentos a digitalizar. Assim sendo, o fluxo de trabalho da organização em causa é acompanhado pelo respetivo fluxo de trabalho técnico.

No que diz respeito à tipificação da companhia do cliente, o SmartDocumentor é transversal a empresas de todos os setores com contabilidade interna ou necessidade de arquivo e desmaterialização de documentos diversos.

No entanto, o processo do SmartDocumentor apresenta uma restrição, dado que após a identificação dos metadados, poderá requerer a re-digitalização do documento do arquivo físico, quando necessário.

3.2 Processos e intervenientes

Nesta secção apresenta-se o modelo de domínio, tal como são as vistas de cenários, lógica e de processos de forma a albergar os processos e intervenientes do projeto descrito neste documento.

3.2.1 Modelo de domínio

Nesta subsecção é apresentado o modelo conceptual do SmartDocumentor na figura 3.2, sendo especificados os conceitos de negócio.

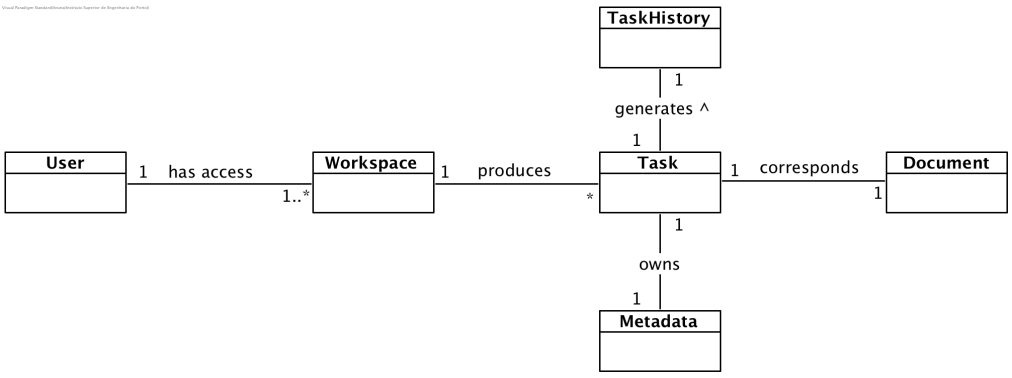

Figura 3.2: Modelo de domínio

3.2.2 Vista de cenários

A vista de cenários do sistema descrito neste documento corresponde ao diagrama de casos de uso da figura 3.3, intervindo três atores:

- `Digitalizador` - tem como função digitalizar os documentos;
- `Revisor` - revê os documentos digitalizados;
- `Consumidor` - consulta os documentos previamente digitalizados e revistos.

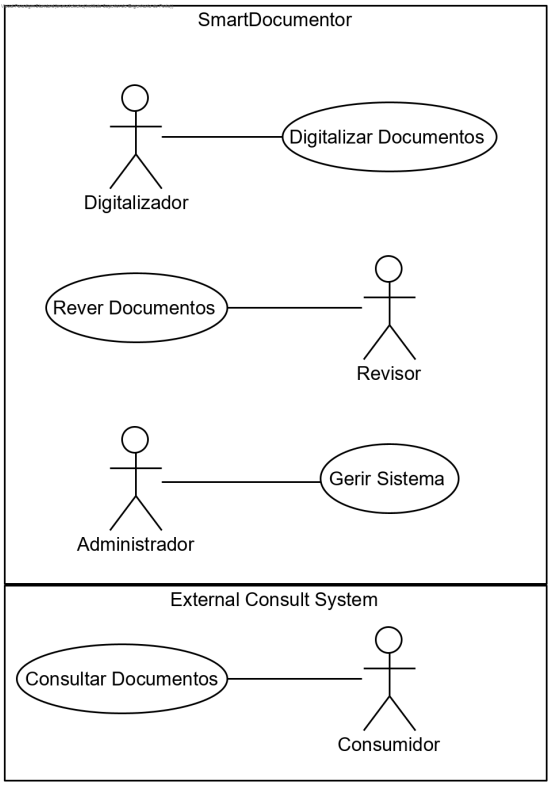

Figura 3.3: Diagrama de casos de uso

3.2.3 Vista lógica

No diagrama de componentes de alto nível da figura 3.4, pode-se observar como os atores interagem de forma a realizarem as suas tarefas, existindo três componentes:

- `Scanner` – dispositivo de entrada que permite a digitalização dos documentos;
- `SmartDocumentor` – componente que fornece as interfaces de gestão, digitalização e revisão. Os dados dos documentos validados podem ser integrados num sistema externo;
- `External Consult System` – sistema de consulta externo que fornece a interface de utilizador de consulta. Este componente corresponde a um sistema de gestão empresarial (ERP), gestão documental (ECM), gestão de clientes (CRM) ou qualquer outro sistema que permita a consulta de documentos.

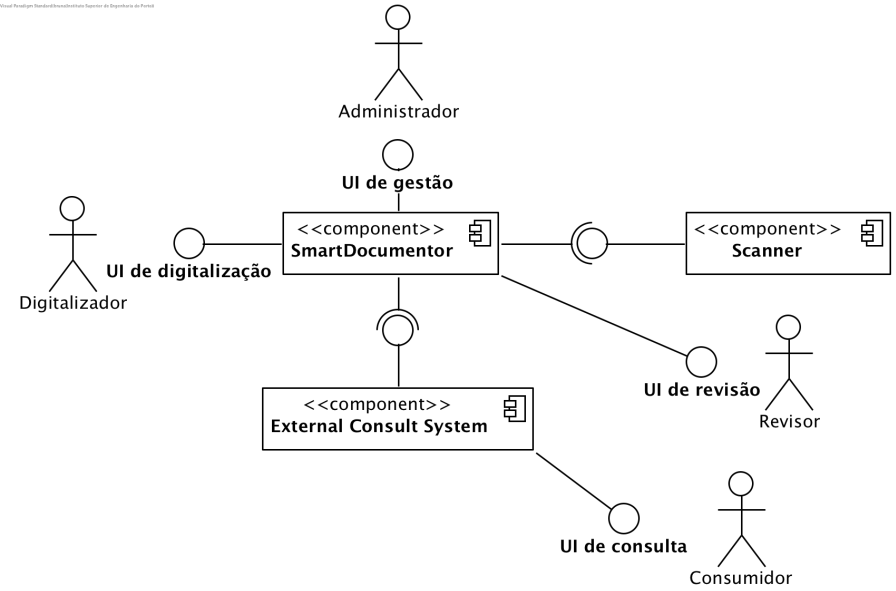

Figura 3.4: Diagrama de componentes de alto nível

Na figura 3.5 apresenta-se o diagrama de componentes do SmartDocumentor. Existem cinco componentes:

- `Scan` – componente que interage diretamente com o utilizador e que guarda informações na base de dados;
- `Process` – processa os itens existentes na base de dados que necessitam desta ação;
- `Review` – verifica se o componente anterior efetuou todas as operações de forma correta;
- `Management` – possibilita a gestão do sistema, possuindo as funções de alteração de definições e criação de *workspaces*;
- `Database` – base de dados que armazena e permite acesso aos dados extraídos/reconhecidos necessários às operações.

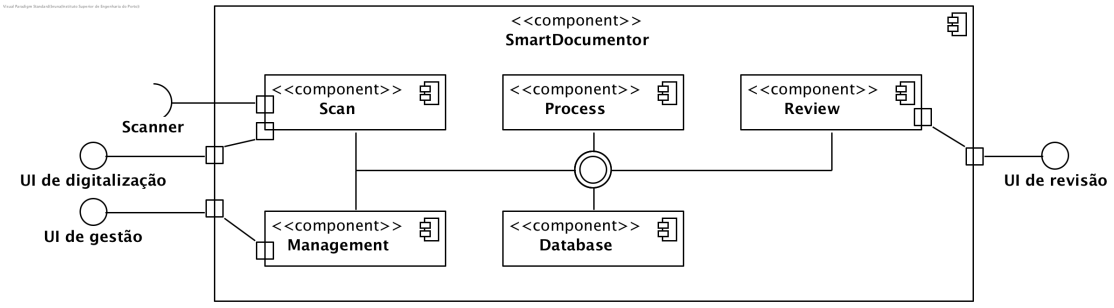

Figura 3.5: Diagrama de componentes do SmartDocumentor

3.2.4 Vista de processos

De seguida apresentam-se as vistas de processos sob a forma de diagrama de sequência dos principais casos de uso do sistema, assim como o referente ao processamento presente no SmartDocumentor.

3.2.4.1 Digitalizar Documentos

Este caso de uso processa-se da forma apresentada na figura 3.6:

- O digitalizador inicia o processo de digitalização de documentos;
- A estação de Scan efetua a digitalização do documento que consta do Scanner;
- O Scanner retorna à estação de Scan os dados relativos ao documento digitalizado;
- A estação de Scan guarda os dados recolhidos na base de dados e informa ao digitalizador o sucesso da operação.

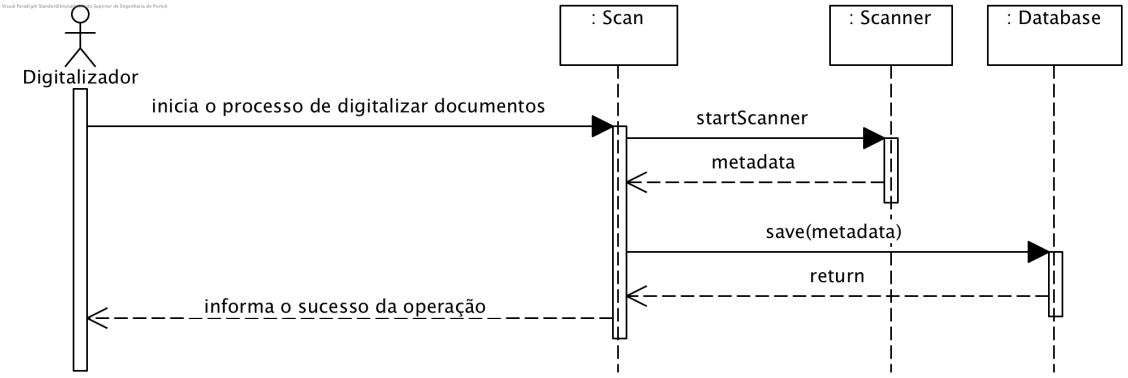

Figura 3.6: Diagrama de sequência - Digitalizar Documentos

3.2.4.2 Processar Documentos

O processamento de documentos possui os passos da figura 3.7:

- De 30 em 30 segundos a estação de Process efetua o *polling* à base de dados, verificando se existe algum item com o estado pretendido;
- A base de dados retorna o resultado;
- Caso o resultado não seja nulo, o item é processado e atualizado na base de dados.

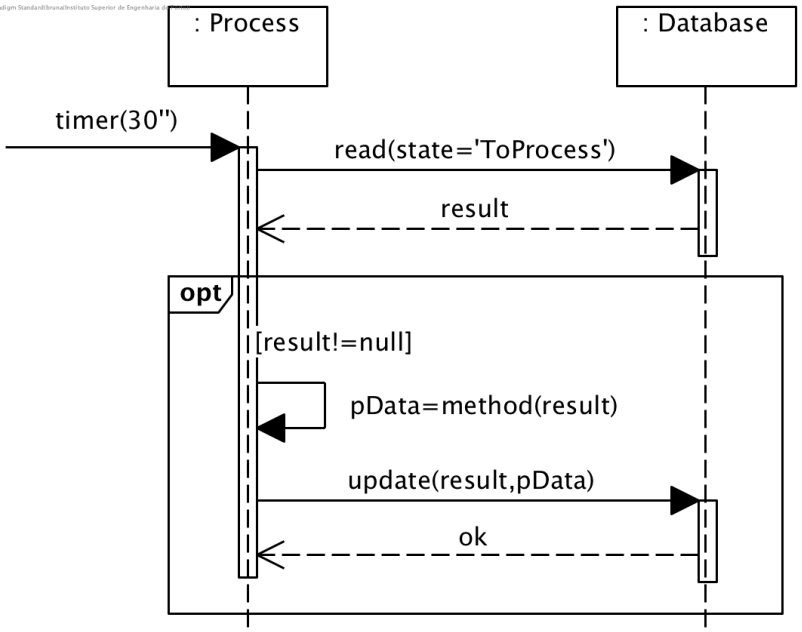

Figura 3.7: Diagrama de sequência - Processar Documentos

3.2.4.3 Rever Documentos

A revisão de documentos acontece do modo apresentado na figura 3.8:

- De 30 em 30 segundos a estação de Review efetua o *polling* à base de dados, verificando se existem documentos para rever;
- O Revisor inicia o processo de revisão de documentos;
- A estação de Review mostra os documentos previamente requisitados à base de dados e solicita seleção;
- O Revisor seleciona um documento;
- A estação de Review mostra os dados do documento;
- O Revisor visualiza o documento e efetua a sua revisão;
- A estação de Review atualiza os dados segundo a revisão realizada e atualiza a base dados;
- A base de dados retorna o documento revisto e o mesmo é salvo num sistema de consulta externa.

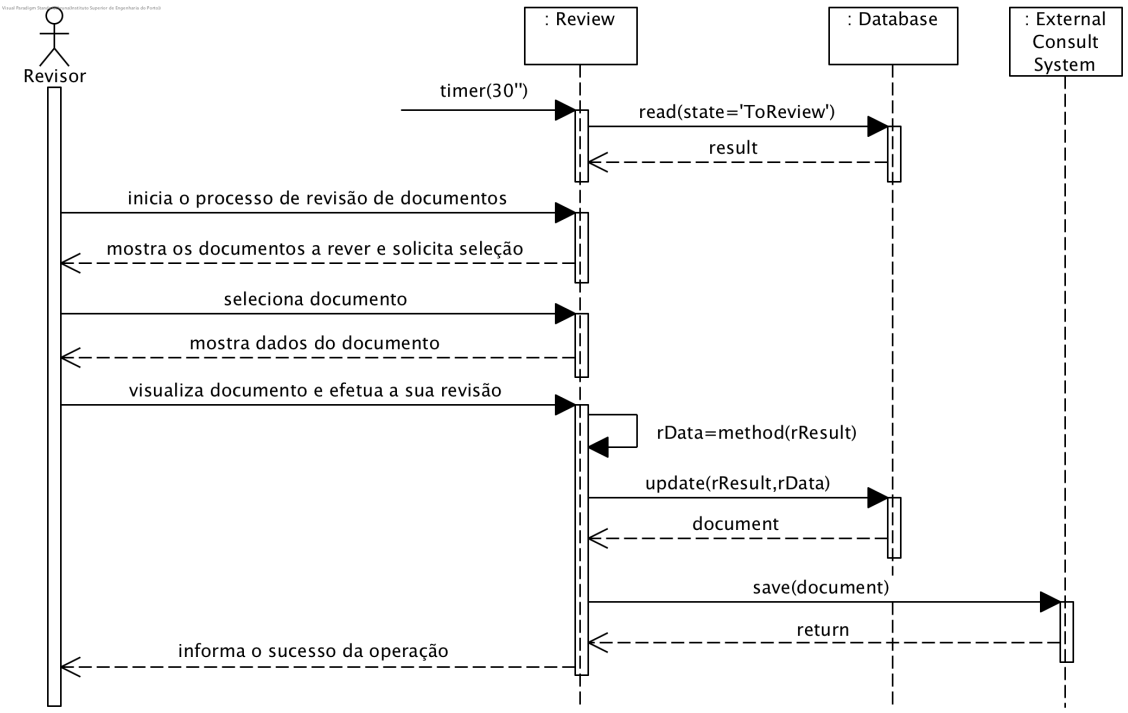

Figura 3.8: Diagrama de sequência - Rever Documentos

Posto isto, vários estilos e padrões arquiteturais são adotados neste sistema do SmartDocumentor, entre os quais:

1. *Client-Server* - existem componentes que funcionam como servidores e fornecem serviços aos respetivos clientes;
2. *Service-oriented architecture (SOA)* - o sistema é composto por diversos serviços reusáveis pelas interfaces disponibilizadas;
3. *Component-based architecture* - o sistema encontra-se decomposto em componentes lógicos que representam interfaces de comunicação que contêm métodos, eventos e propriedades e que podem ser substituídos por outros desde que cumpram a interface definida, o que neste caso significa adotar a estrutura e semântica da base de dados partilhada;
4. *Shared database* - os componentes Scan, Process e Review comunicam diretamente com a base de dados;
5. *Polling* - os componentes Scan, Process e Review acedem à base de dados de 30 em 30 segundos de forma a verificar se há algum dado que tenha as características procuradas.

3.2.5 Generalização

Nesta subsecção apresenta-se a generalização das vistas lógica e de processos do SmartDocumentor.

Da vista lógica da figura 3.5, obteve-se o diagrama de componentes da figura 3.9 que contém:

- `Application A` – aplicação que interage diretamente com o utilizador e que trata dos dados iniciais, estando as restantes dependentes desta ação;
- `Application B` – aplicação que pretende tratar dados com determinado(s) atributo(s) e que por este motivo se encontra dependente de alguma ação;
- `Database` – base de dados partilhada pelas diversas aplicações.

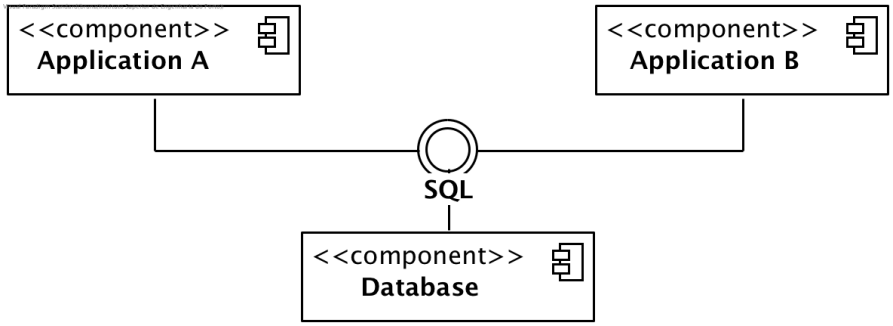

Figura 3.9: Diagrama de componentes generalizado

A vista de processos da figura 3.10 representa os seguintes passos:

- O utilizador interage com o sistema através da inserção de informação;
- A aplicação A processa a informação introduzida, guardando o resultado na base de dados;
- A aplicação B efetua periodicamente pedidos à base de dados de forma a verificar se a mesma possui algum item que possa processar;
- Caso a base de dados retorne algum dado relevante, a aplicação B processa o mesmo e volta a guardá-lo com as alterações efetuadas.

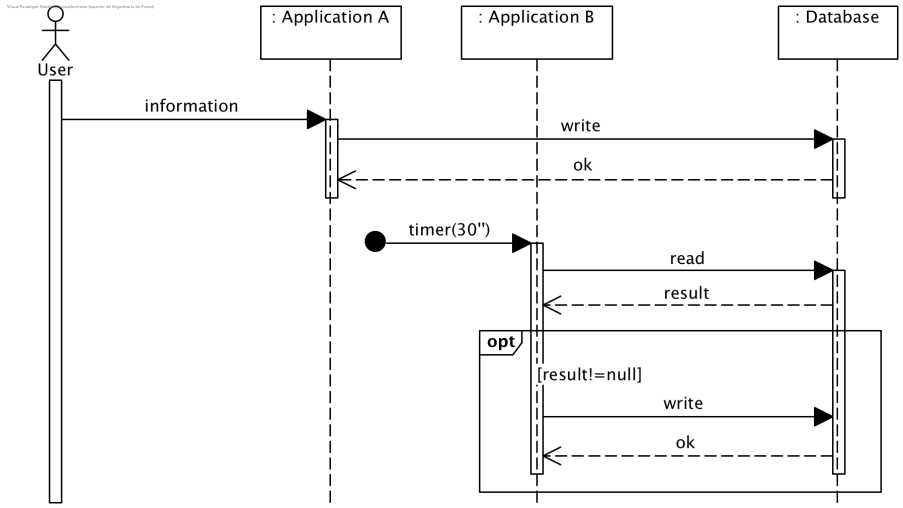

Figura 3.10: Processo generalizado

3.3 Restrições existentes

Nesta secção são apresentadas as restrições existentes quanto ao desenvolvimento da solução.

Aquando da reengenharia do sistema inicial descrito anteriormente, tem-se de ter em consideração que:

- As funcionalidades existentes no sistema inicial devem ser mantidas, nomeadamente o número de documentos a rever;
- O sistema inicial deve ser melhorado e não completamente reformulado;
- A monitorização do fluxo de mensagens entre os diversos componentes é considerado importante pela empresa, dado que oferece vantagens a nível de gestão de recursos;
- Deve ser possível alterar a tecnologia utilizada para a comunicação entre componentes;
- Deve existir integração e entrega contínuas do sistema;
- As exceções que possam ocorrer devem ser tratadas.

Capítulo 4

Análise de valor

Considerando o contexto apresentado anteriormente, no presente capítulo é realizada a análise de valor do projeto descrito neste documento.

A Análise de valor (AV) permite uma abordagem organizada para melhorar a rentabilidade de aplicações de produto [53].

A análise de valor é um processo sistemático, formal e organizado de análise e avaliação. As técnicas que suportam as atividades de AV incluem técnicas comuns usadas para todos os exercícios de análise de valor e outras apropriadas em determinadas condições [53].

O processo de análise de valor é representado na figura 4.1, juntamente com algumas técnicas. Este inclui as fases [54]:

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

Tendo em conta o processo de análise de valor da figura 4.1 e que "every problem is an opportunity"[55], nas secções seguintes apresenta-se: o processo de inovação, o valor, valor para o cliente e valor percecionado, a proposta de valor e o modelo Canvas.

4.1 Processo de inovação

Tal como indicado na figura 4.2, o processo de inovação pode ser dividido em três partes:

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

A primeira parte, FFE, é geralmente considerada uma das maiores oportunidades de melhoria de todo o processo de inovação [56].

O foco nas atividades de front-end precede o formal e estruturado processo de modo a aumentar o seu valor, quantidade e probabilidade de sucesso de conceitos altamente rentáveis

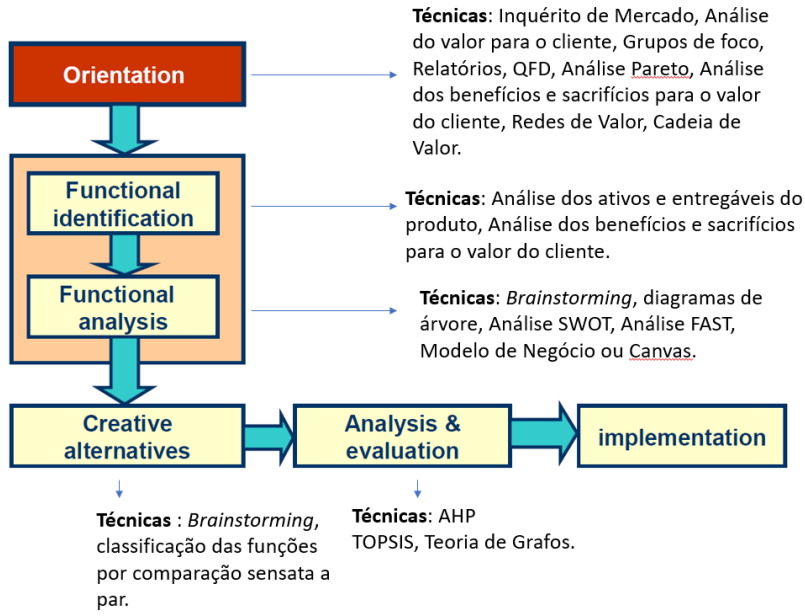

Figura 4.1: Processo de análise de valor
Fonte: [54] (adaptado)

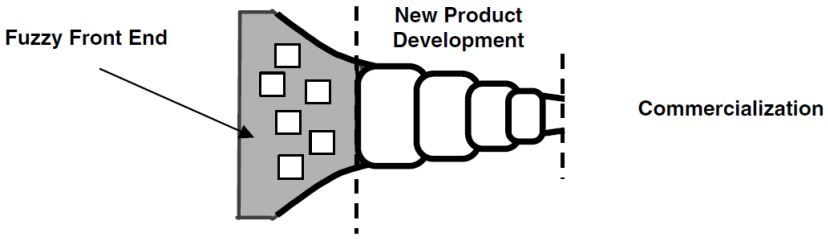

Figura 4.2: Modelo Fuzzy Front End
Fonte: [56]

pela introdução do desenvolvimento do produto e comercialização. O Modelo de desenvolvimento de novo conceito (NCD) foi criado para gerir eficazmente o FFE, pelo que se encontra apresentado na subsecção seguinte [56].

4.1.1 Modelo de desenvolvimento de novo conceito (NCD)

O NCD fornece uma linguagem e perceção comuns das atividades do front-end [57].

Tal como se pode observar na figura 4.3, o modelo de desenvolvimento de novo conceito foca-se no progresso do produto e divide-se em três partes [58], [59]:

- **Motor** – parte central constituída pelos elementos principais que auxiliam com maior impacto o processo, correspondendo a atributos da organização, nomeadamente gestão envolvida, visão, estratégia, recursos, cultura, equipas e colaboração.
- **Roda** – parte interior do modelo que inclui cinco elementos de atividade:

  - `Identificação da oportunidade` – esta escolha a seguir por parte da organização é importante, uma vez que existirá a alocação de recursos e tecnologias a novas áreas de crescimento que dependem dos objetivos do negócio. De facto, a essência deste elemento encontra-se na origem e métodos utilizados para identificar oportunidades, nomeadamente mapeamento, análise das tendências tecnológicas, verificação de clientes e da competição, pesquisa de mercado e planeamento de cenário;
  - `Análise da oportunidade` – neste elemento é confirmado se vale a pena seguir a oportunidade identificada anteriormente. Desta forma, deve ser procurada informação adicional para transformar a mesma em oportunidades de negócio e tecnológicas específicas. Uma análise inclui os mesmos métodos empregados para identificar futuras oportunidades, porém de uma forma muito mais detalhada;
  - `Produção e enriquecimento da ideia` – inclui o surgimento, desenvolvimento e maturação da ideia em concreto, alguns dos métodos utilizados incluem o envolvimento de um cliente, a identificação de novas soluções tecnológicas, as necessidades do mercado e negócio em constante renovação devido aos avanços tecnológicos e a variedade dos incentivos de estimulação;
  - `Seleção da ideia` – efetuar uma boa seleção é essencial à saúde sucesso futuros da empresa. No entanto, o processo de efetuar uma boa decisão não se encontra padronizado. Assim sendo, a maioria das decisões inclui a identificação da oportunidade, a análise da oportunidade e a produção e enriquecimento da ideia, normalmente com novas perspetivas dos fatores influenciadores e novas diretrizes do motor;
  - `Definição de conceito` – consiste no elemento final do modelo. O inovador deve concluir através da construção e um caso apelativo a investimento na proposta de âmbito de negócio ou tecnológica.

- **Aro** – parte externa que representa os fatores do meio ambiente que influenciam o motor e os elementos de atividade nos âmbitos: político, económico, social, tecnológico e legal.

O modelo é circular de modo a indicar o fluxo de ideias, circulação e iteração de entre os cinco elementos. As setas que apontam em direção ao modelo retratam os pontos iniciais para projetos que podem começar tanto na identificação da oportunidade como na produção e enriquecimento da ideia [58], [59].

De facto, uma visão mais clara do mercado e de requisitos técnicos, fontes de risco e um plano de negócio bem definido para um novo produto pode auxiliar a uma gestão mais eficiente dos estados de desenvolvimento e comercialização de forma a diminuir o número de refazer e redirecionar atividades [56].

Tendo em conta o referido acerca do modelo de desenvolvimento de novo conceito, foram identificados os seguintes elementos no contexto do presente documento [2]:

- **Identificação da oportunidade** – Os sistemas distribuídos têm evoluído ao longo do tempo, passando pelo uso das seguintes abordagens:

  1. `File Transfer` - uma aplicação escreve num ficheiro e outra lê a informação posteriormente. Os requisitos incluem que ambas as aplicações tenham definido

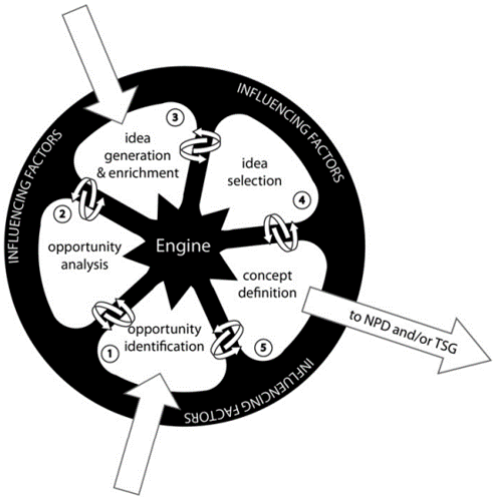

Figura 4.3: Modelo de desenvolvimento de novo conceito
Fonte: [58]

o mesmo nome, localização e formato de ficheiro, quando o mesmo vai ser escrito ou lido e quem o elimina;

2. `Shared Database` - múltiplas aplicações partilham o mesma esquema de base de dados, localizado numa única base de dados física. Em consequência da não duplicação da componente de armazenamento de dados, não é necessária a sua transferência de uma aplicação para outra;

3. `Remote Procedure Invocation` - uma aplicação expõe funcionalidades de forma a serem acedidas pelas demais remotamente. A comunicação ocorre em tempo real e de modo síncrono;

4. `Messaging` - uma aplicação publica dada mensagem para um message channel comum. As restantes podem ler as respetivas mensagens e a comunicação é assíncrona.

Anteriormente ao reconhecimento do facto de que a existência de eventos é importante no desenho de sistemas, estava em voga aceder às restantes aplicações remotamente. Por outro lado, previamente era utilizado o método de base de dados partilhada, que é o caso inicial explorado neste documento [60]. Efetivamente, nos últimos anos ocorreu uma transformação no que diz respeito ao facto de mais e mais programadores desenvolverem aplicações escaláveis na nuvem, aproveitando os serviços de gestão disponibilizados para as implantar e correr. Desta forma, as arquiteturas baseadas em microsserviços tornaram-se o padrão para construir sistemas na nuvem e prevê-se que em 2022 90% das novas aplicações serão deste género.

Esta alteração deu-se dado que uma arquitetura orientada a microsserviços oferece benefícios aliciantes, incluindo escalabilidade, serviços pouco acoplados e implantações independentes. No entanto, esta abordagem não é simples, pois é necessário dominar a compreensão e técnicas no âmbito de sistemas distribuídos. Programadores que pretendem focar-se na lógica de negócio, frequentemente e incrementalmente migrando código enquanto se apoiam em plataformas para fornecer às suas aplicações escalabilidade, resiliência, manutenibilidade, elasticidade e outros atributos de sistemas implantados na nuvem. Todavia, consideram complicada a limitação de portabilidade

entre a nuvem e respetiva extremidade, tendo de continuamente acabar por resolver os mesmos problemas inerentes a sistemas distribuídos, nomeadamente gestão de estados, chamadas resilientes de métodos e lidar com eventos [51].

Sendo assim, a oportunidade inclui passar sistemas distribuídos que utilizam estilos como *shared database* para sistemas com arquiteturas orientadas a mensagens;

- **Análise da oportunidade** – Tendo em conta o *roadmapping* realizado no ponto anterior, relativamente à análise pode-se referir que será realizada uma reengenharia a aplicações como o SmartDocumentor, já que é reconhecido que a arquitetura *shared database* funciona, porém não é adequada;
- **Produção e enriquecimento da ideia** – De modo a enriquecer a ideia, primeiramente foram reunidos os principais conceitos relacionados com integração de sistemas e arquiteturas orientadas a mensagens. Em segundo lugar, foi realizado o estado da arte em tecnologia relevante, nomeadamente a pesquisa de plataformas na nuvem, protocolos de mensagens e tecnologias para o uso de *messaging*;
- **Seleção da ideia** – Foi efetuada a comparação entre as tecnologias para o uso de *messaging* e selecionou-se o Azure Service Bus e RabbitMQ de forma a efetuar os testes seguintes;
- **Definição de conceito** – O objetivo do projeto é fazer uma reengenharia, extendendo abordagens da solução a outros projetos, ou seja, serve enquanto protótipo para a implementação de arquiteturas orientadas a mensagens noutros sistemas de *software*;

4.2 Valor, valor para o cliente e valor percecionado

4.2.1 Valor

"A criação de valor é um conceito difícil de atingir, compreender, modelar e/ou conceitualizar. Alguns autores consideram que a criação de valor é um compromisso entre benefícios e sacrifícios percecionados pelos clientes da oferta de um fornecedor"[61].

Desta forma, pode-se observar os benefícios e sacrifícios do projeto descrito neste relatório na tabela 4.1.

Tabela 4.1: Benefícios e Sacrifícios

	Serviço	Relação
Benefícios	As mensagens permitem que se use comunicação assíncrona, ou seja, nenhum sistema fica empatado à espera de outro, otimizando o fluxo de dados. Caso uma parte do sistema falhe, os restantes componentes continuam a funcionar. Possibilidade de reutilizar componentes. Possibilidade de escalar ou alterar os componentes necessários sem interferir diretamente com os diversos. Possibilidade de simplificação do código das aplicações desacopladas, facilitando também a sua compreensão. Implantação mais simples e rápida.	Desempenho Confiabilidade Flexibilidade Escalabilidade Manutenibilidade Desacoplamento Tempo de implantação
Sacrifícios	A comunicação entre serviços pode ser uma operação complexa. Maior tempo de implementação relativamente a arquiteturas monolíticas caso não haja reutilização de serviços. Mais componentes para implantar, aumentando o custo inicial.	Complexidade Tempo de implementação Esforço Monitorização

4.2.2 Valor para o cliente

O termo "valor para o cliente" é utilizado no marketing para representar o que é entregue ao cliente pelo fornecedor e vice-versa. As qualidades do objeto desempenham um papel fulcral nas ações de tomada de decisão, pois o cliente procura atributos do produto/serviço que tragam vantagem para as suas vidas [62].

As arquiteturas orientadas a mensagens apresentam valor para o utilizador, dado que permitem a otimização de sistemas de *software*, da sua implantação e da sua implementação a longo prazo.

4.2.3 Valor percecionado

Distintos clientes percecionam diferente valor para os mesmos produtos/serviços. Para além disso, organizações que possam estar envolvidas no processo de compra podem possuir diferentes perceções do que seja valor para os clientes [63].

Apesar de as arquiteturas orientadas a mensagens possuírem uma complexidade mais elevada relativamente, por exemplo, ao uso de base de dados partilhada, aquando da compreensão do seu funcionamento tornam-se uma boa solução a aplicar. Portanto, dependendo do conhecimento e necessidades do utilizador pode ser percecionada enquanto adequada ou desadequada.

4.3 Proposta de valor

Para construir a gestão num cenário de desenvolvimento de *software*, devem ser aplicados modelos formais. Uma abordagem formal à modelação de propostas de valor permite aos gestores aproveitar os modelos mentais, perceber e comunicar as mesmas, melhorar a sua implementação, comparar com a competição e eventualmente promover a inovação [64].

Os pilares "o quê", "quem", "como" e "quanto" traduzem-se na expressão do que a organização oferece, os seus alvos, como pode ser realizado e quanto pode ser ganho por o efetuar [64].

Assim sendo, a proposta de valor deste documento depreende-se no facto de as arquiteturas orientadas a mensagens permitirem aos desenvolvedores de *software* melhorar os seus sistemas através da otimização da comunicação entre componentes das aplicações, havendo consequente diminuição dos custos de transação e favorecimento da reutilização de componentes.

4.4 Modelo Canvas

O modelo Canvas é uma ferramenta de negócio que representa a estrutura da empresa para gerar e capturar valor. O mesmo encontra-se dividido em nove blocos:

- **Segmentos de Clientes** (**Customer Segments**) – uma organização atua num ou vários segmentos de cliente;
- **Propostas de Valor** (**Value Propositions**) – procuram resolver os problemas do(s) cliente(s) e satisfazer as respetivas necessidades;
- **Canais** (**Channels**) – dada proposta de valor é entregue ao(s) cliente(s) através de canais de comunicação, distribuição e venda;
- **Relações com Clientes** (**Customer Relationships**) – são estabelecidas e mantidos com cada um dos segmentos de cliente;
- **Fontes de Receita** (**Revenue Streams**) – resultam da proposta de valor oferecida com sucesso aos clientes;
- **Recursos Chave** (**Key Resources**) – recursos requeridos para oferecer e entregar os elementos previamente descritos;
- **Atividades Chave** (**Key Activities**) – atividades pelas quais os recursos são entregues;

- **Parcerias Chave** (**Key Partnerships**) – algumas atividades são feitas por terceiros e alguns recursos são adquiridos fora da organização;
- **Estrutura de Custos** (**Cost Structure**) – recursos requeridos para oferecer e entregar os elementos previamente descritos, sendo que os elementos do modelo de negócio resultam no custo da estrutura [65].

O Modelo Canvas referente ao projeto descrito neste documento encontra-se representado na figura 4.4.

<table>
<tr>
<td rowspan="2">Key Partners

• Equipas de desenvolvimento de software</td>
<td>Key Activities

• Desenvolvimento da plataforma</td>
<td rowspan="2">Value Propositions

• Aprimoração da arquitetura de soluções de software, inclusive implantação e implementação a longo prazo
• Maior escalabilidade, flexibilidade e confiabilidade</td>
<td>Customer Relationships

• Suporte
• Otimização das soluções</td>
<td rowspan="2">Customer Segments

• Colaboradores da DevScope
• Parceiros da DevScope
• Clientes da DevScope</td>
</tr>
<tr>
<td>Key Resources

• Equipa
• Soluções existentes
• Membros das equipas de desenvolvimento de software</td>
<td>Channels

• Arquiteturas e respetivas soluções produzidas
• Demostrações para as equipas</td>
</tr>
<tr>
<td colspan="3">Cost Structure

• Equipas de desenvolvimento de software
• Desenvolvimento e manutenção da solução
• Implantação</td>
<td colspan="2">Revenue Streams

• Diminuição de tempo de implementação e implantação
• Aumento da confiabilidade e manutenibilidade</td>
</tr>
</table>

Figura 4.4: Modelo Canvas

Capítulo 5

Desenho da solução

O presente capítulo tem como foco o desenho da solução a ser construída segundo as possibilidades existentes.

Primeiramente é realizada a análise das abordagens a aplicar e efetuada a respetiva seleção segundo a avaliação, dado que a arquitetura do sistema se encontra dependente destas decisões.

Posteriormente é apresentado o desenho arquitetural da solução proposta e consequente generalização para facilitar a sua aplicação noutros sistemas.

5.1 Abordagens alternativas

Nesta secção são apresentadas alternativas de tecnologias e procede-se à seleção criteriosa das mesmas. Para isso, adotou-se o método AHP para priorizar e selecionar as tecnologias a adotar.

"O Analytic Hierarchy Process (AHP) é um método de apoio à decisão multicritério (AMD) que permite priorizar alternativas em cenários de critérios conflituosos, tendo o intuito de satisfazer as restrições com objetivos conflituantes, isto é, uma solução sem compromisso [66]". "A ideia principal do AHP é dividir o problema de decisão em níveis hierárquicos, o que facilita a sua compreensão e avaliação, havendo a possibilidade do uso de critérios qualitativos bem como quantitativos no processo"[67].

O método AHP é composto pelas fases [54]:

1. Construção da árvore hierárquica de decisão;
2. Comparação entre os critérios da hierarquia;
3. Prioridade relativa de cada critério;
4. Avaliar a consistência das prioridades relativas;
5. Construção da matriz de comparação paritária para cada critério, considerando cada uma das alternativas selecionadas;
6. Obter a prioridade composta para as alternativas;
7. Escolha da alternativa.

Seguidamente, são analisadas e selecionadas as tecnologias de suporte.

Decidiu-se testar as alternativas que melhor preenchiam os critérios da tabela 2.2 do capítulo 2 correspondente à comparação entre tecnologias para o uso de *messaging*.

Desta análise resultaram as alternativas: RabbitMQ, Apache Kafka e Redis. Por outro lado, a alternativa Azure Service Bus também foi escolhida dado que preenche os requisitos essenciais e a empresa possui uma parceria que pode ser proveitosa no âmbito do uso da mesma.

Portanto, foi realizada a árvore hierárquica de decisão da figura 5.1 composta por:

- `Objetivo` - Escolher tecnologia;
- `Critérios` - Monitorização, Escalabilidade, Facilidade de implantação na nuvem, Possibilidades de implantação;
- `Alternativas` - Azure Service Bus, RabbitMQ, Apache Kafka, Redis.

Tomou-se a decisão de não incluir o desempenho enquanto critério, uma vez que não existe forma de preconizar este critério dada a falta de informação encontrada neste âmbito.

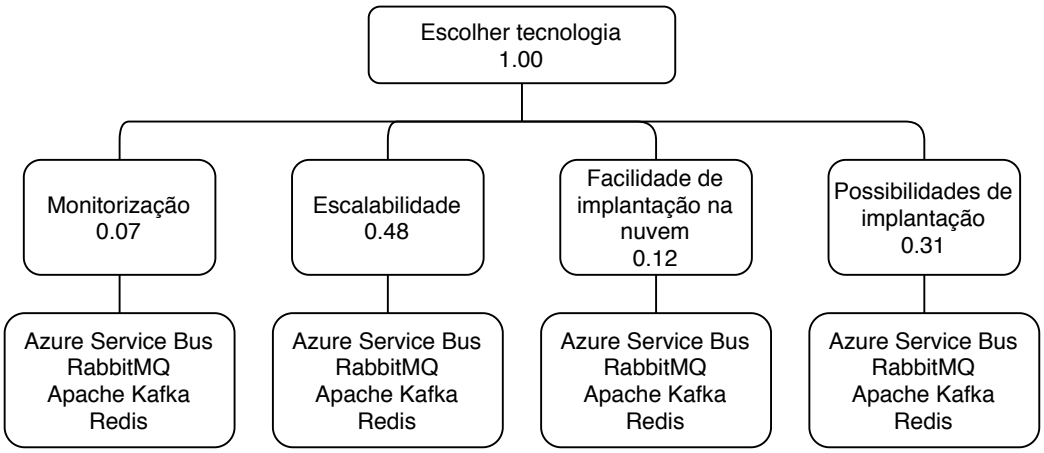

Figura 5.1: Árvore hierárquica de decisão de tecnologia

Quanto à comparação entre os critérios, construiu-se a tabela 5.1.

Tabela 5.1: Matriz de comparação dos critérios de tecnologia do segundo nível

	Monitorização	**Escalabilidade**	**Facilidade de implantação na nuvem**	**Possibilidades de implantação**
Monitorização	1	1/6	1/2	1/5
Escalabilidade	6	1	4	2
Facilidade de implantação na nuvem	2	1/4	1	1/3
Possibilidades de implantação	5	1/2	3	1

Na fase 3 construiu-se a matriz normalizada de critérios e calculadas as prioridades relativas de critérios da tabela 5.2.

Tabela 5.2: Prioridade relativa de cada critério de tecnologia

	Monitorização	**Escalabilidade**	**Facilidade de implantação na nuvem**	**Possibilidades de implantação**	**Prioridade Relativa**
Monitorização	1/14	2/23	1/17	3/53	**0,0684**
Escalabilidade	3/8	12/23	8/17	30/53	**0,4833**
Facilidade de implantação na nuvem	1/7	3/23	2/17	5/53	**0,1213**
Possibilidades de implantação	5/14	6/23	6/17	15/53	**0,3135**

Pela tabela 5.2, chega-se à conclusão de que o critério Escalabilidade é o primeiro, seguido de Possibilidades de implantação, Facilidade de implantação e, por fim, Monitorização.

Na fase 4 é calculada a Razão de Consistência (RC) dos critérios de tecnologia de forma a aferir a consistência dos julgamentos relativamente a grandes amostras de juízos completamente aleatórios.

Na subsecção referente à fase 4 da tecnologia do anexo B pode-se observar os cálculos realizados, nos quais se conclui que o valor de RC é menor que 0.1, isto é, os valores das prioridades relativas dos critérios de tecnologia encontram-se consistentes.

Seguidamente encontram-se as matrizes de comparação paritárias para cada critério, tendo em conta as alternativas escolhidas. Para simplificar, as matrizes normalizadas estão na subseção acerca da fase 5 referente a tecnologia do anexo B.

Critério - Monitorização

Tabela 5.3: Matriz de comparação paritária de monitorização

	Azure Service Bus	**RabbitMQ**	**Apache Kafka**	**Redis**
Azure Service Bus	1	4	6	3
RabbitMQ	1/4	1	3	1/2
Apache Kafka	1/6	1/3	1	1/4
Redis	1/3	2	4	1

Tabela 5.4: Prioridade relativa de monitorização

	Azure Service Bus	**RabbitMQ**	**Apache Kafka**	**Redis**	**Prioridade Relativa**
Azure Service Bus	12/21	6/11	3/7	12/19	**0,5442**
RabbitMQ	1/7	3/22	3/14	2/19	**0,1497**
Apache Kafka	2/21	1/22	1/14	1/19	**0,0662**
Redis	4/21	3/11	2/7	4/19	**0,2399**

Critério - Escalabilidade

Tabela 5.5: Matriz de comparação paritária de escalabilidade

	Azure Service Bus	**RabbitMQ**	**Apache Kafka**	**Redis**
Azure Service Bus	1	1/3	1/4	2
RabbitMQ	3	1	1/2	4
Apache Kafka	4	2	1	5
Redis	1	1/4	1/5	1

Tabela 5.6: Prioridade relativa de escalabilidade

	Azure Service Bus	**RabbitMQ**	**Apache Kafka**	**Redis**	**Prioridade Relativa**
Azure Service Bus	1/9	4/43	5/39	1/6	**0,1248**
RabbitMQ	1/3	12/43	10/39	1/3	**0,3005**
Apache Kafka	4/9	24/43	20/39	5/12	**0,4830**
Redis	1/9	3/43	4/39	1/12	**0,1959**

Critério - Facilidade de implantação na nuvem

Tabela 5.7: Matriz de comparação paritária de facilidade de implantação na nuvem

	Azure Service Bus	**RabbitMQ**	**Apache Kafka**	**Redis**
Azure Service Bus	1	3	5	3
RabbitMQ	1/3	1	3	1
Apache Kafka	1/5	1/3	1	1/3
Redis	1/3	1	3	1

Tabela 5.8: Prioridade relativa de facilidade de implantação na nuvem

	Azure Service Bus	RabbitMQ	Apache Kafka	Redis	Prioridade Relativa
Azure Service Bus	15/28	9/16	1/2	9/16	**0,5402**
RabbitMQ	5/28	3/16	3/10	3/16	**0,2134**
Apache Kafka	3/28	1/16	1/10	1/16	**0,0830**
Redis	5/28	3/16	3/10	3/16	**0,2134**

Critério - Possibilidades de implantação

Tabela 5.9: Matriz de comparação paritária de possibilidades de implantação

	Azure Service Bus	RabbitMQ	Apache Kafka	Redis
Azure Service Bus	1	1/7	1/5	1/6
RabbitMQ	7	1	4	2
Apache Kafka	5	1/4	1	1/3
Redis	6	1/2	3	1

Tabela 5.10: Prioridade relativa de possibilidades de implantação

	Azure Service Bus	RabbitMQ	Apache Kafka	Redis	Prioridade Relativa
Azure Service Bus	1/19	28/371	1/41	1/21	**0,0500**
RabbitMQ	7/19	28/53	20/41	12/21	**0,4890**
Apache Kafka	5/19	7/53	5/41	2/21	**0,1531**
Redis	6/19	14/53	15/41	6/21	**0,3079**

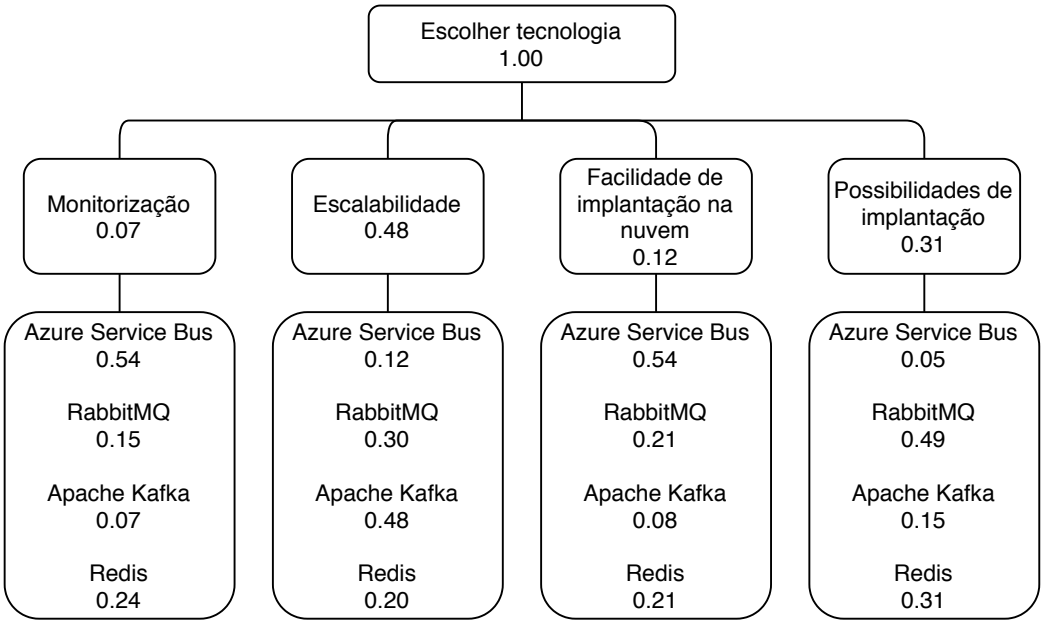

Figura 5.2: Árvore hierárquica de tecnologia resultante dos cálculos

A prioridade corresponde ao resultado entre a multiplicação da matriz prioridade e os pesos dos critérios:

$$\begin{bmatrix} 0.54 & 0.12 & 0.54 & 0.05 \\ 0.15 & 0.30 & 0.21 & 0.49 \\ 0.07 & 0.48 & 0.08 & 0.15 \\ 0.24 & 0.20 & 0.21 & 0.31 \end{bmatrix} x \begin{bmatrix} 0.05 \\ 0.49 \\ 0.15 \\ 0.31 \end{bmatrix} = \begin{bmatrix} 0.18 \\ \mathbf{0.33} \\ 0.26 \\ 0.23 \end{bmatrix}$$

A alternativa "RabbitMQ"aparece como a mais indicada enquanto escolha tecnológica, em função dos critérios definidos e respetivas importâncias.

Azure Service Bus Vs. RabbitMQ

Dada a análise multicritério realizada, escolheu-se testar na fase de implementação o Azure Service Bus e o RabbitMQ.

Apesar de aparecer como a alternativa menos indicada aquando da aplicação do método AHP, o Azure Service Bus foi escolhido com o intuito de o seu desempenho ser testado na nuvem e tendo em conta que a empresa possui parcerias que o tornam de baixo custo para o problema em questão. Por outro lado, a tomada de decisão relativa ao RabbitMQ deve-se a corresponder à melhor alternativa aquando da aplicação do método AHP e por ser o message broker open-source mais usado e possuir bastante documentação relevante.

Ambas as tecnologias podem recorrer queues ou tópicos, os dois utilizados para guardar mensagens temporárias com a distinção que no caso dos tópicos é possível ter diferentes tipos de subscritores.

Para o problema em questão, foi decidido lidar com queues, pois será empregue o padrão de consumidores concorrentes.

Devido ao facto de as duas tecnologias escolhidas serem message bus, apenas com recurso a um dos mesmos pode-se ter várias queues, tornando-os adaptáveis a mais soluções.

A grande vantagem do RabbitMQ comparativamente ao Azure Service Bus é a opção de escolha relativamente à plataforma a utilizar, não estando dependente do Azure como o segundo. De facto, o RabbitMQ pode ser utilizado em plataformas como o Azure, AWS, IBM Cloud, Google Cloud Platform, entre outras, para além haver a hipótese de ser usado onpremise.

No que se refere a **processos**, foi decidido utilizar messaging com o auxílio de message channels, sendo obtidas as arquiteturas orientadas a mensagens apresentadas no capítulo 5. Por outro lado, também é utilizado um estilo arquitetural baseado em microsserviços.

Relativamente à **implantação**, decidiu-se para cada tecnologia considerar as opções Microsoft Azure, onpremise, outras plataformas ou mesmo recorrendo a uma arquitetura semelhante ao Dapr, tal como se pode observar na tabela 5.11.

Tabela 5.11: Implantação

Tecnologia	Implantação
Azure Service Bus	Microsoft Azure, Dapr
RabbitMQ	Microsoft Azure, onpremise, outras plataformas, Dapr

No capítulo 7 as tecnologias Azure Service Bus e RabbitMQ serão testados sob a forma de prova de conceito e comparadas em determinadas grandezas de forma a obter uma comparação mais prática.

5.2 Desenho arquitetural

A presente secção contém o desenho arquitetural do sistema proposto enquanto solução e a sua generalização.

Para a representação do desenho decidiu-se aplicar uma abordagem baseada na junção dos modelos:

- Modelo C4 de Brown [68], criado com o intuito de facilitar a compreensão da arquitetura de *software* através do uso de diferentes níveis de detalhe, serão empregues os níveis de granularidade necessários à compreensão do *software*:
  - `Nível 1` - expõe o contexto do sistema no meio envolvente;
  - `Nível 2` - mostra o contexto de *container*[1];
  - `Nível 3` - representa o contexto de componentes[2];
  - `Nível 4` - descreve o contexto de componente.

[1] "Um *container* (contentor) é essencialmente um contexto ou limite no qual algum código é executado ou algum conjunto de dados é guardado. Cada contentor corresponde a uma implantação/execução distinta ou a um ambiente de execução diferente que normalmente (mas não sempre) corre no seu próprio espaço de processamento. Portanto, a comunicação entre *containers* frequentemente assume a forma de comunicação entre processos."[68]

[2] "Componente é um termo bastante sobrecarregado no desenvolvimento de *software*, porém no contexto do modelo corresponde a um grupo de funcionalidades encapsuladas numa interface bem definida."[68]

- Modelo de vistas 4+1 de Krutchen [69], usado para descrever e demostrar o sistema de várias perspetivas:
  - `Vista lógica` - resume as funções e relações entre os componentes do *software*. Esta vista pode ser representada por um diagrama de componentes;
  - `Vista de processos` - demonstra as responsabilidades e colaborações dos elementos lógicos. No contexto presente será apresentada na forma de diagrama de sequência;
  - `Vista de implantação` - implantação física de processos e componentes para nós de processamento e a rede física entre nós;
  - `Vista de cenários` - resume os casos de uso com maior importância arquitetural e os seus requisitos não funcionais;
  - `Vista de implementação` - mostra a dependência e acoplamento entre *packages* do software numa perspetiva de construção, bem como a enunciação e descrição de tecnologias e condições de implementação.

  No âmbito do projeto descrito neste documento foram adotadas, nesta secção, três das cinco vistas distintas: vista lógica, vista de processos e vista de implantação.

De seguida é apresentado o desenho arquitetural do sistema e consequente generalização ambas fundamentadas na combinação do modelo C4 de Brown [68] com o modelo de vistas 4+1 de Krutchen [69].

5.2.1 Desenho arquitetural da solução do SmartDocumentor

Nesta subsecção são apresentados os diagramas de diferentes granularidades da solução definida para o caso concreto do SmartDocumentor.

5.2.1.1 Desenho arquitetural de nível 1

5.2.1.1.1 Vista lógica

O diagrama de componentes da figura 5.3 corresponde à vista lógica de alto nível da solução pretendida que em nada difere da inicial dado que apresenta a aplicação SmartDocumentor e a sua interação com terceiros.

No mesmo pode-se observar três componentes distintos:

- `SmartDocumentor` - sistema de apoio à gestão documental e arquivo com digitalização, possui três ligações com interfaces do utilizador - gestão, digitalização e revisão;
- `Scanner` - dispositivo de entrada que fornece uma interface para digitalização de documentos;
- `External Consult System` - sistema externo no qual são armazenados os documentos tratados, fornece uma ligação a ser usada pela interface de consulta do utilizador.

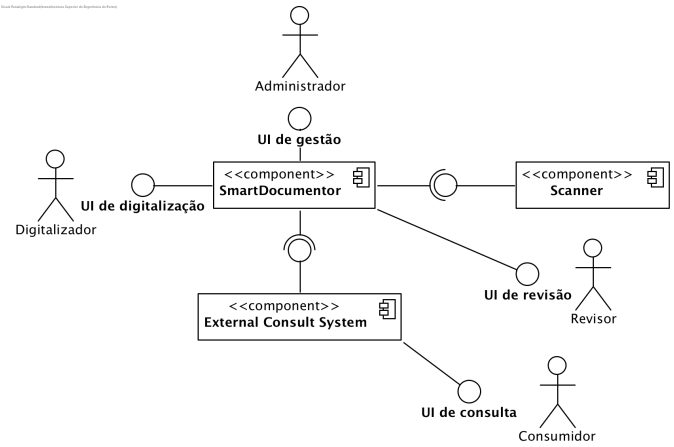

Figura 5.3: Vista lógica de nível 1

5.2.1.2 Desenho arquitetural de nível 2

5.2.1.2.1 Vista lógica

Relativamente à vista lógica de granularidade 2, foi construído o diagrama de componentes da figura 5.4. Na mesma é possível observar os componentes iniciais, representados na figura 3.5 do capítulo 3, à exceção da base de dados que foi substituída pela aplicação servidora "Backend".

A aplicação servidora contém todas as operações necessárias às ações de negócio das várias estações (Scan, Process, Review e Management).

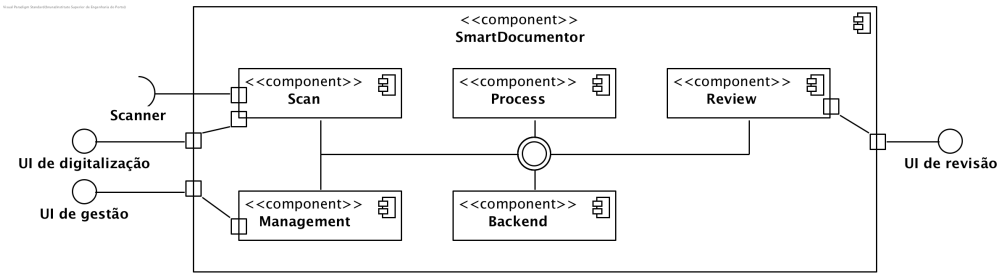

Figura 5.4: Vista lógica de nível 2

5.2.1.2.2 Vista de processos

Seguidamente são apresentadas as vistas de processos de nível 2 dos casos de uso significativos do SmartDocumentor.

5.2.1.2.2.1 Digitalizar documentos

Quanto à interação entre a interface com o utilizador e o servidor, neste caso de uso, não existe uma diferença significativa como se pode observar na figura 5.5.

Resumidamente, o processo desenrola-se da seguinte forma:

- O digitalizador seleciona a fonte de documento e insere a informação solicitada respetiva;

- A estação de Scan mostra os documentos e ativa a funcionalidade de enviar documentos para processamento;
- O digitalizador requer o envio de documentos e a estação de Scan reencaminha a informação para o Backend (aplicação servidora).

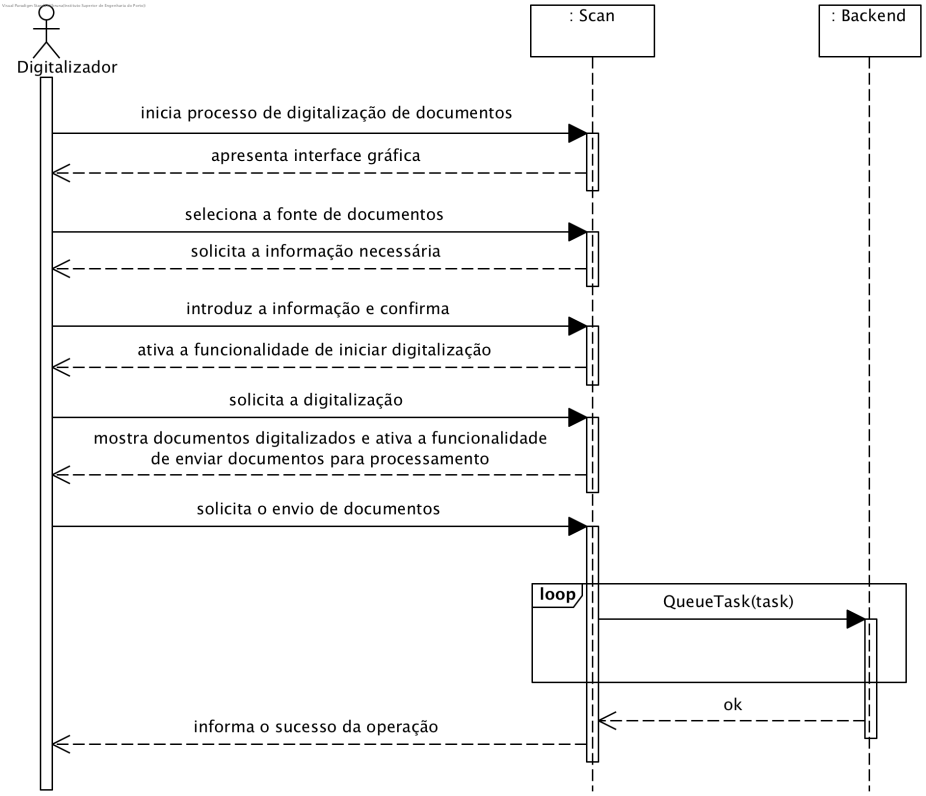

Figura 5.5: Vista de processos de nível 2 - Digitalizar documentos

5.2.1.2.2.2 Processar documentos

No que se refere a este caso de uso, a distinção entre o sistema inicial e o proposto é significativa. Esta estação deixa de contactar a base de dados de 30 em 30 segundos e passa a receber uma tarefa assim que esta fica disponível na queue respetiva.

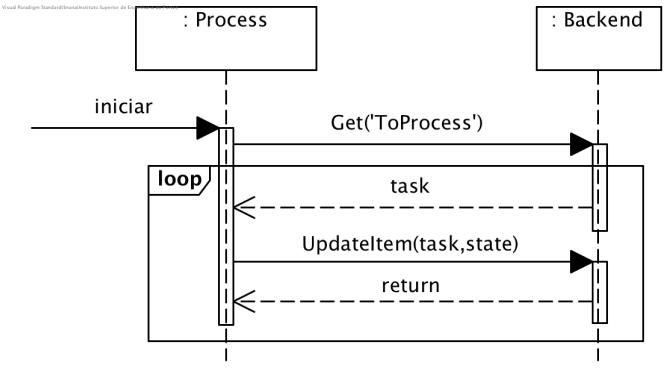

Figura 5.6: Vista de processos de nível 2 - Processar documentos

5.2.1.2.2.3 Rever documentos

Relativamente à revisão de documentos, a diferença entre o sistema inicial assenta no facto de chegar uma tarefa de cada vez, em detrimento de aquando da atualização da interface ser efetuada a recolha na base de dados de determinado número de documentos a rever.

No momento de revisão, para além de o sistema mostrar o documento respetivo e a quantidade de documentos à espera de serem revistos, solicita uma ação. Existem três alternativas de ação a realizar ao documento:

- Rever - são efetuadas as alterações necessárias ao documento e o mesmo é dado como revisto;
- Eliminar - é decidido pelo revisor eliminar a tarefa relativa ao documento;
- Reprocessar - o revisor escolhe que o documento volte a passar pela fase de processamento.

Considerando que podem existir vários revisores, esta solução traz vantagens em termos de produtividade e gestão dos recursos humanos existentes.

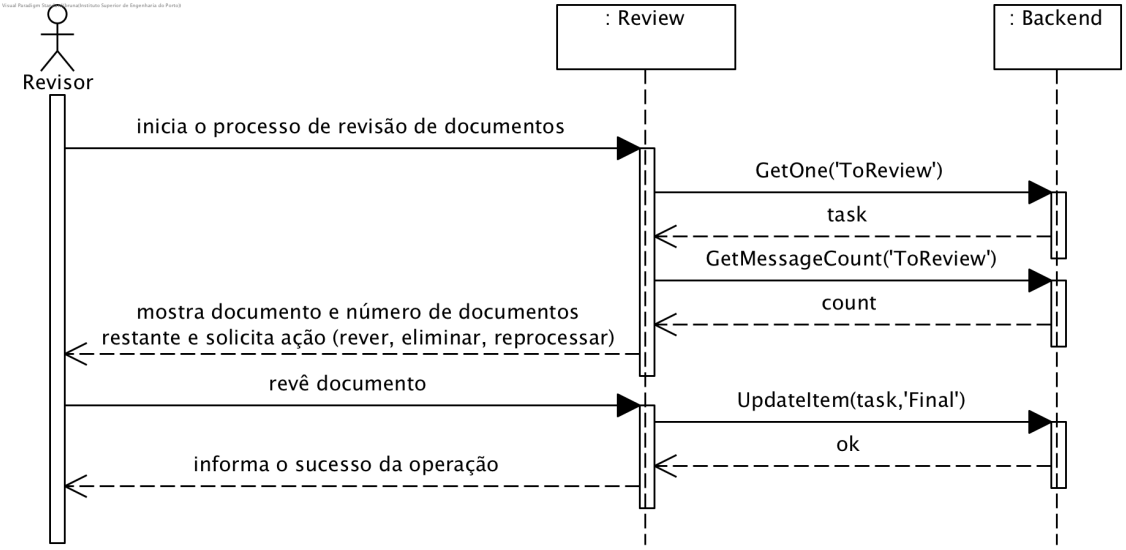

Figura 5.7: Vista de processos de nível 2 - Rever documentos

5.2.1.3 Desenho arquitetural de nível 3

5.2.1.3.1 Vista lógica

A vista lógica de nível 3 apresentada na figura 5.8 expõe os componentes pertencentes ao Backend:

- `Service` - componente que possui um conjunto de operações que são, por sua vez, disponibilizadas pelo Provider e a Storage. Este componente interage com as interfaces gráficas;
- `Provider` - implementa e suporta as operações do message bus, nomeadamente a inserção de dados nas queues;

- `Message Bus` - alberga as queues que guardam e reencaminham a informação pretendida;
- `Storage` - implementa e suporta as operações *Create, Read, Update, Delete* (CRUD) e tem acesso ao componente Database;
- `Database` - Sistema de Gestão de Base de Dados (SGBD) que persiste toda a informação relativa à aplicação. Disponibilizará uma interface, neste caso concreto SQL.

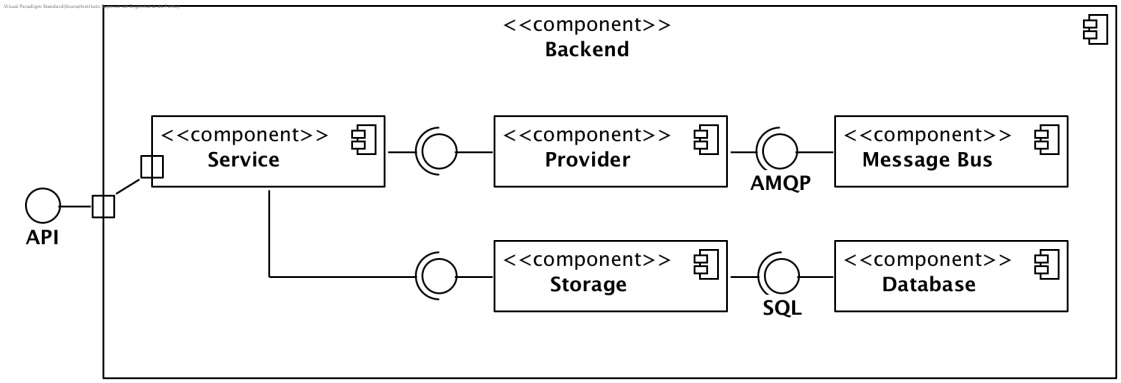

Figura 5.8: Vista lógica de nível 3

5.2.1.3.2 Vista de processos

Nas figuras seguintes: 5.9, 5.10, 5.11, encontram-se representadas as vistas de processos dos casos de uso mais relevantes.

Estes diagramas de sequência correspondem sequencialmente a:

- Digitalizar documentos;
- Processar documentos;
- Rever documentos.

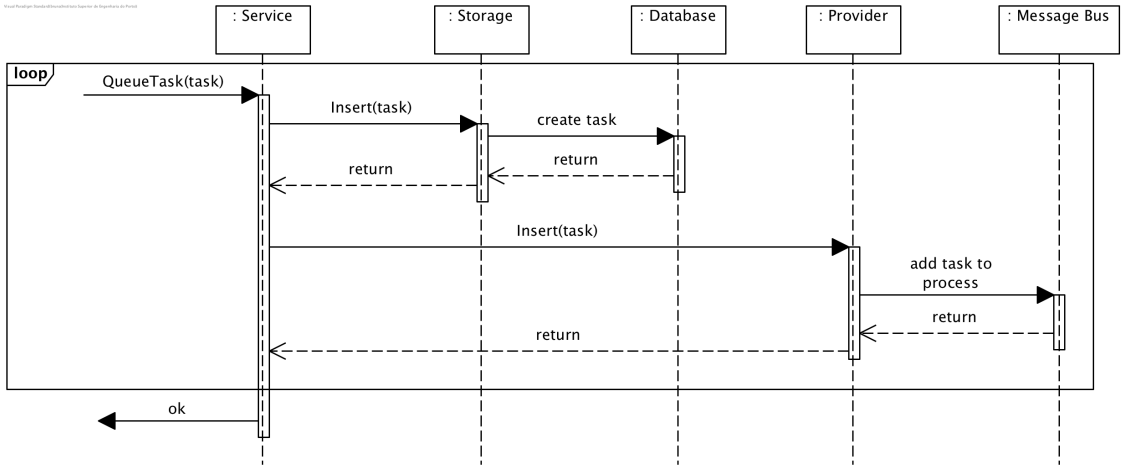

Figura 5.9: Vista de processos de nível 3 - Digitalizar documentos

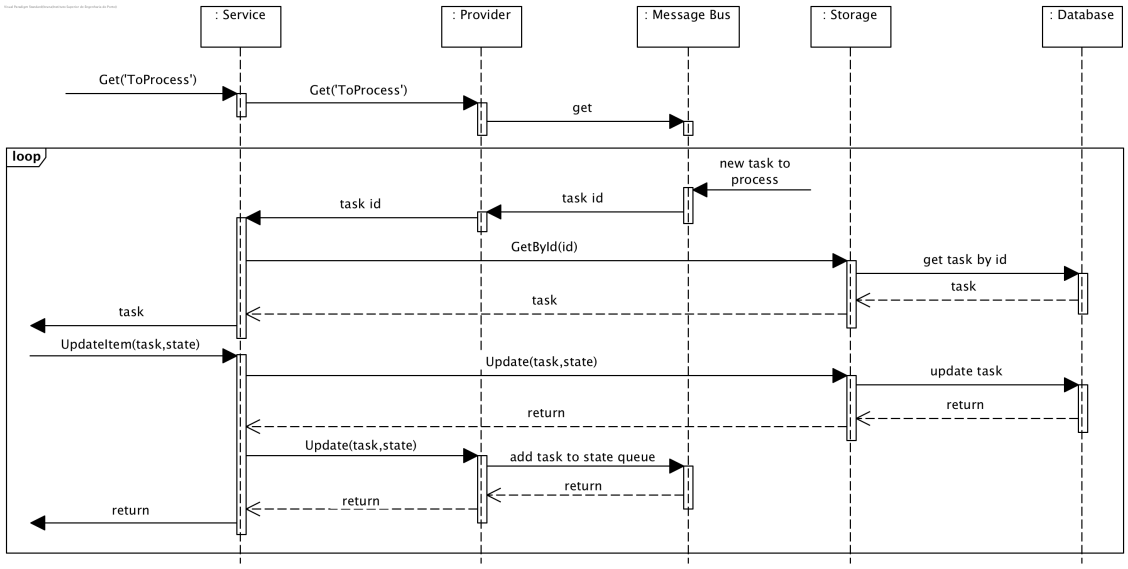

Figura 5.10: Vista de processos de nível 3 - Processar documentos

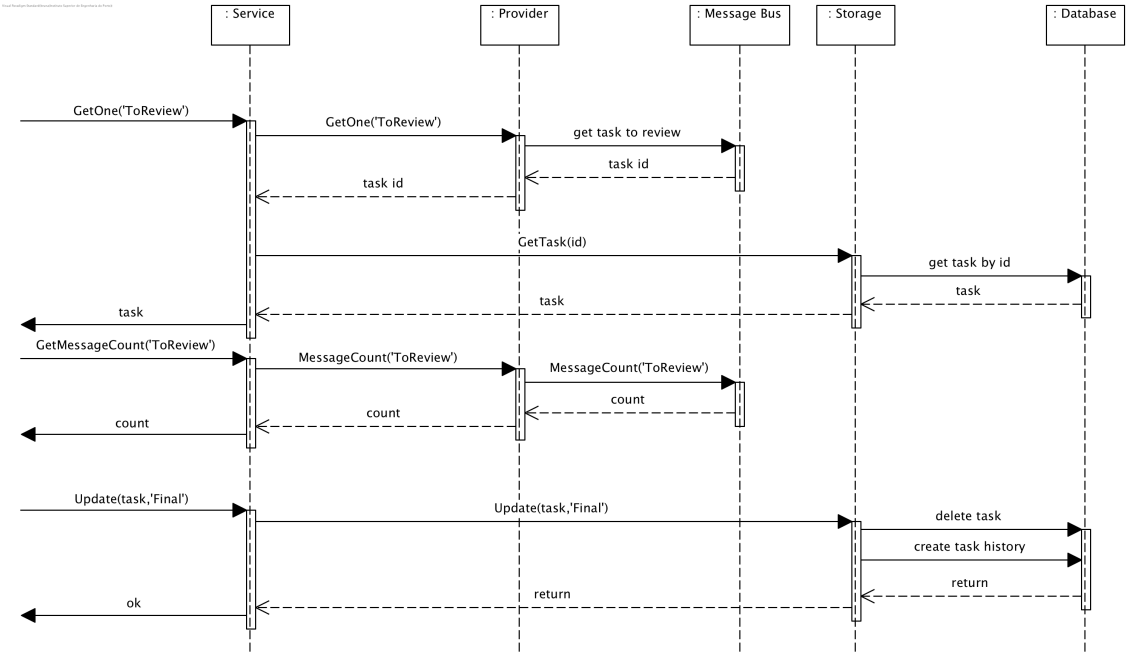

Figura 5.11: Vista de processos de nível 3 - Rever documentos

5.2.1.3.3 Vista de implantação

Na figura 5.12 apresenta-se o diagrama de implantação. No mesmo pode-se observar quatro nós distintos:

- `Windows` - máquina na qual estão a correr as aplicações;
- `CloudAMQP` - permite a implantação do *message bus* RabbitMQ;
- `Azure` - plataforma Microsoft Azure que possibilita a implantação do Azure Service Bus;

- `SQL Server` - nó no qual se encontra a base de dados.

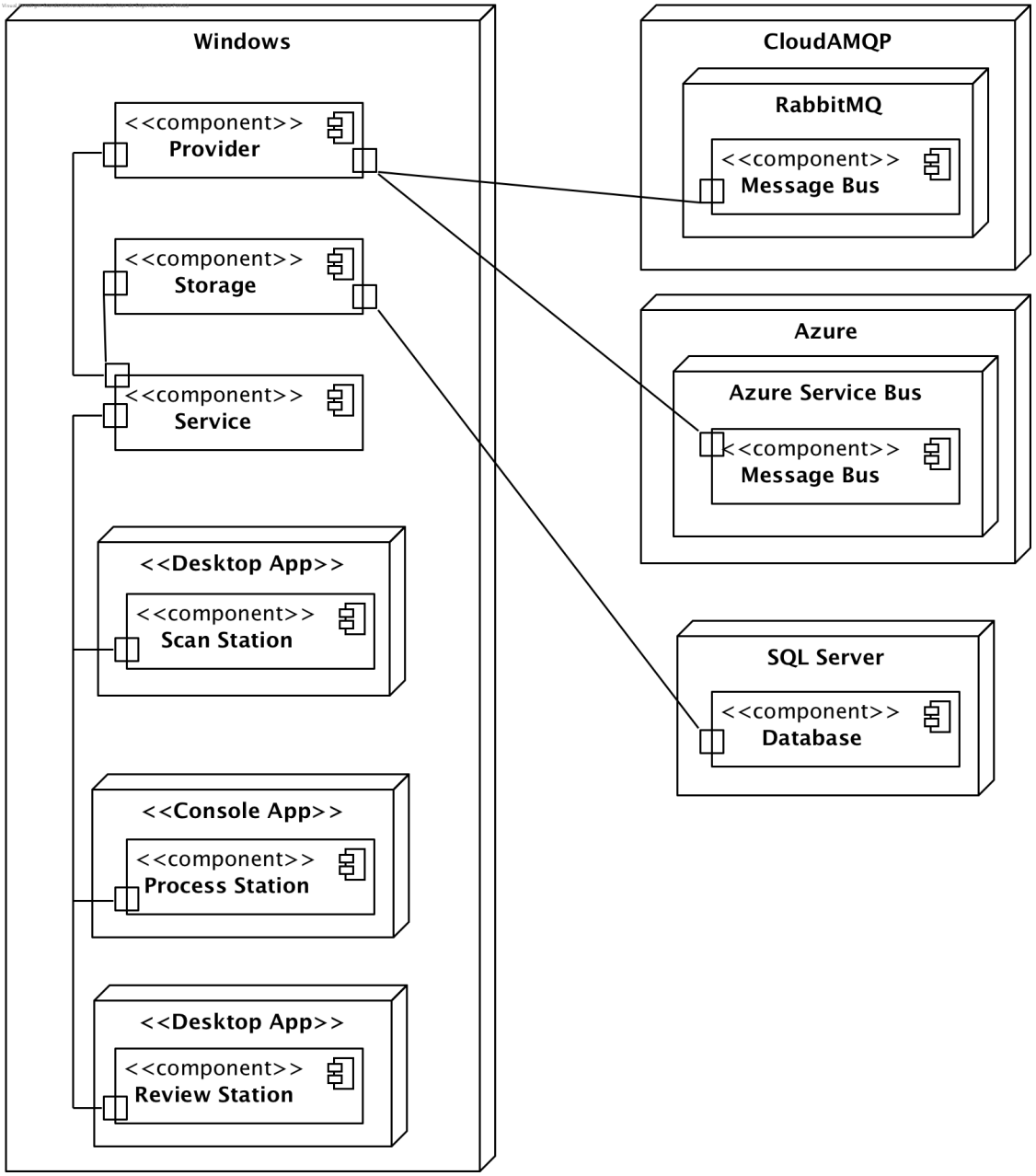

Figura 5.12: Vista de implantação de nível 3

5.2.2 Desenho arquitetural da solução generalizada

Na presente subsecção é apresentado o desenho arquitetural da solução generalizada nos diversos níveis.

5.2.2.1 Desenho arquitetural de nível 1

5.2.2.1.1 Vista lógica

Na figura 5.13 encontra-se o diagrama de componentes de alto nível do sistema generalizado.

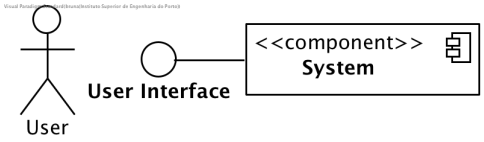

Figura 5.13: Vista lógica de nível 1

5.2.2.2 Desenho arquitetural de nível 2

5.2.2.2.1 Vista lógica

O diagrama de componentes da figura 5.14 corresponde à vista lógica de nível dois. O mesmo contém os componentes generalizados referidos no capítulo 3 com a distinção de a base de dados ser substituída por uma aplicação servidora.

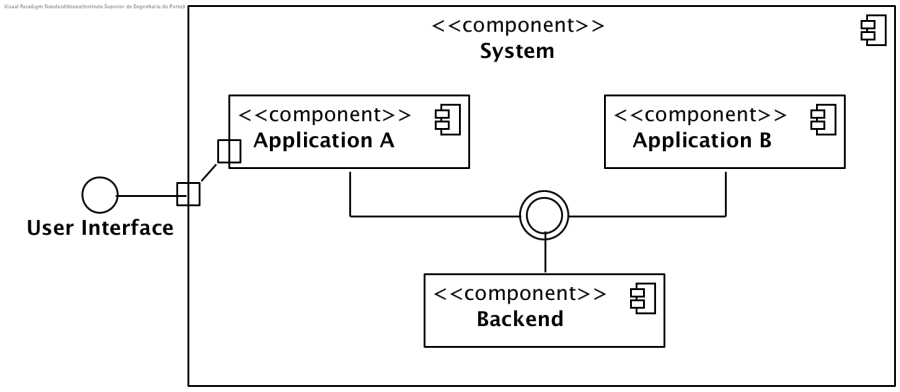

Figura 5.14: Vista lógica de nível 2

5.2.2.2.2 Vista de processos

Para uma melhor compreensão, foi convencionado que o sistema deve funcionar do modo apresentado na figura 5.15:

1. O utilizador interage com a UI e introduz a informação respetiva;
2. A UI envia os dados à aplicação A;
3. A aplicação A trata a informação;
4. A aplicação B tem como intuito tratar a informação resultante do ponto anterior e caso esta se encontre disponível realiza a sua função.

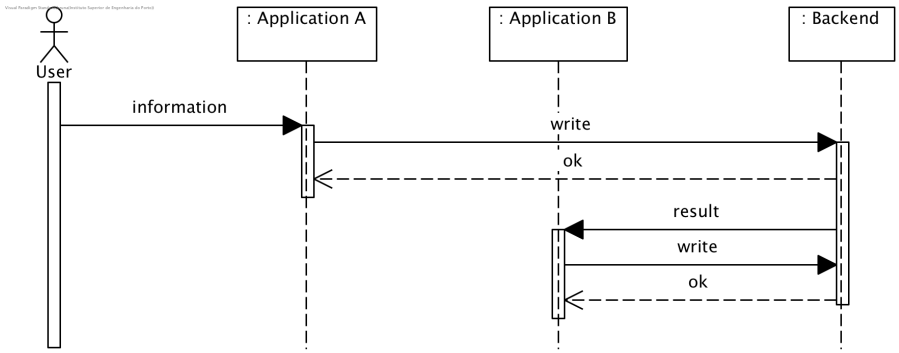

Figura 5.15: Vista de processos de nível 2

5.2.2.3 Desenho arquitetural de nível 3

5.2.2.3.1 Vista lógica

A vista lógica de nível 3 da figura 5.16 da solução generalizada é idêntica à que será implementada no caso do SmartDocumentor. De facto, foi pensado realizar a arquitetura desta forma para que os componentes relativos ao backend possam ser reutilizados noutras soluções.

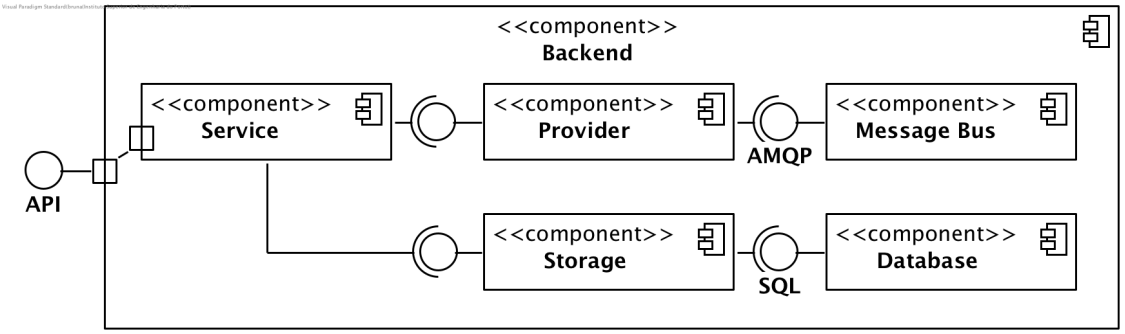

Figura 5.16: Vista lógica de nível 3

5.2.2.3.2 Vista de processos

Foram desenhadas duas alternativas para a vista de processos simplificada.

Na figura 5.17 é apresentada a alternativa correspondente a um processo orientado a mensagens com identificação, semelhante à solução a ser aplicada no SmartDocumentor. Desta forma, possui os seguintes passos:

1. O Service é requirido pela primeira aplicação;
2. O módulo Storage cria o item na base de dados;
3. A identificação do item é enviada para o módulo Provider e este introduz a informação na queue respetiva;
4. A segunda aplicação encontra-se à escuta e aquando do envio da identificação para a queue correspondente, a aplicação recebe o item respetivo.
5. A segunda aplicação altera o item e requere atualização por parte do módulo Storage;

6. A Storage atualiza o item na base de dados.

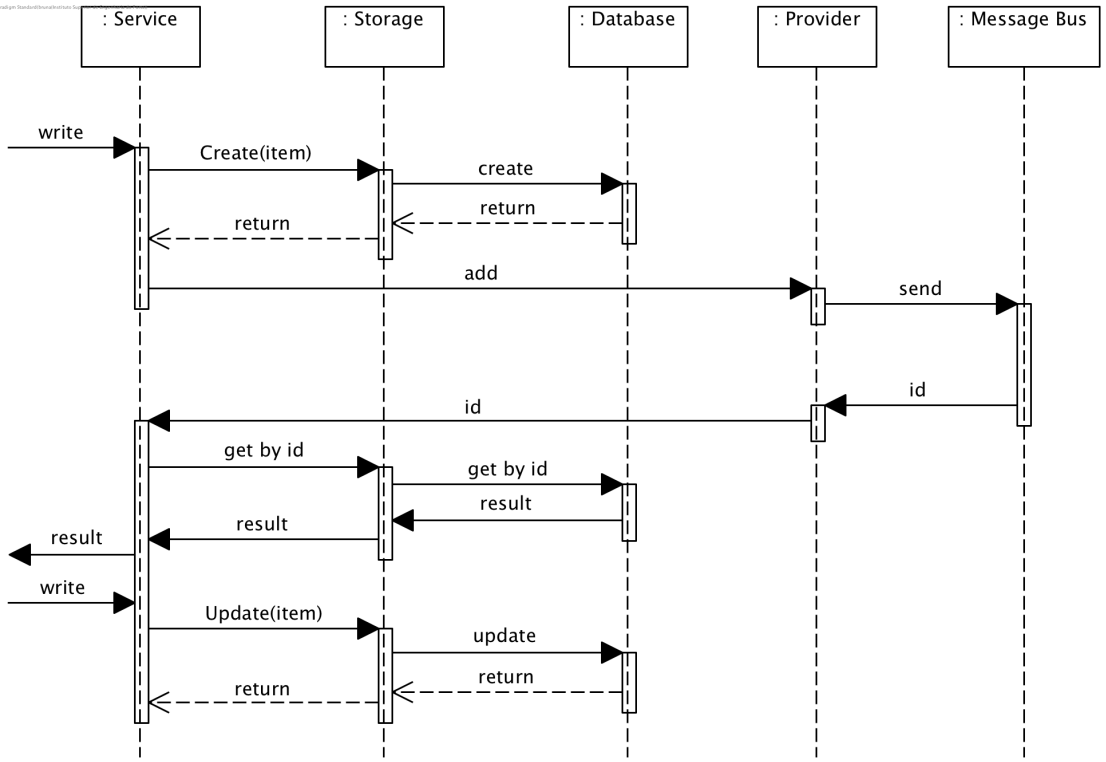

Figura 5.17: Vista de processos de nível 3 - Alternativa 1

Existe ainda a alternativa do uso de um processo orientado a mensagens, sendo o único papel da base de dados guardar os dados finais para posterior consulta. Assim sendo, este processo representado na figura 5.18 desenrola-se da seguinte forma:

1. O utilizador interage com a User Interface (UI) e introduz a informação respetiva;
2. A UI envia os dados à aplicação A;
3. A aplicação A trata a informação;
4. A aplicação B encontra-se à escuta da aplicação A com recurso a um sistema de mensagens como intermediário;
5. A aplicação A envia o objeto a ser tratado;
6. O sistema de mensagens envia à aplicação B o objeto;
7. A aplicação B trata os dados, atualizando os mesmos.

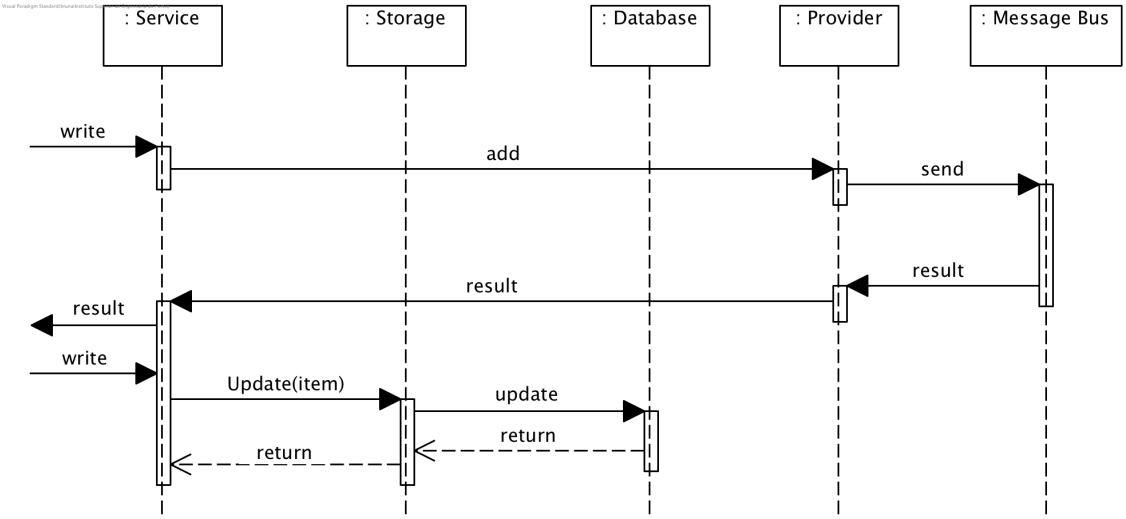

Figura 5.18: Vista de processos de nível 3 - Alternativa 2

5.2.2.3.3 Vista de implantação

Relativamente à vista de implantação da solução generalizada, foi construído o diagrama da figura 5.19. No mesmo observa-se três nós distintos:

- `Operating System` - sistema operativo no qual correm as aplicações desenvolvidas. Pode ser escolhido implantar as aplicações na nuvem invés de ser escolhida a opção *onpremise*;
- `Platform` - plataforma na qual é implantado dado *message bus*, dependendo a sua decisão das opções disponíveis para a tecnologia respetiva;
- `Database Server` - nó no qual é implantada a base de dados.

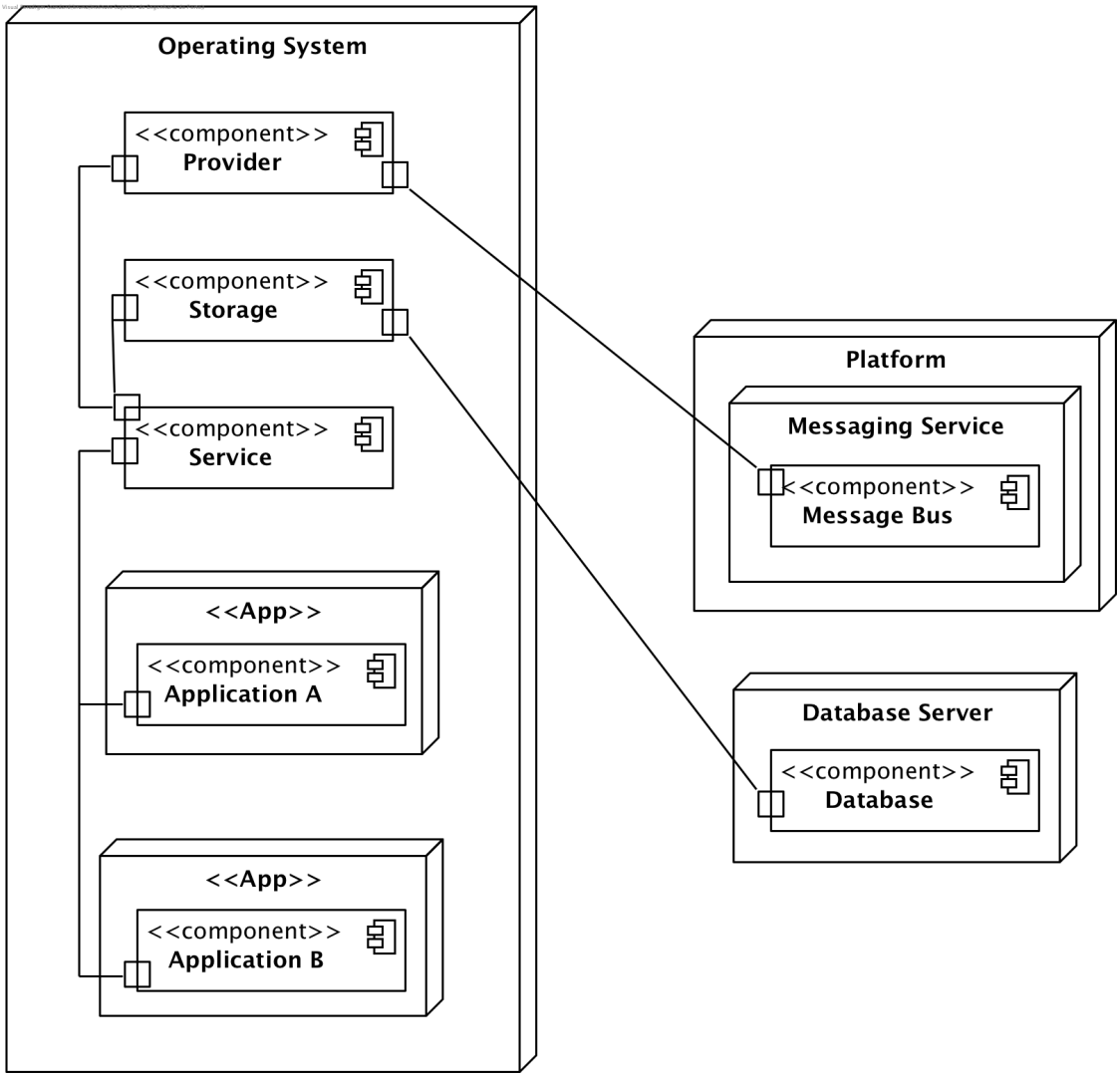

Figura 5.19: Vista de implantação de nível 3

5.3 Base de dados

No que diz respeito aos processos que necessitam de base de dados para o seu funcionamento, decidiu-se utilizar uma base de dados SQL Server. Esta decisão foi tomada dado que no sistema inicial o tipo da base de dados utilizada é também SQL Server.

Por outro lado, foi decidido manter a existência de uma base de dados por dois motivos:

- A necessidade de armazenar os dados finais;
- Para não aumentar a carga do sistema de mensagens, considerando que existem limitações a nível do número de caracteres que uma mensagem pertencente a uma queue pode possuir.

Na solução relativa ao SmartDocumentor foi utilizado o modelo de dados da figura 5.20.

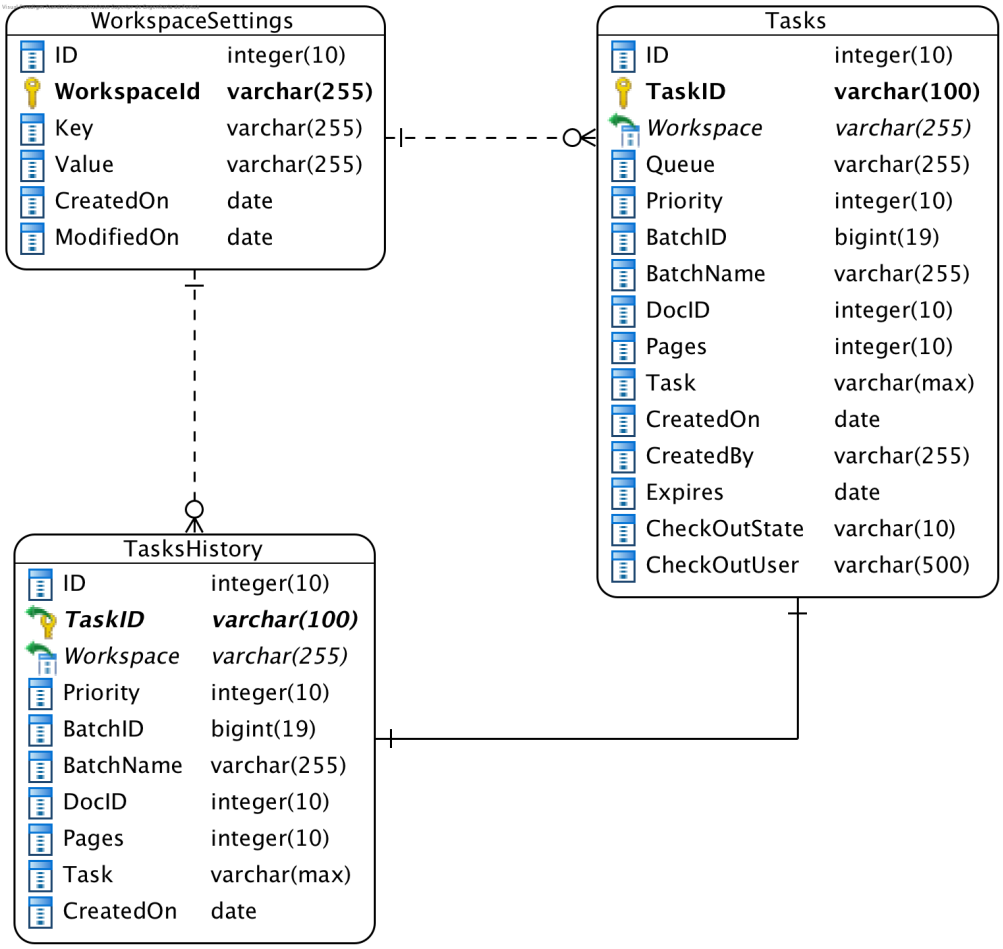

Figura 5.20: Modelo de dados do SmartDocumento

5.4 Resumo do capítulo

Primeiramente, foram apresentadas neste capítulo as abordagens alternativas de tecnologias e procedeu-se à seleção das mesmas com o auxílio do método AHP. Tendo em conta esta análise foi decidido adotar o Azure Service Bus e o RabbitMQ.

De seguida, foi realizado o desenho arquitetural da proposta de solução. Assim sendo, com o intuito de representar esta arquitetura, foram exibidos os diagramas necessários à construção da solução que se apresenta no capítulo seguinte.

Capítulo 6

Construção da solução

O presente capítulo tem como intuito a apresentação da construção da solução para o problema respetivo.

6.1 Detalhes da implementação

A decisão de realizar uma prova de conceito em detrimento da alteração direta SmartDocumentor foi tomada pela empresa considerando:

- A complexidade do produto;
- A alteração de substancial de arquitetura.

O desafio consistiu em realizar uma implementação abstrata da solução preconizada no capítulo 5, não dependendo o código chamado da tecnologia escolhida.

A sequência de chamadas encontra-se representada na figura 6.1:

- A estação contacta o serviço;
- O serviço utiliza os métodos do componente responsável pelas ações relativas ao message bus;
- O Provider usa os métodos do message bus respetivo.

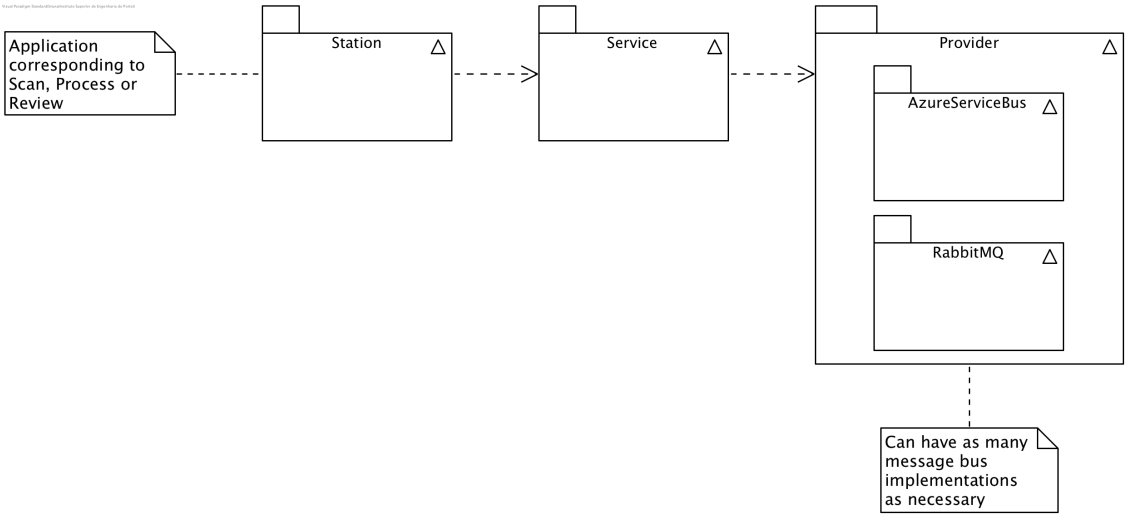

Figura 6.1: Vista arquitetural de implementação abstrata

O Provider contém as operações de acesso ao message bus. Para além disso, este componente foi construído pensando na abstração que devia existir na sua chamada. Posto isto, apresenta-se o diagrama de classes da criação de Provider da figura 6.2.

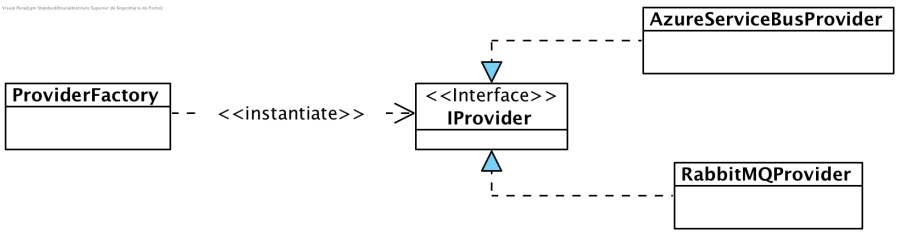

Figura 6.2: Implementação de Provider

Esta implementação pode ser reutilizada noutros projetos e há a possibilidade de acrescentar mais tipos de *message bus* devido ao seu nível de abstração.

6.2 Prova de conceito

Considerando que o SmartDocumentor corresponde à aplicação utilizada enquanto prova de conceito de implementação de arquitetura orientada a mensagens para comunicação entre componentes, nesta secção serão apresentadas parte das funcionalidades da estações respetivas, sendo que algumas imagens das restantes constam do anexo C.

6.2.1 Scan Station

Uma das partes fulcrais da aplicação é o envio de documentos por parte da Scan Station, representado na figura 6.3. Esta ação acontece após a sua digitalização e faz com que as tarefas sejam criadas e passadas para o próximo estado.

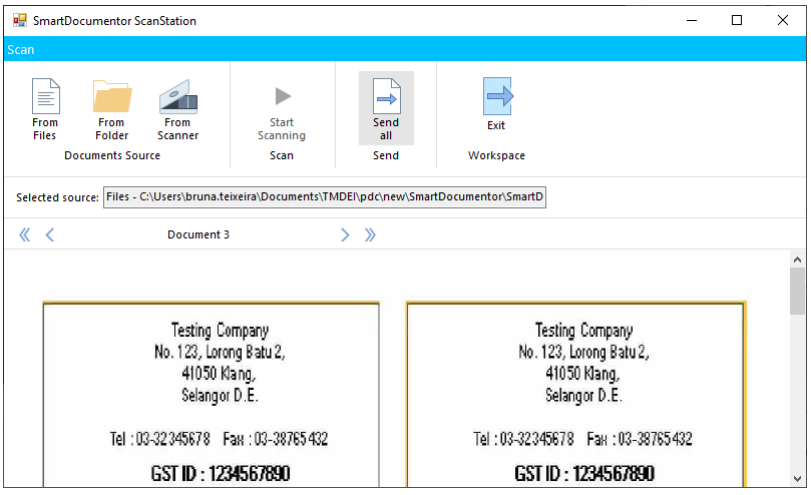

Figura 6.3: Scan Station - Enviar documentos

6.2.2 Process Station

A Process Station corresponde a um componente que se encontra à escuta de mensagens, recebendo conforme são enviadas as identificações dos documentos a ser processados.

Aquando da receção de uma identificação a estação trata do seu processamento e reencaminha para a fase seguinte que pode, por exemplo, corresponder à revisão do documento, tal como se pode observar na figura 6.4.

Para a demostração da prova de conceito foi decidido correr uma aplicação na consola. No entanto, o produto real possui um serviço Windows a correr no cliente enquanto Process Station.

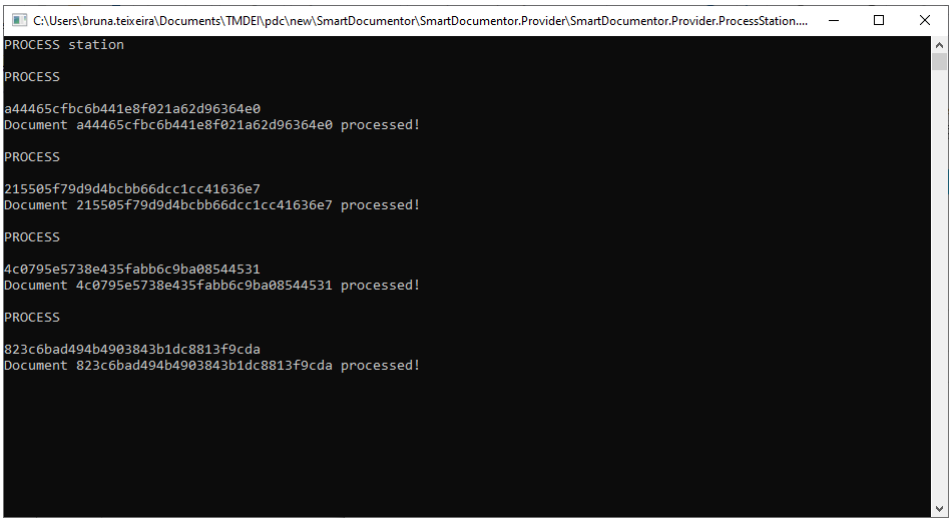

Figura 6.4: Process Station - Processar documentos

6.2.3 Review Station

A Review Station permite ao revisor rever os documentos. As identificações encontram-se na queue correspondente e cada um dos revisores vai efetuando a tarefa de revisão, podendo processar, eliminar ou pedir um novo processamento do documento atual. Para além disso, também é possível visualizar a quantidade de mensagens restantes na queue, facilitando a gestão do tempo do utilizador desta estação.

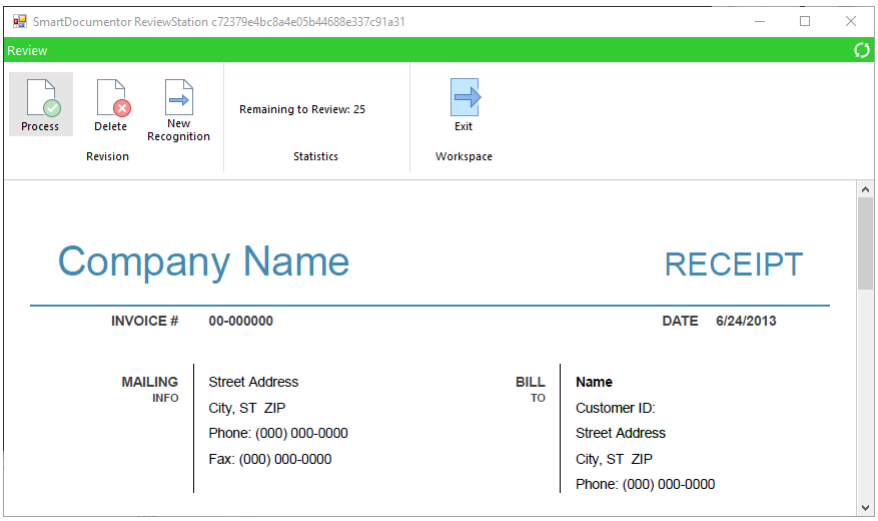

Figura 6.5: Review Station - Rever documento

6.3 Descrição de situações particulares

Nesta secção são expostos os detalhes da implementação das tecnologias Azure Service Bus e RabbitMQ no projeto.

Aquando do uso de *message broker* ou *message bus* pode-se recorrer ao uso de determinadas *frameworks* em .NET de modo a facilitar a implementação por parte do programador. Existem ferramentas para .NET que tornam a criação de aplicações e serviços que implementam comunicação baseada em mensagens mais simples. As três mais usadas são: MassTransit [70], NServiceBus [71] e EasyNetQ [72].

6.3.1 Azure Service Bus

O Azure Service Bus foi implantado na nuvem do Microsoft Azure recorrendo ao portal respetivo.

A monitorização do Portal do Microsoft Azure [73] foi observada de modo a verificar as métricas disponibilizadas por esta ferramenta.

Na figura 6.6 pode-se observar os detalhes de determinada queue, incluindo a quantidade de mensagens e respetivo estado e o espaço ocupado na queue.

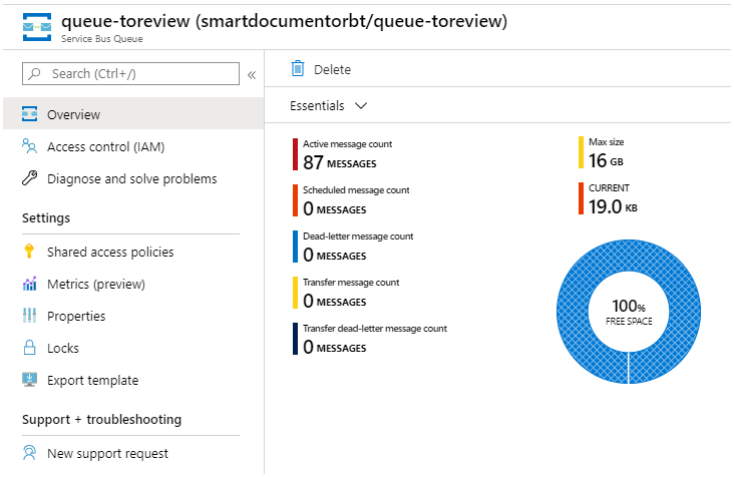

Figura 6.6: Azure Service Bus - Detalhes

No Portal Azure também é possível visualizar o fluxo de mensagens até 30 dias ou mesmo o total das mesmas que circularam no message bus, representado na figura 6.7.

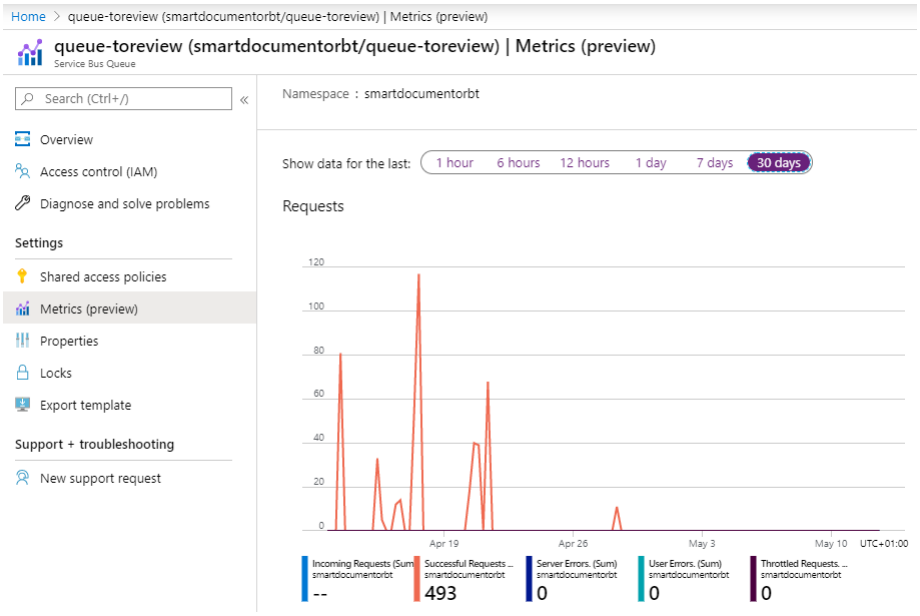

Figura 6.7: Azure Service Bus - Métricas

6.3.2 RabbitMQ

O RabbitMQ pode ser implantado em várias nuvens, tendo sido escolhida o serviço CloudAMQP [74] como auxiliar para o fazer na plataforma AWS.

Para a implementação usando o message bus RabbitMQ, procedeu-se à instalação do pacote NuGet da API de cliente EasyNetQ [75]. EasyNetQ corresponde a um projeto *open source* que pode implementar o padrão de consumidores concorrentes seguindo um tutorial inicial [76]. O restante código foi realizado posteriormente, tendo sido efetuada uma pesquisa intensiva de modo a cumprir os requisitos do sistema.

O RabbitMQ disponibiliza um serviço de monitorização para a gestão do message bus. Neste gestor, há a possibilidade de selecionar a queue pretendida e verificar quantas mensagens contém, representado na figura 6.8.

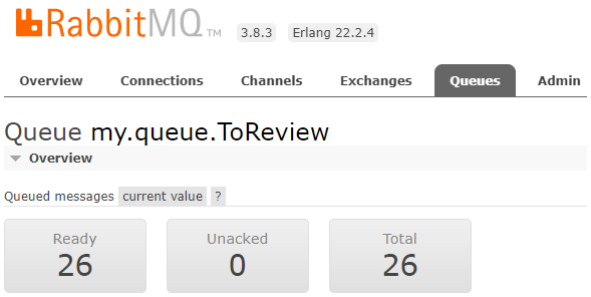

Figura 6.8: RabbitMQ - Detalhes

De entre outras funcionalidades, destaca-se o facto de ser possível verificar no gestor o fluxo de mensagens de uma queue. No entanto, só se encontram disponíveis os resultados de até uma hora antes do momento de visualização, tal como se pode observar na figura 6.9.

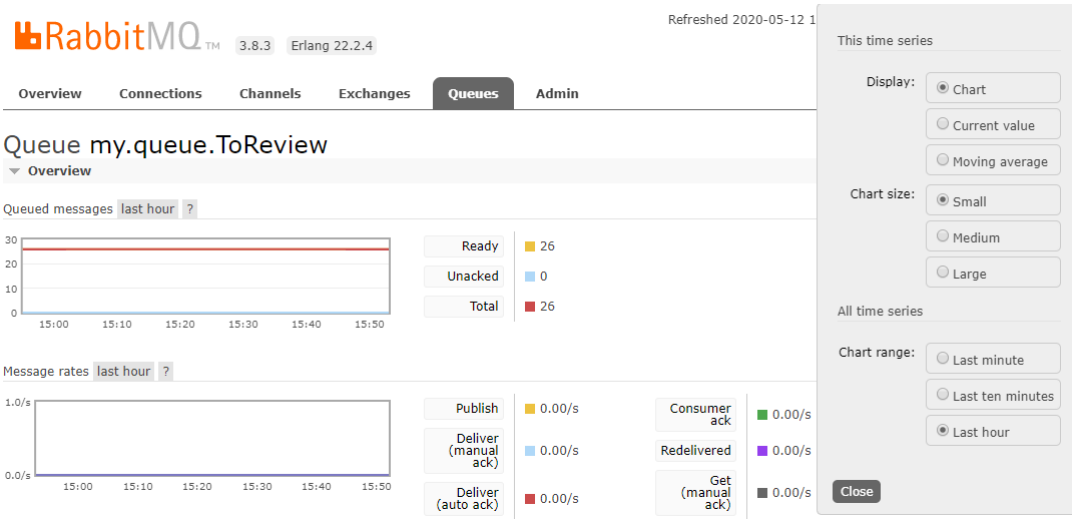

Figura 6.9: RabbitMQ - Métricas

6.4 Integração e entrega contínuas (CI/CD)

As técnicas de integração e entrega contínuas albergam um conjunto de princípios de operação que auxiliam o desenvolvimento de sistemas de *software*, tornando a tarefa de entrega de alterações de código mais frequente e confiável [77].

Continuous Integration (CI) corresponde a uma prática de desenvolvimento de *software* na qual os membros da equipa integram e combinam o trabalho realizado, nomeadamente código [77].

Continuous Delivery (CD) tem como intuito assegurar que a aplicação está pronta para ser lançada após passar os testes respetivos [77].

Assim sendo, de modo a garantir um ambiente de CI/CD foi decidido usar o serviço Azure DevOps [78].

Azure DevOps é uma plataforma Software as a Service (SaaS)[1] da Microsoft que fornece um conjunto de ferramentas, destacando-se para o projeto descrito neste documento Azure Repos [80] e Azure Pipelines [81].

O Azure Repos foi utilizado para alocar o repositório do projeto na nuvem. Por outro lado, foi criada a pipeline do sistema através do Azure Pipelines. A título de exemplo, encontra-se na figura 6.11 parte do código utilizado para gerar a pipeline do projeto da figura 6.10.

[1]SaaS corresponde a *software* como um serviço, sendo "um modelo de distribuição de *software* na qual um fornecedor externo aloca aplicações e torna-as disponíveis para os clientes através de Internet"[79].

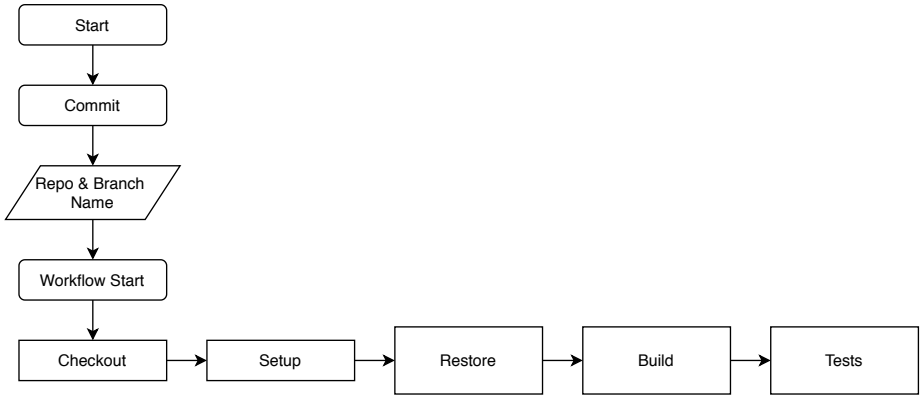

Figura 6.10: Pipeline

```
variables:
  solution: '**/*.sln'
  buildPlatform: 'Any CPU'
  buildConfiguration: 'Release'

steps:
Settings
- task: NuGetToolInstaller@1

Settings
- task: NuGetCommand@2
  inputs:
    restoreSolution: '$(solution)'

Settings
- task: VSBuild@1
  inputs:
    solution: '$(solution)'
    msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true
    platform: '$(buildPlatform)'
    configuration: '$(buildConfiguration)'
```

Figura 6.11: Excerto do código da pipeline

6.5 Documentação

Nesta secção apresenta-se a forma como o código do projeto foi documentado de modo a facilitar a sua legibilidade e consequente manutenibilidade.

6.5.1 Comentários XML

Foram adicionados comentários XML à maioria do código com o auxílio da ferramenta GhostDoc [82].

O GhostDoc é uma extensão do Visual Studio utilizada para produzir e manter os comentários XML do código, nomeadamente a descrição de classes, métodos e respetivas variáveis [82]. Depois de instalada, é usada, por exemplo, da forma descrita na figura 6.12 e gera comentários do género da figura 6.13.

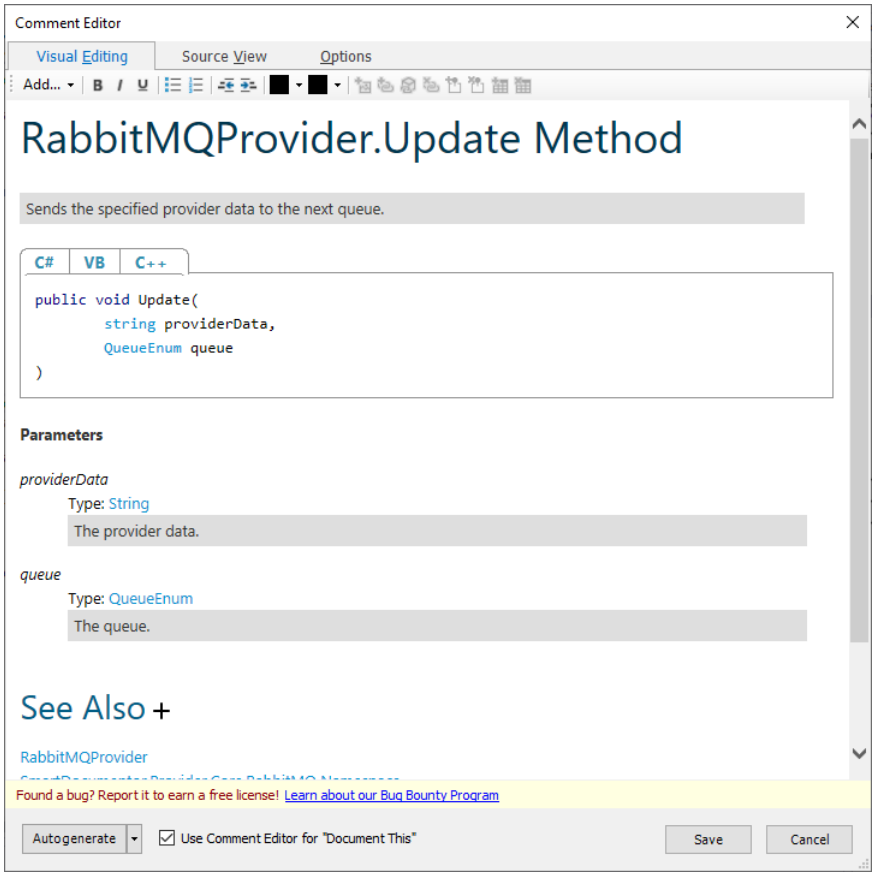

Figura 6.12: Uso da extensão GhostDoc

```
/// <summary>
/// Sends the specified provider data to the next queue.
/// </summary>
/// <param name="providerData">The provider data.</param>
/// <param name="queue">The queue.</param>
4 references | BrunaT, 21 days ago | 1 author, 2 changes
public void Update(string providerData, QueueEnum queue)
```

Figura 6.13: Comentários XML

6.5.2 Doxygen

O Doxygen é uma ferramenta empregue para gerar a documentação de um projeto a partir dos seus comentários. A mesma suporta várias linguagens de programação, nomeadamente C++, C, C# e PHP, Java [83].

Após a instalação do programa foi seguido um tutorial [84], tendo-se obtido um HTML da documentação cuja página de exemplo se apresenta na figura 6.14.

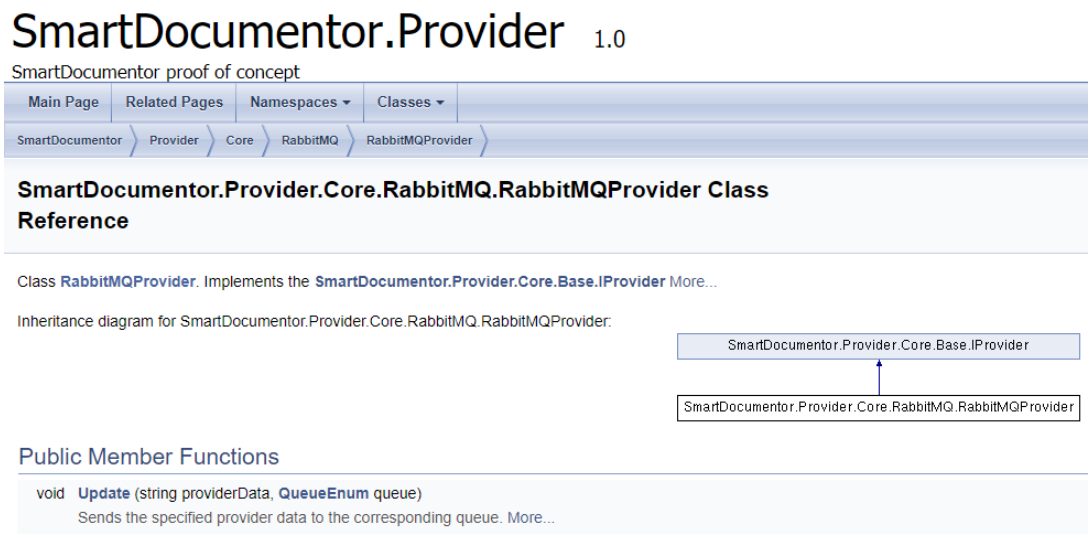

Figura 6.14: Página de documentação

6.6 Análise de código

Nesta secção são apresentadas as ferramentas de análise usadas no presente projeto tendo em conta o uso do Ambiente de Desenvolvimento Integrado (IDE) Visual Studio 2019. As mesmas ajudam na validação do código e da respetiva estrutura, detetando defeitos que possam existir.

Portanto, decidiu-se utilizar o **FxCop** [85] e o **StyleCop** [86].

O FxCop verifica erros no código relativos, nomeadamente, a segurança, desempenho e desenho [85].

O StyleCop encontra erros de estilo no código de um projeto [86].

A ferramenta FxCop foi usada no projeto através da instalação do pacote que contém as regras mais importantes Microsoft.CodeAnalysis.FxCopAnalyzers [85].

Por outro lado, o StyleCop foi incorporado pela instalação do pacote StyleCop.Analyzers, tendo sido posteriormente adicionado o ficheiro `stylecop.json` da figura D.6 do anexo D com as regras de análise.

O FxCop e o StyleCop são de fácil uso, sendo possível visualizar as infrações como descrito na figura 6.15.

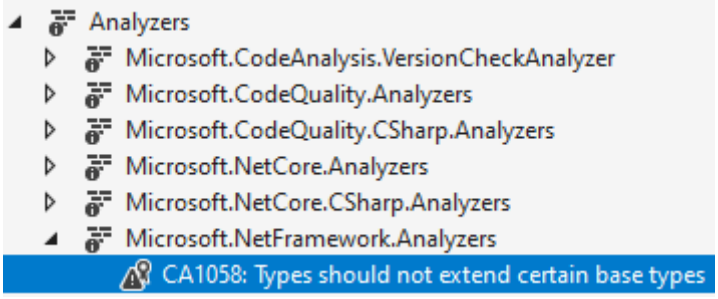

Figura 6.15: Visualizar infrações usando as ferramentas de análise

De seguida, selecionando as propriedades da violação pretendida, pode-se observar as suas características, incluindo a gravidade e a ligação onde se constata a forma de o resolver, representado na figura 6.16.

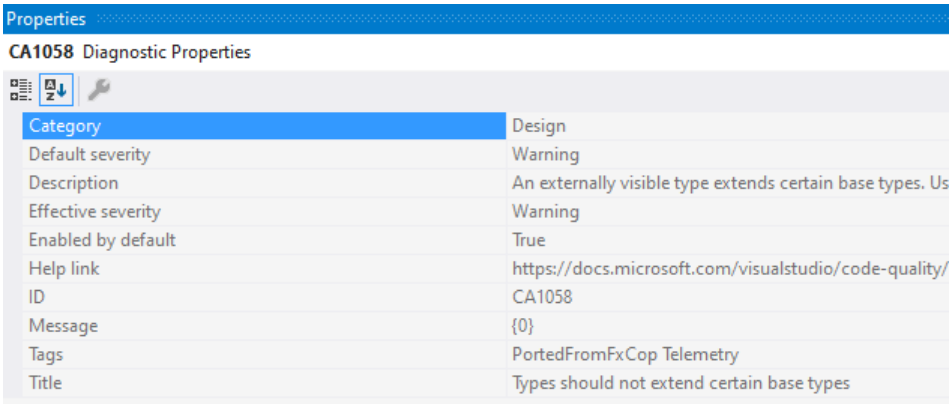

Figura 6.16: Propriedades de uma violação

Portanto, as ferramentas de análise de código facultaram a revisão do código do projeto.

6.7 Testes

Os testes são um passo importante para assegurar que o sistema de *software* satisfaz as necessidades do utilizador [87].

Uma vez que foi seguido o modelo representado na figura 6.17, a fase de testes sucedeu à fase da implementação.

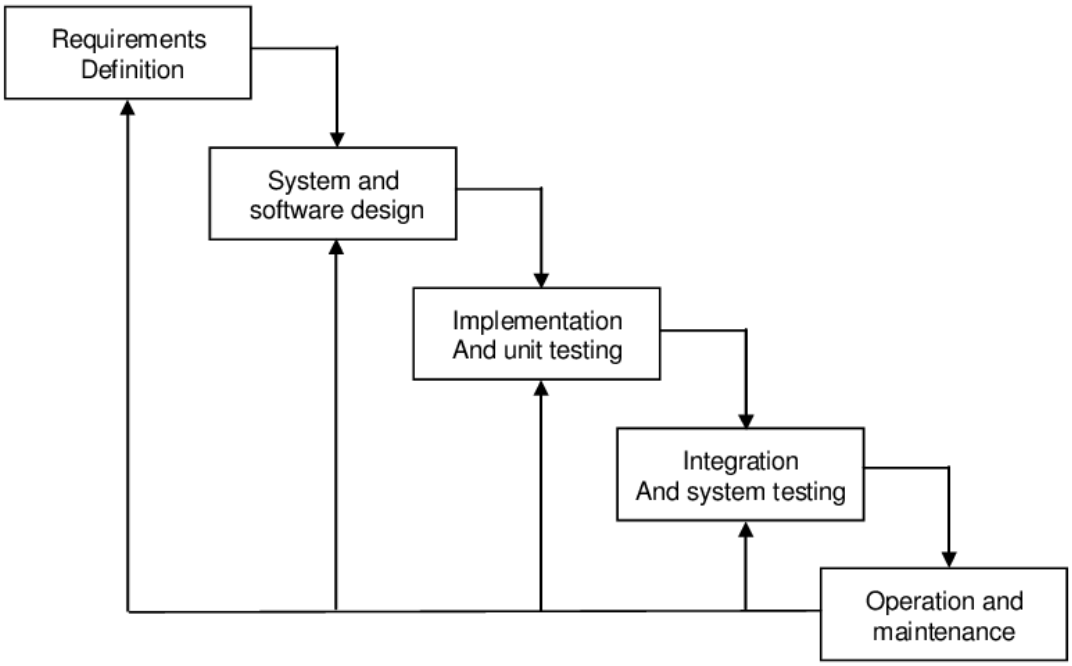

Figura 6.17: Modelo em cascata
Fonte: [88]

Portanto, foram realizados no âmbito do projeto descrito neste documento os diversos tipos de teste:

- Testes unitários;
- Testes de integração;
- Testes de sistema;
- Testes de aceitação.

6.7.1 Testes unitários

Um teste unitário testa determinada unidade de código isoladamente e verifica se foi obtido o resultado pretendido [89].

Assim sendo, foram realizados testes unitários, após a escrita do código, aos vários componentes do backend do projeto, nomeadamente: Provider, Storage e Service.

Na figura 6.18 apresenta-se um exemplo de teste unitário.

```
[TestMethod]
0 references | BrunaT, 6 hours ago | 1 author, 1 change
public void DeleteItem_ItemNotExists()
{
    //check if throws sql exception
    Assert.ThrowsException<SqlException>(() => storage.Delete("1"));
}
```

Figura 6.18: Exemplo de teste unitário

Relativamente à cobertura, foram obtidos os resultados da figura 6.19, na qual as colunas correspondem sequencialmente a:

- Módulo do sistema;
- Percentagem de blocos não testados;
- Percentagem de blocos testados;
- Percentagem de linhas não testadas;
- Percentagem de linhas testadas.

smartdocumentor.provider...	5.14%	94.18%	93.38%	6.62%
smartdocumentor.services....	5.71%	91.43%	93.02%	6.98%
smartdocumentor.storage....	53.29%	46.71%	48.39%	51.61%

Figura 6.19: Cobertura de testes dos módulos

Observando os resultados anteriores, pode-se concluir que foi atingido uma percentagem razoável de testes unitários ao código realizado no âmbito do backend.

6.7.2 Testes de integração

Os testes de integração são realizados após os testes unitários de forma a testar interações entre os componentes. O intuito deste nível de testes é expor falhas que possam existir na combinação dos testes enquanto grupo [90].

Portanto, efetuou-se testes relativos à interação entre o componente Service e os componentes Provider e Storage, comparando os resultados obtidos em ambos os casos. Na figura 6.20 apresenta-se um desses testes.

```
[TestMethod]
0 references | 0 changes | 0 authors, 0 changes
public void MessageCount_Integration()
{
    // Arrange
    var provider = ProviderFactory.CreateProvider();
    var service = ServiceFactory.CreateService();
    // Act
    var expectedResult = provider.MessageCount(QueueEnum.ToProcess);
    var result = service.MessageCount(QueueEnum.ToProcess);
    // Assert
    Assert.IsNotNull(expectedResult);
    Assert.IsNotNull(result);
    Assert.AreEqual(expectedResult, result);
}
```

Figura 6.20: Exemplo de teste de integração

6.7.3 Testes de sistema

Os testes de sistema são realizados, tal como o próprio nome indica, ao sistema completo, verificando a sua concordância com os requisitos específicos [91].

Posto isto, foram realizados testes de sistema às várias funcionalidades do *software*, nomeadamente à criação e envio de documentos digitalizados, tal como apresentado na figura 6.21.

```
[TestMethod]
0 references | BrunaT, 21 days ago | 1 author, 1 change
public void QueueTask_QueueExists()
{
    var tasks = new SDTask
    {
        TaskId = Guid.NewGuid().ToString("N"),
        CreatedOn = DateTime.UtcNow,
        CreatedBy = $"{Environment.UserDomainName}\\{Environment.UserName}",
        DocumentId = Guid.NewGuid().ToString("N")
    };
    tasks.MetaData.Add("TESTES", "TESTE");
    tasks.MetaData.Add("_OriginalFileName", "OriginalFileName.pdf");
    tasks.Workspace = "DO";
    tasks.Priority = 5;
    tasks.Priority = 5;
    tasks.BatchId = 2003030007;
    tasks.BatchName = "DO_2003030007";
    tasks.DocId = 6;
    tasks.Pages = 5;

    //check if runs without exception
    service.QueueTask(tasks);
}
```

Figura 6.21: Exemplo de teste de sistema

6.7.4 Testes de aceitação

Os testes de aceitação "possibilitam ao utilizador, cliente ou outra entidade autorizada determinar se o produto deve ser aceite"[92].

Tendo em conta que os testes de aceitação são realizados com base nos requisitos, foi elaborada a tabela 6.1.

Tabela 6.1: Testes de aceitação

Requisito	**Resultado**
A Scan Station envia a identificação dos documentos digitalizados para determinada queue.	Cumpre
A Process Station recebe as identificações dos documentos a processar e processa-os, reencaminhando de seguida as identificações respetivas em caso de sucesso para a próxima queue.	Cumpre
A Review Station recebe da queue a identificação do próximo documento a rever.	Cumpre
É possível visualizar na Review Station a quantidade de documentos a rever, ou seja, quantos elementos constam da queue correspondente.	Cumpre

6.8 Resumo do capítulo

No presente capítulo procedeu-se à apresentação da construção da solução preconizada no capítulo 5. Como tal, abordou-se:

- Os detalhes da implementação;
- O funcionamento do sistema da prova de conceito (SmartDocumentor);
- As situações particulares relativas ao Azure Service Bus e ao RabbitMQ, nomeadamente a nível de monitorização;
- As técnicas de integração e entrega contínuas utilizadas;
- A forma como foi gerada a documentação do código do projeto;
- As ferramentas de análise de código utilizadas;
- Os testes efetuados ao sistema de *software*.

O próximo capítulo tem como foco as experiências e consequente avaliação realizada à solução desenvolvida tendo em conta a fase de construção da solução.

Capítulo 7

Experiências e Avaliação

Este capítulo foca-se na conceção do processo e avaliação da solução desenvolvida.

7.1 Objetivo

O objetivo da avaliação consistiu em avaliar o resultado do trabalho desenvolvido ao longo do projeto.

O aperfeiçoamento do desempenho de programas é o processo de alterar o sistema de *software* para que funcione mais eficientemente e execute mais rápido ou verificando a quantidade de redução de trabalho. Esta melhoria pode ser efetuada monitorizando e analisando uma aplicação e identificando formas de a melhorar. [93]

A alteração do sistema de *software* foi realizada em vários aspetos:

- Diversas alternativas a nível de *design*, pois o desenho de um sistema pode ser melhorado tendo em conta os recursos disponíveis, desempenhando a arquitetura um papel fulcral. Relativamente às arquiteturas, determinou-se verificar se a solução orientada a mensagens apresenta efetivamente melhor desempenho que o sistema inicial. A avaliação de desempenho será realizada em termos de tempo;
- As tecnologias Azure Service Bus e o RabbitMQ, no âmbito da solução orientada a mensagens no que se refere aos indicadores da secção seguinte.

7.2 Indicadores e fontes de informação

Para cada um dos indicadores - desempenho (tempo, tempo de trabalho), monitorização, implantação na nuvem e possibilidades de implementação - foram testadas as hipóteses de forma a avaliar e comparar as soluções.

Apesar de os testes serem semelhantes não são exatamente iguais, estando as fontes de informação dependentes da grandeza:

- Tempo, duração através de testes recorrendo ao JMeter;
- Tempo de trabalho, averiguar duração e número de passos para a construção de cada uma das soluções;
- Monitorização, verificar quantidade de informação disponível;
- Implantação na nuvem, aferir facilidade e tempo de implantação;

- Possibilidades de implantação, analisar a quantidade de opções existente.

7.3 Metodologia de avaliação

Na metodologia esteve envolvida a pessoa que desenvolveu o projeto e foram seguidos os passos:

- Reprodução simplificada do sistema inicial;
- Desenvolvimento do sistema referente à solução pretendida com as opções Azure Service Bus e RabbitMQ;
- Verificado para cada uma das soluções o resultado das grandezas, averiguando a duração, número de passos ou quantidade de opções disponíveis;
- Comparação de resultados.

7.4 Descrição das experiências

7.4.1 Tempo

Relativamente ao tempo, foi decidido realizar testes estatísticos, tendo-se escolhido o teste paramétrico t-test ou o teste de Wilcoxon nos momentos de comparação de duas amostras.

No âmbito do teste a duas variáveis independentes foram seguidos os seguintes passos:

- Obter amostra de 30 execuções de dada variável independente, sendo transmitidos exatamente os mesmos dados e em cada tentativa anotado o tempo de execução;
- Efetuar testes Shapiro-Wilk de modo a verificar se as amostras seguem uma distribuição normal;
- Caso não se verifique distribuições normais será utilizado o t-test, caso contrário o teste de Wilcoxon;
- Efetuar o t-test/Wilcoxon.

A tecnologia utilizada para a realização do projeto foi o .NET, pelo que foi efetuado o estado de arte em tecnologias de testes relevantes para o problema em questão. Existem diversas ferramentas *open-source* grátis cujo foco é testar o desempenho das aplicações pretendidas, de entre as mesmas destacam-se: Locust.io [94], Bees with Machine Guns [95], Multi-Mechanize [96], Siege [97], Apache Bench [98], Httperf [99], JMeter [100], Google PageSpeed Insights [101], Sitespeed.io [102], PageSpeed Module [103] e WebPagetest.org [104].

Para o caso concreto descrito neste documento, escolheu-se o JMeter. Portanto, primeiramente foram obtidas através da utilização da ferramenta JMeter três amostras de 30 execuções, cujo resumo de resultado pode ser visualizado na subsecção E.1.1 do anexo E, tendo sido produzido um ficheiro CSV semelhante ao excerto da figura 7.1 para cada um dos casos:

- Solução inicial com base de dados partilhada e *polling*;
- Solução com recurso a Azure Service Bus implantado no Microsoft Azure;

- Solução com recurso a RabbitMQ implantado no CloudAMQP.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	timeStamp,elapsed,label,responseCode,responseMessage,threadName,dataType,success,failureMessage,bytes,sentBytes,grpThreads,allThreads,URL,Latency,IdleTime,Connect													
2	1587980824951,3837,HTTP Request,200,OK,Thread Group 1-8,text,true,,293,132,30,30,https://localhost:44355/api/BasicSql,3837,0,1525													
3	1587980824939,3849,HTTP Request,200,OK,Thread Group 1-1,text,true,,293,132,30,30,https://localhost:44355/api/BasicSql,3849,0,1590													
4	1587980824982,3806,HTTP Request,200,OK,Thread Group 1-9,text,true,,293,132,30,30,https://localhost:44355/api/BasicSql,3806,0,1540													

Figura 7.1: Excerto de ficheiro CSV

De seguida, os dados foram lidos no RStudio [105] recorrendo à linguagem R [106], e efetuou-se testes Shapiro-Wilk em R da forma apresentada na figura 7.2. Posto isto, para cada um ficheiros CSV:

- Armazenou-se a informação do mesmo numa estrutura;
- Efetuou-se o teste Shapiro-Wilk para a coluna relativa ao tempo da estrutura anterior.

```
sqlSpeeds <- read.csv(file = '/Users/brunamoreirat/Documents/MEI/TMDEI/Performance/sql.csv')
shapiro.test(sqlSpeeds$elapsed)
```

Figura 7.2: Exemplo de teste Shapiro-Wilk

Conclui-se através destes testes que as amostras não seguiam uma distribuição normal.

Um teste de hipóteses apoia ou não uma hipótese nula (H_0), isto é, o estado atual, comparativamente a uma hipótese alternativa (H_1) [107].

Os procedimentos de testes de hipóteses determinam se dada hipótese nula deve ou não ser rejeitada.

Em todos os testes estatísticos preconizou-se que a hipótese nula é rejeitada caso o *p-value* seja inferior a 5%.

7.4.1.1 Azure Service Bus implantado no Microsoft Azure/RabbitMQ implantado no CloudAMQP

$$H_0 : \mu_1 = \mu_2$$

$$H_1 : \mu_1 < \mu_2$$

Dada a natureza da normalidade das amostras, foi realizado o teste de Wilcoxon da figura 7.3.

```
> #asb vs rmq
> res1 <- wilcox.test(asbSpeeds$elapsed, rmqSpeeds$elapsed)
> res1

        Wilcoxon rank sum exact test

data:  asbSpeeds$elapsed and rmqSpeeds$elapsed
W = 253, p-value = 0.003194
alternative hypothesis: true location shift is not equal to 0
```

Figura 7.3: Teste de Wilcoxon - Azure Service Bus/RabbitMQ

Uma vez que o valor do *p-value* obtido foi inferior a 5%, o Azure Service Bus implantado no Microsoft Azure foi considerada uma opção melhor do que o RabbitMQ implantado no CloudAMQP em termos de tempo de resposta.

7.4.1.2 Solução proposta/Sistema inicial

$$H_0 : \mu_1 = \mu_2$$

$$H_1 : \mu_1 < \mu_2$$

Relativamente à comparação de tempo de resposta foi efetuado o teste de Wilcoxon da figura 7.4, dado que as amostras não seguem uma distribuição normal.

```
> #asb vs sql
> res2 <- wilcox.test(asbSpeeds$elapsed, sqlSpeeds$elapsed)
> res2

        Wilcoxon rank sum exact test

data:  asbSpeeds$elapsed and sqlSpeeds$elapsed
W = 90, p-value = 6.637e-09
alternative hypothesis: true location shift is not equal to 0
```

Figura 7.4: Teste de Wilcoxon - Azure Service Bus/RabbitMQ

A hipótese nula foi rejeitada, ou seja, as arquiteturas orientadas a mensagens podem ser efetivamente melhores do que as baseadas em base de dados partilhada com *polling* na grandeza de tempo.

7.4.1.3 Conclusão

Tal como foi verificado anteriormente, o desempenho da solução proposta é superior ao do sistema inicial.

De forma a comparar as três opções anteriores: base de dados partilhada SQL com *polling*, Azure Service Bus implantado no Microsoft Azure e RabbitMQ implantado no CloudAMQP, construiu-se no RStudio o *boxplot* da figura 7.5 que mostra a distribuição dos dados.

Posto isto, tal como verificado através dos testes de hipóteses, o *boxplot* confirma que a solução com melhor performance é a relativa ao Azure Service Bus implantado no Microsoft Azure, seguida do RabbitMQ implantado no CloudAMQP e por fim SQL Server com *polling*.

7.4.2 Tempo de trabalho

Quanto ao Azure Service Bus, foi utilizada maioritariamente a informação disponível no site da Microsoft. A sua implementação procedeu-se de um modo mais complexo, pois os tutoriais oficiais não preenchem todos os requisitos procurados ao funcionamento correto do sistema.

O RabbitMQ é o *message broker open-source* mais utilizado, pelo que foi simples encontrar informação útil relativa ao mesmo. Quanto à implementação foi utilizado um pacote que tornou a mesma rápida e intuitiva.

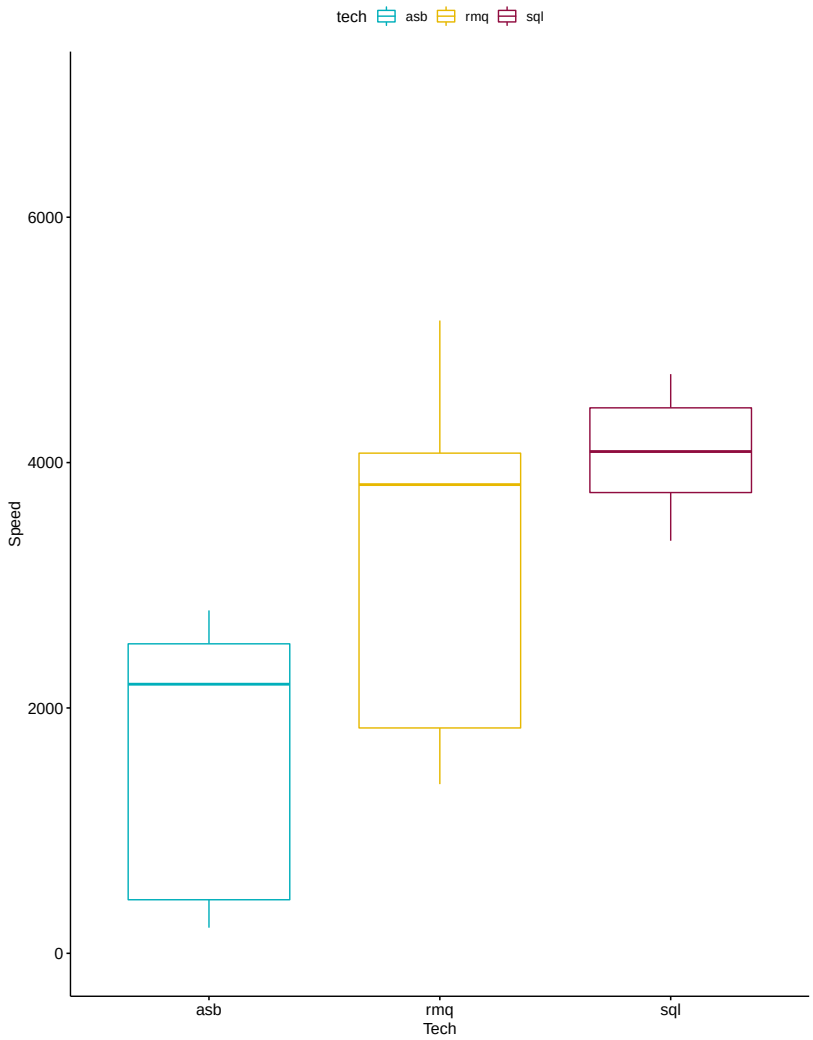

Figura 7.5: *Boxplot* de distribuição de dados

Posto isto, relativamente ao tempo aplicado em pesquisa e respetiva implementação, verificou-se que o processo do Azure Service Bus foi mais moroso quando comparado ao RabbitMQ.

7.4.3 Monitorização

No que diz respeito à monitorização, foi efetuada pesquisa de forma a comparar as tecnologias Azure Service Bus e RabbitMQ.

A verificação deste indicador pode ser observada na secção 6.3 do capítulo 6.

De facto, da análise anteriormente efetuada, chegou-se à conclusão que a nível de monitorização o Azure Service Bus é superior ao RabbitMQ.

7.4.4 Implantação na nuvem

O Azure Service Bus pertence à Microsoft e, portanto, a implantação pode ser realizada no Microsoft Azure. Esta implantação é intuitiva e pode ser realizada de várias formas simples e eficazes.

No caso do RabbitMQ, existem várias plataformas com este intuito, porém esta deve ser realizada nos *websites* disponíveis pelas mesmas e por vezes a forma não é tão intuitiva quanto se desejaria para o desenvolvedor.

Assim sendo, relativamente à implantação na nuvem, o Azure Service Bus é ligeiramente superior ao RabbitMQ.

7.4.5 Possibilidades de implantação

O Azure Service Bus apresenta apenas uma possibilidade de implantação, a plataforma do Microsoft Azure.

O RabbitMQ possui um variado conjunto de opções de implantação, entre as mesmas:

- *Onpremise*;
- CloudAMQP;
- Microsoft Azure;
- Google Cloud Platform;
- AWS.

Portanto, pode-se inferir que a nível de possibilidades de implantação o RabbitMQ supera significativamente o Azure Service Bus.

7.5 Resumo dos resultados

Tendo em conta os resultados das secções anteriores, foi construída a tabela 7.1 que mostra para cada indicador qual a tecnologia superior.

Tabela 7.1: Azure Service Bus vs RabbitMQ

Indicador	**Tecnologia**
Tempo	Azure Service Bus
Tempo de trabalho	RabbitMQ
Monitorização	Azure Service Bus
Implantação na nuvem	Azure Service Bus
Possibilidades de implantação	RabbitMQ

Portanto, pode-se concluir que na maioria dos critérios a solução relativa ao Azure Service Bus é vantajosa comparativamente ao RabbitMQ.

Para além disso, foi também concluído que as arquiteturas orientadas a mensagens apresentam melhor desempenho comparativamente ao uso de base de dados partilhada com *polling*.

Capítulo 8

Conclusões

O presente capítulo tem como conteúdo as conclusões do projeto, incluindo os objetivos alcançados, limitações e trabalho futuro e apreciação final.

8.1 Objetivos alcançados

O objetivo deste projeto consistia em passar um sistema de software em que existe comunicação indireta entre os vários componentes através da base de dados para uma solução orientada a mensagens, ou seja, realizar uma reengenharia.

Assim sendo, seguidamente são apresentados os requisitos e respetivo grau de realização.

Tabela 8.1: Realização de requisitos

Requisito	**Resultado**
Levantamento de conceitos de integração de sistemas e comunicação entre componentes.	Realizado
Análise e comparação de tecnologias para comunicação entre componentes.	Realizado
Ser possível um componente enviar mensagens para a respetiva queue.	Realizado
Possibilitar que dado componente receba mensagens da queue pretendida.	Realizado
Verificar o número de mensagens que acarreta uma queue.	Realizado
Verificar o fluxo de mensagens numa queue.	Realizado maioritariamente
Possibilidade de alterar a tecnologia utilizada enquanto *message bus*.	Realizado
Comparação entre as escolhas tecnológicas.	Realizado
Comparação entre os sistemas inicial e final.	Realizado
Ambiente de integração e entrega contínuas.	Realizado parcialmente
Tratamento de exceções	Realizado
Pesquisa no âmbito da implantação de soluções distribuídas.	Realizado parcialmente
Várias possibilidades de implantação de *message bus*.	Realizado parcialmente

Tal como se pode observar na tabela 8.1, a maioria dos requisitos definidos na mesma foram realizados.

8.2 Limitações e trabalho futuro

Independentemente do empenho colocado no desenvolvimento deste projeto, nem tudo foi realizado com completo sucesso. Como tal, nesta secção são expostas as limitações aferidas ao longo do projeto, assim como trabalho futuro que poderia ser realizado.

Apesar de se ter chegado à conclusão de que as arquiteturas orientadas a mensagens apresentam melhor desempenho comparativamente às que se baseiam base de dados partilhada com *polling*, a diferença não é considerada significativa.

Posto isto, antes da alteração de um sistema de *software* que apresente uma arquitetura semelhante ao SmartDocumentor, esta decisão deve ser analisada antes de ser executada devido à complexidade que a transformação possa acarretar.

No entanto, no caso de sistemas que necessitam de monitorização do fluxo de informação ou de modificação completa do negócio, o rumo a tomar pode incluir passar para uma arquitetura orientada a mensagens.

Assim sendo, o trabalho futuro inclui adotar cada vez mais arquiteturas baseadas em mensagens na empresa de forma a evoluir os sistemas construídos pela mesma.

8.3 Apreciação final

O projeto foi uma boa forma de aprofundar conhecimentos no âmbito de arquiteturas orientadas a mensagens, algo que é cada vez mais utilizado no desenvolvimento de sistemas de *software*.

A documentação executada neste projeto fornece as bases para a implementação de *messaging* aquando da construção de aplicações.

Portanto, espera-se que a pesquisa e trabalho desenvolvidos sejam úteis nos casos semelhantes ao SmartDocumentor e também para novos sistemas que a empresa venha a desenvolver.

Referências

[1] *SmartDocumentor - DevScope OCR*. URL: http://www.smartdocumentor.net/ (acedido em 30/10/2019).

[2] G. Hohpe e S. Architect, «Enterprise Integration Patterns», rel. téc., 2002.

[3] STANDS4 LLC., *What does polling mean?* URL: https://www.definitions.net/definition/polling (acedido em 08/06/2020).

[4] P. Writer e E. Nakamura, *Choreography - Cloud Design Patterns | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/choreography (acedido em 28/11/2019).

[5] A. M. Hinze, *Principles and applications of distributed event-based systems*. IGI Global, 2010.

[6] Microsoft, *Choose a messaging model in Azure to loosely connect your services - Learn | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/learn/modules/choose-a-messaging-model-in-azure-to-connect-your-services/ (acedido em 07/11/2019).

[7] G. Hohpe, *Enterprise Integration Patterns - Shared Database*. URL: https://www.enterpriseintegrationpatterns.com/patterns/messaging/SharedDataBaseIntegration.html (acedido em 19/11/2019).

[8] C. Chen, *Command vs. Event in Domain Driven Design - Ingeniously Simple - Medium*, 2018. URL: https://medium.com/ingeniouslysimple/command-vs-event-in-domain-driven-design-be6c45be52a9 (acedido em 03/07/2020).

[9] Microsoft, *Choose whether to use messages or events - Learn | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/learn/modules/choose-a-messaging-model-in-azure-to-connect-your-services/2-choose-whether-to-use-message-queues-or-events (acedido em 14/11/2019).

[10] G. Hohpe, *Enterprise Integration Patterns - Competing Consumers*. URL: https://www.enterpriseintegrationpatterns.com/patterns/messaging/CompetingConsumers.html (acedido em 25/11/2019).

[11] Microsoft, *Padrão de Consumidores Concorrentes - Cloud Design Patterns | Microsoft Docs*. URL: https://docs.microsoft.com/pt-pt/azure/architecture/patterns/competing-consumers (acedido em 25/11/2019).

[12] C. Bennage, M. Wasson, N. Schonning, G. Chan, A. Buck e M. Wilson, *Queue-Based Load Leveling pattern - Cloud Design Patterns | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/queue-based-load-leveling (acedido em 10/12/2019).

[13] N. Schonning, A. Buck, M. Wasson e M. Wilson, *Publisher-Subscriber pattern | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber (acedido em 13/02/2020).

[14] C. Richardson, *Sagas*. URL: https://microservices.io/patterns/data/saga.html (acedido em 27/01/2020).

[15] O. Wolf, *Benefits of Microservices - Choreography over Orchestration, Low Coupling and High Cohesion*. URL: https://specify.io/concepts/microservices (acedido em 17/02/2020).

[16] G. Hohpe, *Enterprise Integration Patterns - Event-Driven Consumer*. URL: https://www.enterpriseintegrationpatterns.com/patterns/messaging/EventDrivenConsumer.html (acedido em 25/11/2019).

[17] C. Marzilli, *First Impressions with Platform Events and the Salesforce Enterprise Messaging Platform | Developer Force Blog*. URL: https://developer.salesforce.com/blogs/developer-relations/2017/05/first-impressions-platform-events-salesforce-enterprise-messaging-platform.html (acedido em 17/02/2020).

[18] L. Atchison, *Microservices: What They Are and How They Work*. URL: https://blog.newrelic.com/technology/microservices-what-they-are-why-to-use-them/ (acedido em 26/11/2019).

[19] T. Mauro, *Microservices at Netflix: Lessons for Architectural Design*. URL: https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/ (acedido em 26/11/2019).

[20] M. Richards, *Software architecture patterns*. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA ..., 2015, vol. 4.

[21] Microsoft, *What is cloud computing? - Learn | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/learn/modules/principles-cloud-computing/2-what-is-cloud-computing (acedido em 10/12/2019).

[22] E. Chezhiyan e U. H. Manivannan, *What are Business Processes? - Business Activity Monitoring*. URL: https://docs.serverless360.com/docs/bam-what-are-business-processes (acedido em 27/11/2019).

[23] Microsoft, *Azure Application Architecture Guide | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/architecture/guide/ (acedido em 25/11/2019).

[24] C. Fernando, *Comparison of Asynchronous Messaging Technologies: JMS, AMQP, and MQTT - DZone IoT*, 2016. URL: https://dzone.com/articles/comparison-of-asynchronous-messaging-technologies (acedido em 09/12/2019).

[25] R. Cohn, «A Comparison of AMQP and MQTT», rel. téc. URL: http://www.oasis-open.org/news/pr/amqp-tc.

[26] A. W. Services, *Amazon Simple Queue Service (SQS) | Message Queuing for Messaging Applications | AWS*, 2019. URL: https://aws.amazon.com/sqs/ (acedido em 04/12/2019).

[27] A. S. Foundation, *ActiveMQ*, 2019. URL: https://activemq.apache.org/ (acedido em 09/12/2019).

[28] R. Monson-Haefel, *5 Minutes or Less: ActiveMQ with JMS Queues and Topics - Tomitribe*, 2019. URL: https://www.tomitribe.com/blog/5-minutes-or-less-activemq-with-jms-queues-and-topics/ (acedido em 09/12/2019).

[29] A. S. Foundation, *Apache Kafka*, 2017. URL: https://kafka.apache.org/ (acedido em 05/12/2019).

[30] J. Vanlightly, *Event-Driven Architectures - Queue vs Log - A Case Study*. URL: https://jack-vanlightly.com/blog/2018/5/21/event-driven-architectures-queue-vs-log-case-study?fbclid=IwAR1ft74G8eQV9Ct4Qb1HN6WIiOimv6ZnQV1qVmX-ZUZCRyjvkufjqPDBxk4 (acedido em 17/06/2020).

[31] Microsoft, *Communicate between applications with Azure Queue storage - Learn | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/learn/modules/communicate-between-apps-with-azure-queue-storage/ (acedido em 09/12/2019).

[32] ——, *Choose a messaging platform - Learn | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/learn/modules/implement-message-workflows-with-service-bus/2-choose-a-messaging-platform (acedido em 13/01/2020).

[33] ——, *Choose Azure Event Grid - Learn | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/learn/modules/choose-a-messaging-model-in-azure-to-connect-your-services/4-choose-event-grid (acedido em 10/12/2019).

[34] ——, *Choose Azure Event Hubs - Learn | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/learn/modules/choose-a-messaging-model-in-azure-to-connect-your-services/5-choose-azure-event-hubs (acedido em 10/12/2019).

[35] G. Brown, *Google Cloud Platform and Applications | Berlioz*, 2019. URL: https://berlioz.cloud/blog/google-cloud-platform-and-applications/ (acedido em 03/12/2019).

[36] I. Pivotal Software, *Messaging that just works — RabbitMQ*. URL: https://www.rabbitmq.com/%7B%5C#%7Dgetstarted (acedido em 09/12/2019).

[37] S. Sanfilippo, *Redis*. URL: https://redis.io/ (acedido em 02/12/2019).

[38] T. Macedo e F. Oliveira, *Redis Cookbook: Practical Techniques for Fast Data Manipulation*, 2011.

[39] M. Wenzel, J. Parente, S. Killeen, P. Kulikov e M. Veloso, *Asynchronous message-based communication | Microsoft Docs*, 2018. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/asynchronous-message-based-communication (acedido em 27/11/2019).

[40] Oracle, *What Is a Socket?*, 2019. URL: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html (acedido em 09/12/2019).

[41] M. Wenzel, J. Parente, S. Killeen, P. Kulikov e M. Veloso, *Asynchronous message-based communication | Microsoft Docs*, 2018. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/asynchronous-message-based-communication (acedido em 27/11/2019).

[42] Google, *Google Cloud Computing, Hosting Services & APIs | Google Cloud*, 2019. URL: https://cloud.google.com/gcp/?utm%7B%5C_%7Dsource=google%7B%5C&%7Dutm%7B%5C_%7Dmedium=cpc%7B%5C&%7Dutm%7B%5C_%7Dcampaign=emea-pt-all-en-dr-bkws-all-all-trial-e-gcp-1007176%7B%5C&%7Dutm%7B%5C_%7Dcontent=text-ad-none-any-DEV%7B%5C_%7Dc-CRE%7B%5C_%7D219658510788-ADGP%7B%5C_%7DHybrid+%7B%5C%%7D7C+AW+SEM+%7B%5C%%7D7C+BKWS+%7B~%7D+EXA%7B%5C_%7D1:1%7B%5C_%7DPT%7B%5C_%7DEN%7B%5C_%7DGeneral%7B%5C_%7DCloud%7B%5C_%7DTOP%7B%5C_%7Dcloud+goog (acedido em 03/12/2019).

[43] Microsoft, *Microsoft Azure Cloud Computing Platform & Services*, 2019. URL: https://azure.microsoft.com/en-us/ (acedido em 03/12/2019).

[44] A. W. Services, *Amazon Web Services (AWS) - Cloud Computing Services*, 2019. URL: https://aws.amazon.com/ (acedido em 03/12/2019).

[45] IBM, *IBM Cloud | IBM*. URL: https://www.ibm.com/cloud (acedido em 13/01/2020).

[46] AlibabaCloud, *Alibaba Cloud: Reliable & Secure Cloud Solutions to Empower Your Global Business*. URL: https://www.alibabacloud.com/ (acedido em 13/01/2020).

[47] T. L. Foundation, *What is Kubernetes - Kubernetes*. URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (acedido em 06/02/2020).

[48] Techopedia Inc., *What is a Central Processing Unit (CPU)? - Definition from Techopedia*. URL: https://www.techopedia.com/definition/2851/central-processing-unit-cpu (acedido em 06/02/2020).

[49] Docker, *What is a Container? | App Containerization | Docker*. URL: https://www.docker.com/resources/what-container (acedido em 06/02/2020).

[50] J. Hollan, *Announcing KEDA: bringing event-driven containers and functions to Kubernetes - Open Source Blog*, 2019. URL: https://cloudblogs.microsoft.com/opensource/2019/05/06/announcing-keda-kubernetes-event-driven-autoscaling-containers/ (acedido em 05/02/2020).

[51] D. P. LEADS, *Announcing Distributed Application Runtime (Dapr), an open source project to make it easier for every developer to build microservice applications - Open Source Blog*, 2019. URL: https://cloudblogs.microsoft.com/opensource/2019/10/16/announcing-dapr-open-source-project-build-microservice-applications/ (acedido em 05/02/2020).

[52] A. Mock, *GitHub - AndreasM009/dapr-keda-azsbqueue: Dapr Azure Servucebus Queue binding and scaling with keda*, 2020. URL: https://github.com/AndreasM009/dapr-keda-azsbqueue (acedido em 06/02/2020).

[53] N. Rich e M. Holweg, «Value analysis», *Value engineering: Innoregio: dissemination of innovation and knowledge management techniques, report produced for the EC funded project. United Kingdom: Lean Enterprise Research Centre Cardiff*, 2000.

[54] S. Nicola, *Análise Valor Aula 2 22OUT 2019 1hora AHP*, 2019.

[55] ——, *Análise de Valor de negócio*, 2019.

[56] P. A. Koen, G. M. Ajamian, S. Boyce, A. Clamen, E. Fisher, S. Fountoulakis, A. Johnson, P. Puri e R. Seibert, «Fuzzy front end: effective methods, tools, and techniques», *The PDMA toolbook 1 for new product development*, 2002.

[57] P. Koen, G. Ajamian, R. Burkart, A. Clamen, J. Davidson, R. D'Amore, C. Elkins, K. Herald, M. Incorvia, A. Johnson et al., «Providing clarity and a common language to the "fuzzy front end"», *Research-Technology Management*, vol. 44, n.º 2, pp. 46–55, 2001.

[58] N. Fain e B. Wagner, «Management of the fuzzy front end of innovation, by Oliver Gassman and Fiona Schweitzer», *International Journal of Market Research*, vol. 58, p. 637, 2016. doi: 10.2501/IJMR-2016-038.

[59] P. Belliveau, A. Griffin e S. Somermeyer, *The PDMA toolbook 1 for new product development*. John Wiley & Sons, 2004.

[60] M. Welsh, D. Culler e E. Brewer, «SEDA: an architecture for well-conditioned, scalable internet services», *ACM SIGOPS Operating Systems Review*, vol. 35, n.º 5, pp. 230–243, 2001.

[61] D. Walters e G. Lancaster, «Implementing value strategy through the value chain», *Management Decision*, 2000.

[62] T. Woodall, «Conceptualising 'value for the customer': an attributional, structural and dispositional analysis», *Academy of marketing science review*, vol. 12, n.º 1, pp. 1–42, 2003.

[63] W. Ulaga e A. Eggert, «Value-based differentiation in business relationships: Gaining and sustaining key supplier status», *Journal of marketing*, vol. 70, n.º 1, pp. 119–136, 2006.

[64] A. Osterwalder e Y. Pigneur, «Modeling value propositions in e-Business», em *Proceedings of the 5th international conference on Electronic commerce*, 2003, pp. 429–436.

[65] ——, *Business model generation: a handbook for visionaries, game changers, and challengers*. John Wiley & Sons, 2010.

[66] J. Buchanan e L. Gardiner, «A comparison of two reference point methods in multiple objective mathematical programming», *European Journal of Operational Research*, vol. 149, n.º 1, pp. 17–34, 2003.

[67] T. L. Saaty, «What is the analytic hierarchy process?», em *Mathematical models for decision support*, Springer, 1988, pp. 109–121.

[68] S. Brown, *The C4 model for visualising software architecture*. URL: `https://c4model.com/` (acedido em 29/04/2020).

[69] P. B. Kruchten, «The 4+ 1 view model of architecture», *IEEE software*, vol. 12, n.º 6, pp. 42–50, 1995.

[70] C. Patterson, *MassTransit*. URL: `https://masstransit-project.com/MassTransit/` (acedido em 04/12/2019).

[71] N. Ltd., *NServiceBus • Particular Software*. URL: `https://particular.net/nservicebus` (acedido em 10/12/2019).

[72] EasyNetQ, *EasyNetQ - An easy .NET API for RabbitMQ*. URL: `http://easynetq.com/` (acedido em 10/12/2019).

[73] Microsoft, *Portal do Microsoft Azure | Microsoft Azure*. URL: `https://azure.microsoft.com/pt-pt/features/azure-portal/` (acedido em 19/05/2020).

[74] CloudAMQP, *CloudAMQP - RabbitMQ as a Service*. URL: `https://www.cloudamqp.com/` (acedido em 19/05/2020).

[75] Y. Pliner, *EasyNetQ - An easy .NET API for RabbitMQ*. URL: `https://easynetq.com/` (acedido em 19/05/2020).

[76] ——, *Send Receive · EasyNetQ/EasyNetQ Wiki · GitHub*. URL: `https://github.com/EasyNetQ/EasyNetQ/wiki/Send-Receive` (acedido em 19/05/2020).

[77] M. Shahin, M. A. Babar e L. Zhu, «Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices», *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[78] Microsoft, *Azure DevOps Services | Microsoft Azure*. URL: `https://azure.microsoft.com/en-us/services/devops/?nav=min` (acedido em 18/05/2020).

[79] Tech Target, *What is Software as a Service (SaaS)? - Definition from WhatIs.com*. URL: `https://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service` (acedido em 19/05/2020).

[80] Microsoft, *Azure Repos – Git Repositories | Microsoft Azure*. URL: `https://azure.microsoft.com/en-us/services/devops/repos/?nav=min` (acedido em 19/05/2020).

[81] ——, *Azure Pipelines | Microsoft Azure*. URL: `https://azure.microsoft.com/en-us/services/devops/pipelines/?nav=min` (acedido em 19/05/2020).

[82] SubMain Software, *SubMain / GhostDoc - Getting Started*. URL: `https://submain.com/ghostdoc/GettingStarted/` (acedido em 12/05/2020).

[83] Doxygen, *Doxygen: Main Page*. URL: `http://www.doxygen.nl/` (acedido em 12/05/2020).

[84] A. Aghazadah, *(9) Doxygen Basics - YouTube*. URL: `https://www.youtube.com/watch?v=TtRn3HsOm1s` (acedido em 12/05/2020).

[85] G. Warren, W. A. Rohm, M. Jones, M. Jacobs e G. Lu, *Install FxCop analyzers - Visual Studio | Microsoft Docs*. URL: `https://docs.microsoft.com/en-`

us/visualstudio/code-quality/install-fxcop-analyzers?view=vs-2019%7B%5C#%7Dnuget-package (acedido em 11/05/2020).

[86] S. Harwell, *GitHub - DotNetAnalyzers/StyleCopAnalyzers: An implementation of StyleCop rules using the .NET Compiler Platform*. URL: https://github.com/DotNetAnalyzers/StyleCopAnalyzers (acedido em 11/05/2020).

[87] I. Sommerville, «Software engineering 9th Edition», *ISBN-10*, vol. 137035152, 2011.

[88] ——, «Making ethnography accessible: Bringing real-world experience to HCI designers and software engineers», em *26th International Conference of Software Engineering. Edinburgh*, 2004, pp. 36–38.

[89] M. Olan, «Unit testing: Test early, test often», *Journal of Computing Sciences in Colleges - JCSC*, vol. 19, jan. de 2003.

[90] STF, *Integration Testing - Software Testing Fundamentals*. URL: http://softwaretestingfundamentalcom/integration-testing/ (acedido em 19/05/2020).

[91] ——, *System Testing - Software Testing Fundamentals*. URL: http://softwaretestingfundamentals.com/system-testing/ (acedido em 19/05/2020).

[92] J. Cooper e M. Fisher, «Software acquisition capability maturity model (SA-CMM)», Version 1.03, tech. report CMU/SEI-2002-TR-010, Software Eng. Inst., Pittsburgh, rel. téc., 2002.

[93] D. Odhiambo, *Performance Optimization in Software Development - The Andela Way - Medium*, 2018. URL: https://medium.com/the-andela-way/performance-optimization-in-software-development-ae7952ab885e (acedido em 06/02/2020).

[94] J. Heyman, C. Byström, J. Hamrén e H. Heyman, *Locust - A modern load testing framework*. URL: https://locust.io/ (acedido em 16/01/2020).

[95] T. N. A. Team, *GitHub - newsapps/beeswithmachineguns: A utility for arming (creating) many bees (micro EC2 instances) to attack (load test) targets (web applications)*. URL: https://github.com/newsapps/beeswithmachineguns (acedido em 16/01/2020).

[96] C. Goldberg, *GitHub - cgoldberg/multi-mechanize: Performance Test Framework in Python*. URL: https://github.com/cgoldberg/multi-mechanize (acedido em 16/01/2020).

[97] J. D. Software, *Siege Home*. URL: https://www.joedog.org/siege-home/ (acedido em 16/01/2020).

[98] A. S. Foundation, *ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4*. URL: https://httpd.apache.org/docs/2.4/programs/ab.html (acedido em 16/01/2020).

[99] Httperf, *GitHub - httperf/httperf: The httperf HTTP load generator*. URL: https://github.com/httperf/httperf (acedido em 16/01/2020).

[100] A. S. Foundation, *Apache JMeter - Apache JMeter™*. URL: https://jmeter.apache.org/ (acedido em 16/01/2020).

[101] Google, *PageSpeed Insights*. URL: https://developers.google.com/speed/pagespeed/insights/ (acedido em 16/01/2020).

[102] Sitespeed.io, *Welcome to the wonderful world of Web Performance*. URL: https://www.sitespeed.io/ (acedido em 16/01/2020).

[103] Google, *PageSpeed Module | Google Developers*. URL: https://developers.google.com/speed/pagespeed/module (acedido em 16/01/2020).

[104] *WebPageTest - Website Performance and Optimization Test*. URL: https://www.webpagetest.org/ (acedido em 16/01/2020).

[105] RStudio PBC., *RStudio | Open source & professional software for data science teams - RStudio*. URL: https://rstudio.com/ (acedido em 29/05/2020).

[106] The R Foundation, *R: The R Project for Statistical Computing*. URL: https://www.r-project.org/ (acedido em 29/05/2020).

[107] A. Leoneti, *Testes paramétricos vs. não paramétricos - ESTRUTURA DOS TESTES DE HIPÓTESES | Coursera*. URL: https://www.coursera.org/lecture/estatistica-nao-parametrica/testes-parametricos-vs-nao-parametricos-5hPqa (acedido em 07/02/2020).

Apêndice A

Estado da arte

A.1 Implantação de soluções distribuídas

A.1.1 Evolução de sistemas onpremises para sistemas na nuvem e/ou híbridos

De entre as diversas diferenças fundamentais do uso de cenários *onpremise*, na nuvem e híbridos destacam-se os aspetos: custo, manutenção, confiabilidade, escalabilidade, controlo e segurança, personalização e implementação, cujas especificações se encontram comparadas na tabela A.1 [21].

Tabela A.1: Onpremise vs Nuvem vs Híbrido

Fator	Onpremise	Nuvem	Híbrido
Custo	Investimento inicial é normalmente elevado, não apenas em termos de desenvolvimento, mas também derivado de custos de informática e hardware associados.	Investimento inicial baixo sem a necessidade de adicionar hardware. Além disso, torna-se mais simples de controlar os custos ao longo do tempo.	A habilidade de usar serviços baseados na nuvem quando necessário significa que as organizações pagam mais apenas quando é precisa computação extra.
Manutenção	A equipa de informática é responsável por assegurar a disponibilidade dos serviços no caso de operações de recuperação de desastre.	O vendedor é responsável pela segurança e sólido plano de backup para recuperação de dados.	Existem muitos fatores que dependem da forma como a empresa organiza e manipula a nuvem. Em qualquer caso, a organização mantém a opção de salvaguardar os seus processos e onde os dados se localizam.

Continuação na próxima página

Tabela A.1 – *Onpremise vs Nuvem vs Híbrido*

Fator	Onpremise	Nuvem	Híbrido
Confiabilidade	Desde que a organização tenha controlo total do departamento de informática, consegue monitorizar constantemente as situações e deduzir quando e porquê os serviços se encontram indisponíveis.	Os serviços podem ser acedidos de qualquer local no mundo e a qualquer momento através do fornecimento de dados de segurança. No entanto, dados valiosos têm de ser confiados ao vendedor.	No caso de haver algum problema no cenário onpremise, as operações podem ser efetuadas na nuvem. No caso de a nuvem se encontrar indisponível por algum motivo, pode-se usar as operações onpremise.
Escalabilidade	O planeamento a longo prazo e compromisso de recursos são requeridos para expandir as operações do cenário de infraestrutura onpremise.	É necessário pouco esforço e tempo para escalar ou efetuar a ação contrária no contexto dos requisitos. O desperdício de recursos é também minimizado.	A escalabilidade varia segundo as operações onpremise particulares da organização e a forma como são geridas na nuvem.
Controlo e segurança	A empresa consegue estimar os riscos e realizar os passos lógicos para os reduzir.	O vendedor da plataforma possui controlo total.	A organização tem a liberdade de criar soluções de segurança para dados sensíveis e operações críticas e depende do vendedor para as restantes.
Personalização	A personalização é possível, no entanto podem ser adicionados custos extra quando o *software* é atualizado pelo vendedor.	Grande margem para personalização e últimas inovações e novidades encontram-se disponíveis em forma de atualização.	A personalização é simples, fornecendo maior flexibilidade à infraestrutura onpremise.

Continuação na próxima página

Tabela A.1 – *Onpremise vs Nuvem vs Híbrido*

Fator	Onpremise	Nuvem	Híbrido
Implementação	O processo de implementação pode ser completo imediatamente ou nas instalações por requisito da organização.	Pode ser rapidamente implementada e os funcionários não precisam de programas longos de treino para trabalhar com eficazmente.	A implementação é rápida dado que a infraestrutura onpremise pode ser utilizada enquanto rampa de lançamento para operações baseadas na nuvem, muitas das quais já foram testadas e fornecidas pelo vendedor.

Apêndice B

Método AHP

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

Figura B.1: Valores de IR para matrizes quadradas de ordem n

B.1 Tecnologia

B.1.1 Fase 3

Tabela B.1: Matriz normalizada dos critérios de tecnologia do segundo nível

	Monitorização	**Escalabilidade**	**Facilidade de implantação na nuvem**	**Possibilidades de implantação**
Monitorização	1/14	2/23	1/17	3/53
Escalabilidade	3/8	12/23	8/17	30/53
Facilidade de implantação na nuvem	1/7	3/23	2/17	5/53
Possibilidades de implantação	5/14	6/23	6/17	15/53

B.1.2 Fase 4

O vetor de prioridades (x) corresponde à coluna referente às prioridades relativas da tabela 5.2 de prioridade relativa de cada critério, ou seja:

$$x = (0,0684; 0,4833; 0,1213; 0,3135)$$

Para calcular o valor próprio (λ_{max}) considera-se que:

$$Ax = \lambda_{max} x$$

A corresponde à matriz de comparação dos critérios do segundo nível representada na tabela 5.1, logo:

$$\begin{bmatrix} 1 & 1/6 & 1/2 & 1/5 \\ 6 & 1 & 4 & 2 \\ 2 & 1/4 & 1 & 1/3 \\ 5 & 1/2 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0.07 \\ 0.48 \\ 0.12 \\ 0.31 \end{bmatrix} \cong \lambda_{\max} \begin{bmatrix} 0.07 \\ 0.48 \\ 0.12 \\ 0.31 \end{bmatrix} \Leftrightarrow$$

$$\Leftrightarrow \begin{bmatrix} 0.27 \\ 2.00 \\ 0.48 \\ 1.26 \end{bmatrix} \cong \lambda_{\max} \begin{bmatrix} 0.07 \\ 0.48 \\ 0.12 \\ 0.31 \end{bmatrix}$$

$$\lambda_{\max} = average\left\{ \frac{0.27}{0.07}, \frac{2.00}{0.48}, \frac{0.48}{0.12}, \frac{0.1.26}{0.31} \right\} \simeq 4.02$$

Tendo em conta que n corresponde ao número de critérios, o índice de consistência (IC) é:

$$IC = \frac{\lambda_{\max} - n}{n - 1} = \frac{4.02 - 4}{4 - 1} \simeq 0.007$$

De acordo com a tabela da figura B.1, o valor de IR para a situação presente, isto é, o uso de 4 critérios é 0.90, considerando a fórmula da Razão de Consistência (RC), então:

$$RC = \frac{IC}{IR} = \frac{0.007}{0.90} \simeq 0.01 < 0.1$$

B.1.3 Fase 5

Tabela B.2: Matriz normalizada de monitorização do segundo nível

	Azure Service Bus	**RabbitMQ**	**Apache Kafka**	**Redis**
Azure Service Bus	12/21	6/11	3/7	12/19
RabbitMQ	1/7	3/22	3/14	2/19
Apache Kafka	2/21	1/22	1/14	1/19
Redis	4/21	3/11	2/7	4/19

Tabela B.3: Matriz normalizada de escalabilidade do segundo nível

	Azure Service Bus	**RabbitMQ**	**Apache Kafka**	**Redis**
Azure Service Bus	1/9	4/43	5/39	1/6
RabbitMQ	1/3	12/43	10/39	1/3
Apache Kafka	4/9	24/43	20/39	5/12
Redis	1/9	3/43	4/39	1/12

Tabela B.4: Matriz normalizada de facilidade de implantação na nuvem do segundo nível

	Azure Service Bus	**RabbitMQ**	**Apache Kafka**	**Redis**
Azure Service Bus	15/28	9/16	1/2	9/16
RabbitMQ	5/28	3/16	3/10	3/16
Apache Kafka	3/28	1/16	1/10	1/16
Redis	5/28	3/16	3/10	3/16

Tabela B.5: Matriz normalizada de possibilidades de implantação do segundo nível

	Azure Service Bus	**RabbitMQ**	**Apache Kafka**	**Redis**
Azure Service Bus	1/19	28/371	1/41	1/21
RabbitMQ	7/19	28/53	20/41	12/21
Apache Kafka	5/19	7/53	5/41	2/21
Redis	6/19	14/53	15/41	6/21

Apêndice C

Desenho arquitetural

C.1 Desenho arquitetural da solução do SmartDocumentor

C.1.1 Vista lógica

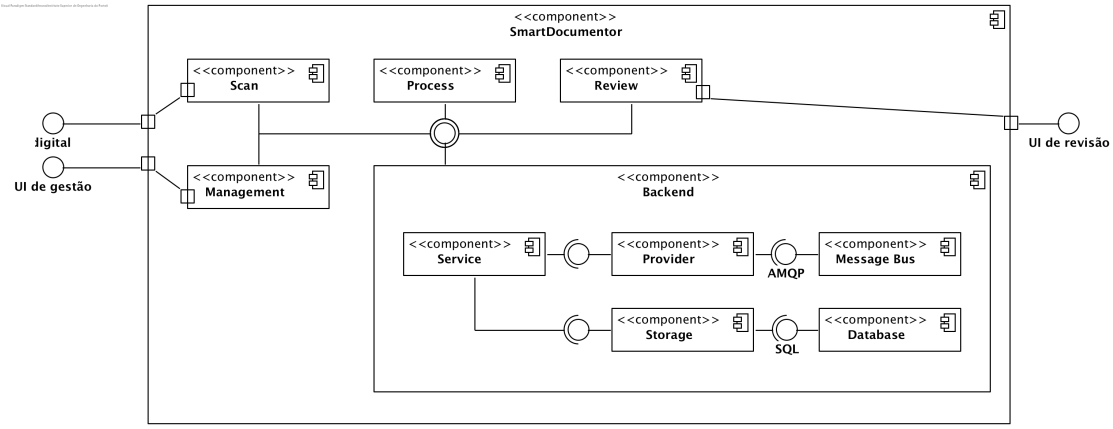

Figura C.1: Vista lógica

C.1.2 Vista de processos

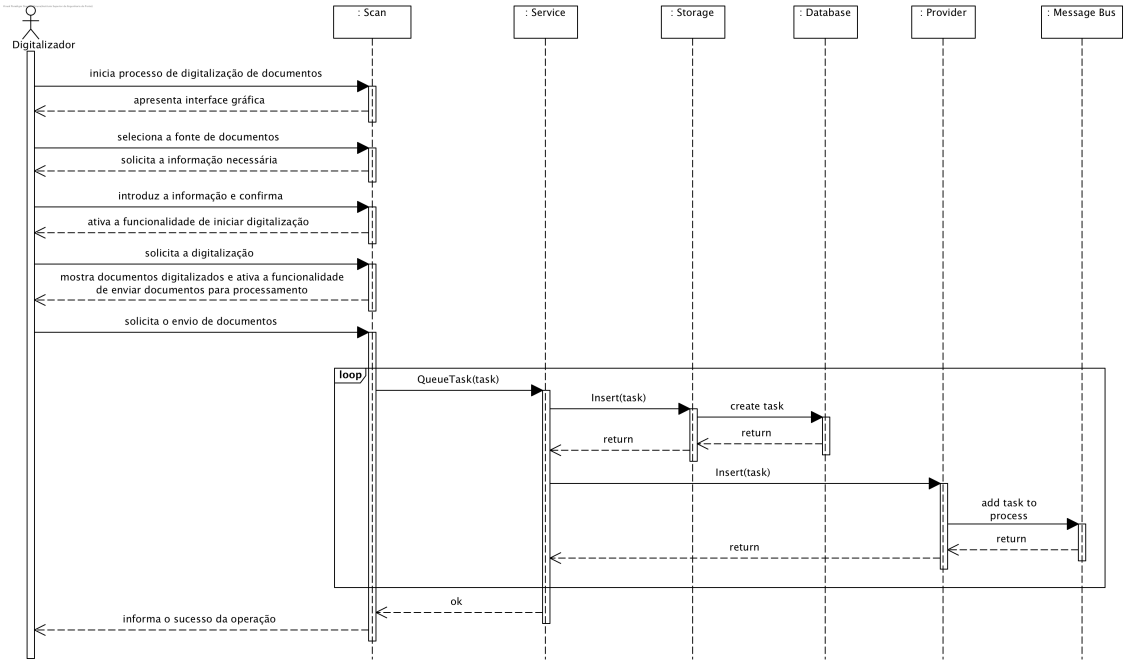

Figura C.2: Vista de processos - Digitalizar documentos

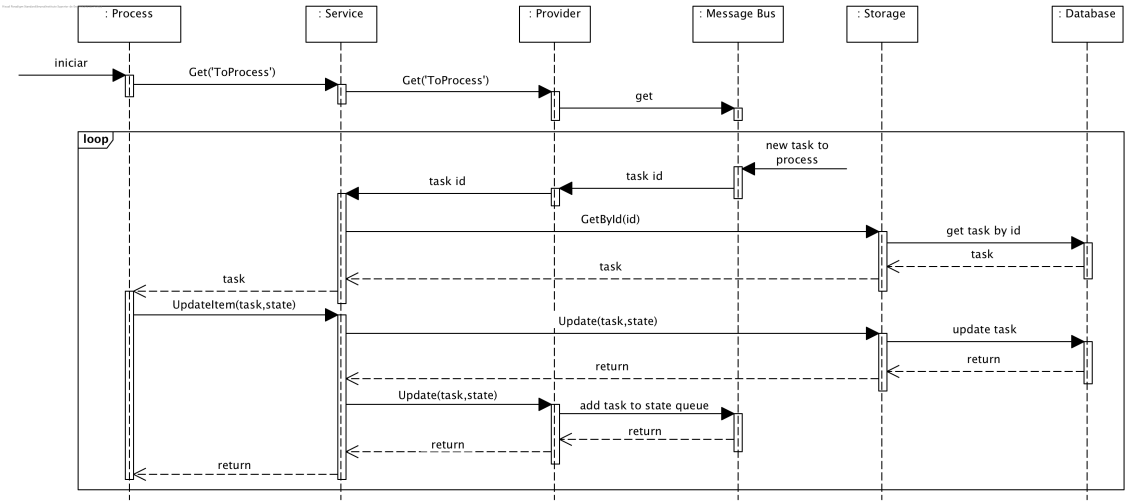

Figura C.3: Vista de processos - Processar documentos

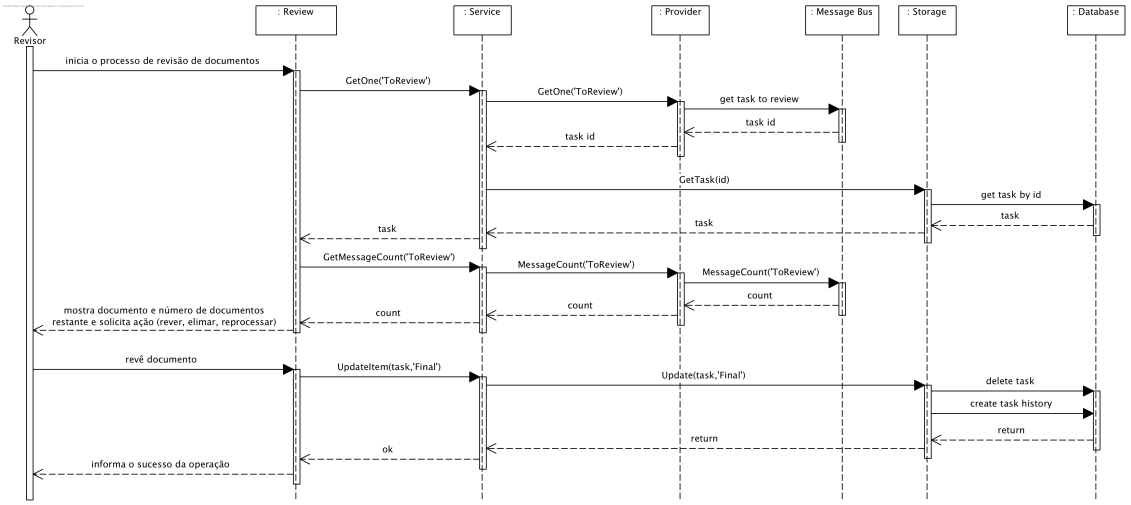

Figura C.4: Vista de processos - Rever documentos

C.2 Desenho arquitetural da solução generalizada

C.2.1 Vista lógica

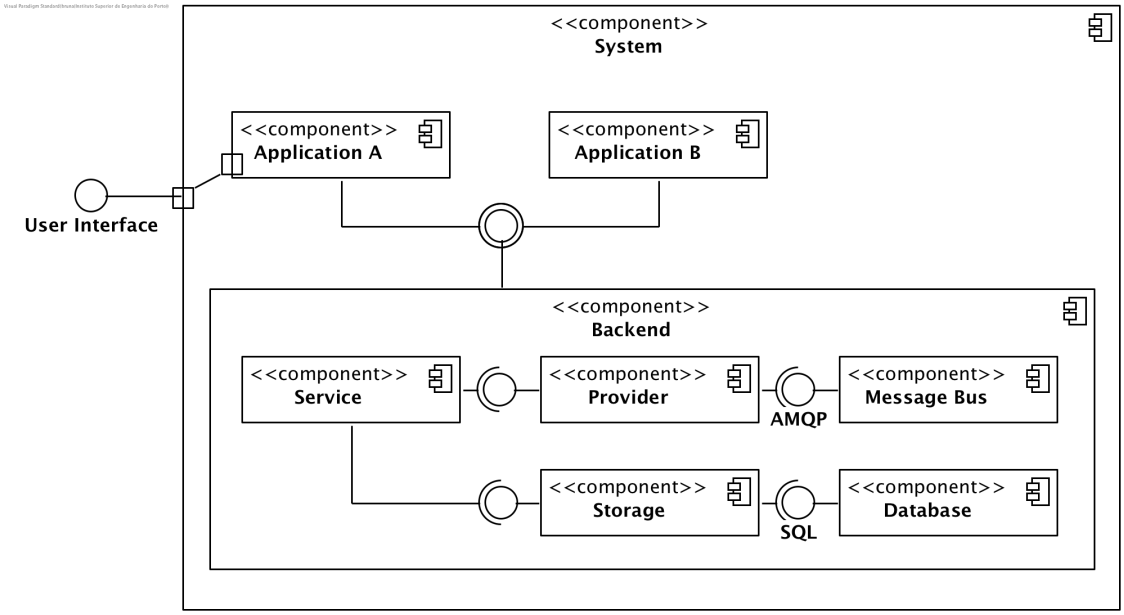

Figura C.5: Vista lógica

C.2.2 Vista de processos

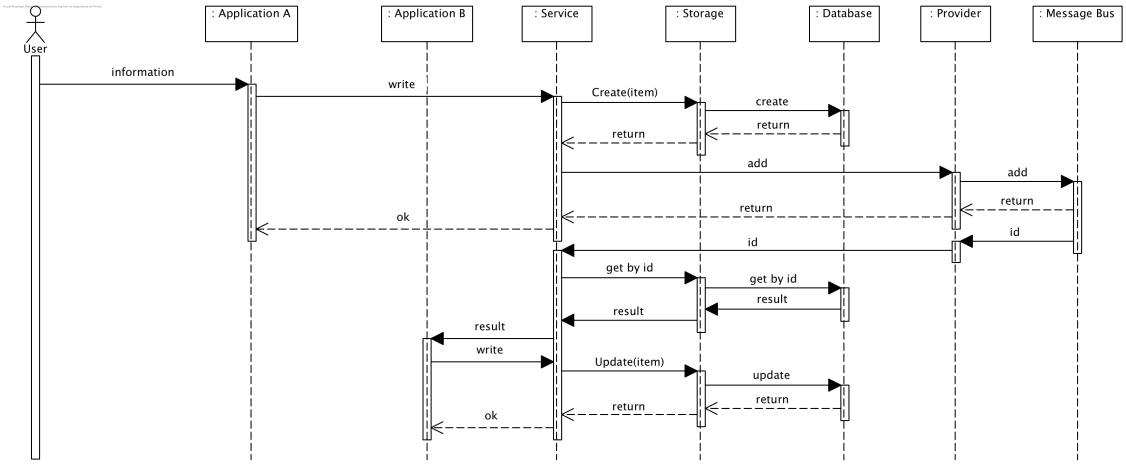

Figura C.6: Vista de processos

Apêndice D

Contrução da Solução

D.1 Prova de conceito

Figura D.1: Scan Station - Selecionar *workspace*

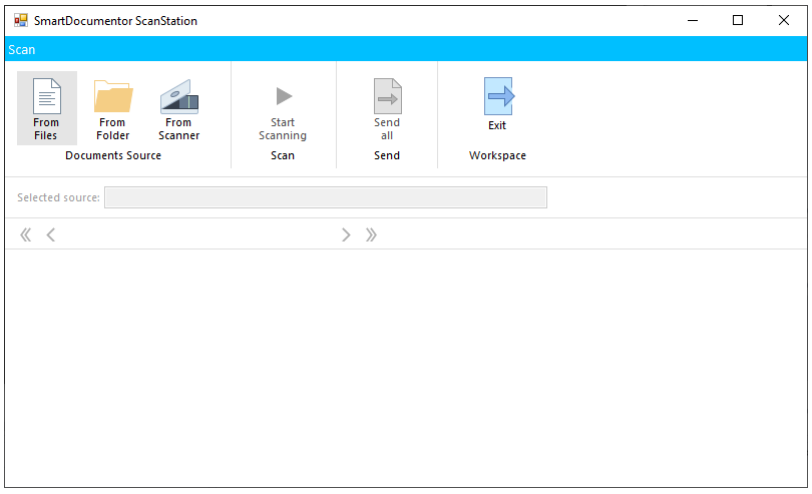

Figura D.2: Scan Station - Selecionar fonte

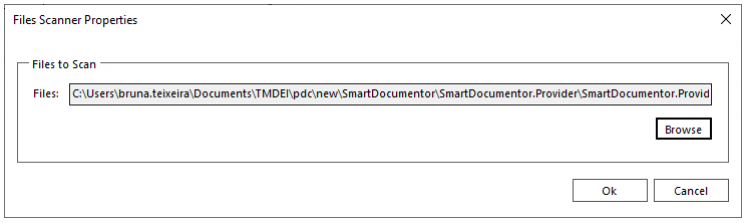

Figura D.3: Scan Station - Selecionar ficheiros

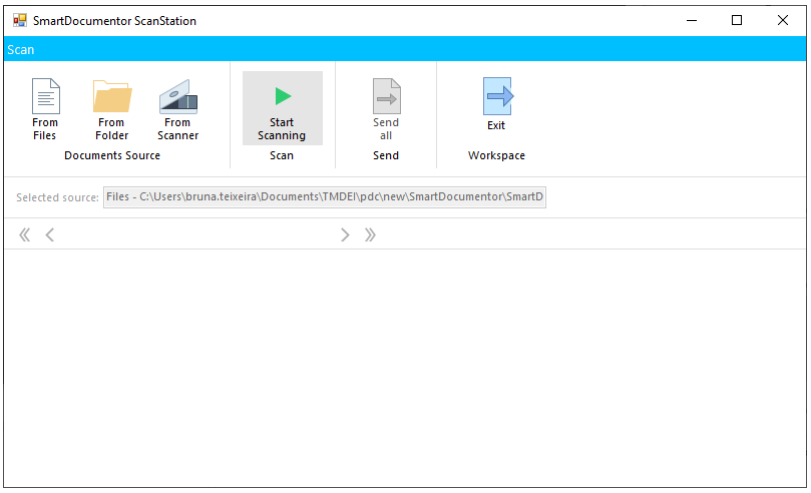

Figura D.4: Scan Station - Digitalizar documentos

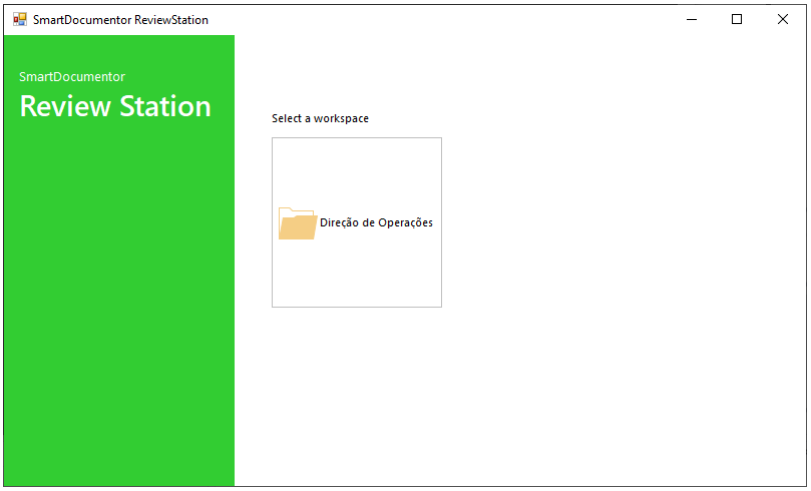

Figura D.5: Review Station - Selecionar *workspace*

D.2 Análise de código

```
"$schema": "https://raw.githubusercontent.com/DotNetAnalyzers/StyleCopAnalyzers/
"settings": {
  "documentationRules": {
    "companyName": "PlaceholderCompany",
    "xmlHeader": false,
    "documentExposedElements": true,
    "documentInternalElements": false,
    "documentPrivateElements": false,
    "documentInterfaces": true,
    "documentPrivateFields": false,
    "documentationCulture": "en-US"
  },
  "maintainabilityRules": {
    "topLevelTypes": [ "class", "interface", "struct", "enum" ]
  },
  "layoutRules": {
    "newlineAtEndOfFile": "require",
    "allowConsecutiveUsings": true
  },
  "indentation": {
    "indentationSize": 4,
    "useTabs": false
  },
  "namingRules": {},
  "orderingRules": {
    "systemUsingDirectivesFirst": false,
    "usingDirectivesPlacement": "outsideNamespace",
    "elementOrder": [
      "kind",
      "constant",
      "accessibility",
      "static",
      "readonly"
    ]
  }
}
}
```

Figura D.6: `stylecop.json` com regras de análise

Da instalação das duas ferramentas resultaram as referências da figura D.7.

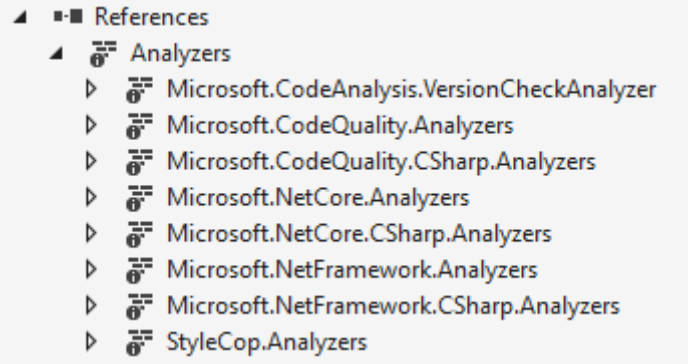

Figura D.7: Referências das ferramentas de análise

Apêndice E

Experiências e Avaliação

E.1 Tempo

E.1.1 Resultados referentes ao JMeter

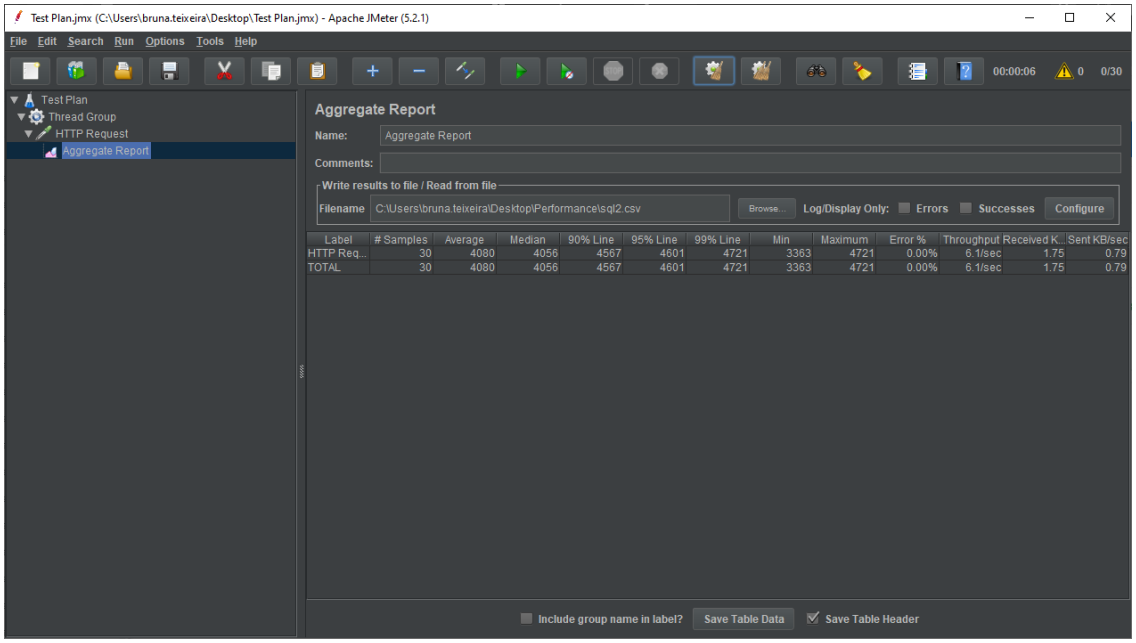

Figura E.1: JMeter - *Shared database*

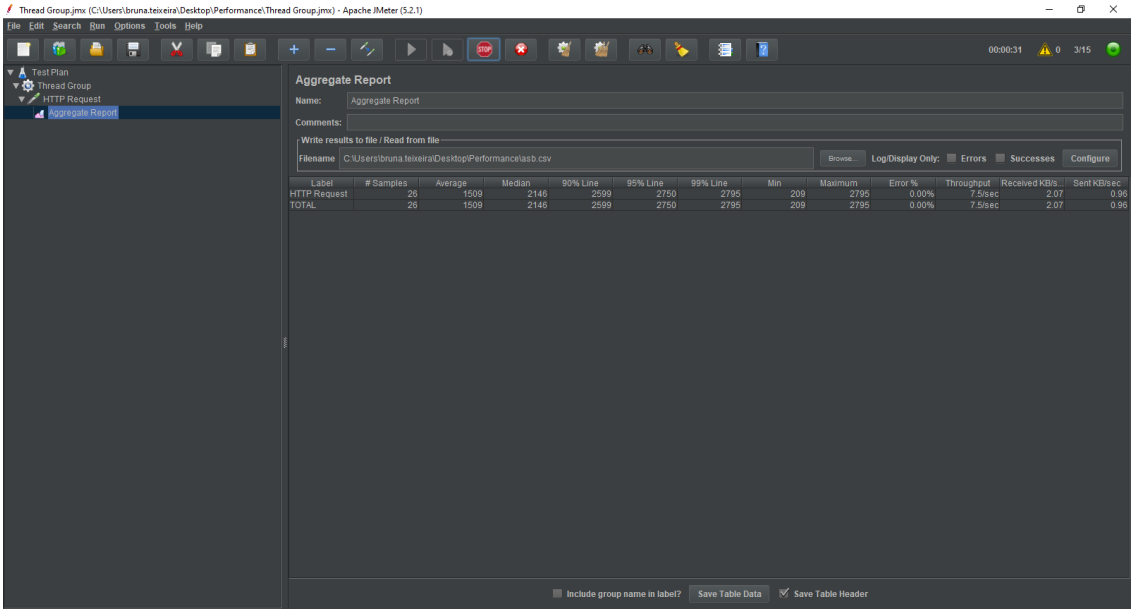

Figura E.2: JMeter - Azure Service Bus

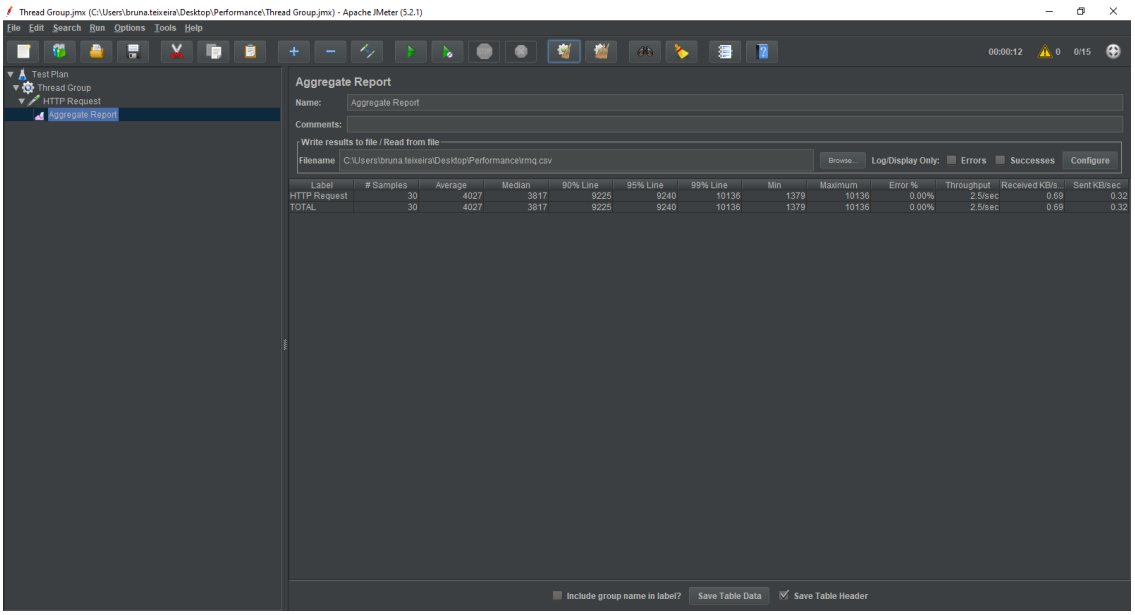

Figura E.3: JMeter - RabbitMQ