INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Desenvolvimento Rápido de Aplicações - Comparação de soluções em Outsystems e Mendix

INÊS MARGARIDA MONTEIRO DE MOURA PINTO
Outubro de 2021

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Desenvolvimento Rápido de Aplicações - Comparação de soluções em Outsystems e Mendix

Inês Margarida Monteiro de Moura Pinto

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

Orientador: Professor Paulo Maio

Porto, Outubro 2021

Resumo

A abordagem de desenvolvimento rápido de soluções (RAD) surgiu como resposta às limitações de modelos de desenvolvimento tradicionais. Esta define pessoas, competências, ferramentas e metodologias como os elementos essenciais do processo de desenvolvimento e considera a implementação de aplicações num período de 60 a 120 dias. As ferramentas ocupam um papel determinante nesta abordagem e os avanços tecnológicos têm permitido o seu desenvolvimento com elevados níveis de abstração, como as plataformas *low-code*.

O contexto de mudança contínua proporcionada pelo aparecimento de novas tecnologias pressiona as organizações de TI a minimizarem o *time-to-market*, sem comprometer a qualidade das soluções. Assim, surge a oportunidade da adoção de abordagens como desenvolvimento rápido de aplicações através de plataformas *low-code*. Contudo, ao nível de uma organização, a decisão de alterar métodos e procedimentos bem estabelecidos suscita algumas questões.

Neste sentido, este projeto de dissertação descreve os principais modelos e metodologias de desenvolvimento de software e as principais ferramentas *low-code*. Para além disso, avalia a aplicabilidade destas ferramentas, tendo em conta a adoção de diferentes arquiteturas de sistema, o impacto em termos de custo, qualidade e tempo em relação a ferramentas de desenvolvimento tradicionais e fatores organizacionais como metodologia de desenvolvimento e perfil de competências. Para esta avaliação, este projeto apresenta dois casos de estudo do mesmo domínio, sendo estes considerados para a construção de soluções nas ferramentas mais promissoras – Outsystems e Mendix. Desta avaliação, resulta uma comparação das duas plataformas que, por sua vez, permite analisar os aspetos considerados para a avaliação da aplicabilidade das ferramentas.

Palavras-chave: desenvolvimento rápido de aplicações, RAD, plataformas *low-code*, Outsystems, Mendix, metodologias de desenvolvimento de software

Abstract

The Rapid Application Development (RAD) approach has emerged as a response to the limitations of traditional development models. It defines people, skills, tools, and methodologies as the essential elements of the development process and considers the implementation of applications within 60 to 120 days. Tools play a key role in this approach, and technological advances have enabled their development with high levels of abstraction, such as low-code platforms.

The context of continuous change provided by the emergence of new technologies puts pressure on IT organizations to minimize time-to-market, without compromising the quality of solutions. Thus, the opportunity arises for the adoption of approaches such as rapid application development through low-code platforms. However, at the level of an organization, the decision to change well-established methods and procedures raises some questions.

In this sense, this dissertation project describes the main models and methodologies of software development and the main low-code platforms. Furthermore, it evaluates the applicability of these tools, considering the adoption of different software architectures, the impact in terms of cost, quality and time in relation to traditional development tools, and organizational factors such as development methodology and skill profile. For this evaluation, this project presents two case studies from the same domain, which are considered for building solutions on the most promising platforms - Outsystems and Mendix. This evaluation results in a comparison of the two platforms which, in turn, allows for an analysis of the aspects considered for the evaluation of the tools' applicability.

Keywords: rapid application development, RAD, low-code platforms, Outsystems, Mendix, software development methodologies

Agradecimentos

Agradeço ao Professor Paulo Maio pela sugestão deste projeto e pelo acompanhamento e orientação prestados ao longo da sua realização, destacando a atenção e disponibilidade para rever o trabalho, apresentando, em todos os momentos, críticas construtivas que permitiram melhorar a qualidade do trabalho realizado.

Agradeço ao ISEP e, em particular ao DEI, por todos as ferramentas que me deu e por todas as portas que me abriu.

Agradeço à minha família e, em particular, aos meus pais e à minha irmã por me apoiarem, em todos os momentos, e por me incentivarem a não desistir e continuar a trabalhar para alcançar os meus objetivos.

Agradeço aos meus amigos e aos meus colegas por todo o apoio e compreensão ao longo desta jornada.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

B2B	Business to Business
B2C	Business to Consumer
ER	Entity-Relationship
DSDM	Dynamic systems development method
DTO	Data Transfer Object
JAD	Joint Application Design
JRP	Joint Requirements Planning
LCDPs	Low-code development platforms
MDE	Model-driven engineering
PaaS	Platform-as-a-service
PWA	Progressive Web App
ORM	Object-Relational Mapping
RAD	Rapid Application Development
SPA	Single Page Application
SOA	Service-Oriented Architecture
SSR	Server Side Render
TDD	Test-Driven Development
XP	Extreme Programming

1 Introdução

Este capítulo apresenta um breve enquadramento do projeto, através da descrição do contexto e do problema, bem como dos objetivos do projeto e da abordagem para a concretização desses objetivos. Para além disso, este capítulo descreve a estrutura deste documento.

1.1 Contexto

O aparecimento do modelo em cascata (Rovce, 1970) revolucionou, na época, o processo de desenvolvimento de software nas organizações. Apesar disso, as limitações deste modelo foram tornando-se evidentes para alguns autores (Gilb, 1985) (Boehm, 1988) (Martin, 1991) (Beck, 2000) que, a partir daí, definiram e impulsionaram o surgimento de novos modelos, métodos e metodologias de desenvolvimento mais adequados às necessidades das organizações.

Desta evolução surgiu a abordagem ao desenvolvimento rápido de aplicações (RAD) (Martin, 1991). Esta abordagem foi definida tendo em conta os fatores velocidade, qualidade e custo e considerando pessoas, competências, ferramentas e metodologias como os elementos essenciais ao processo de desenvolvimento (Martin, 1991). Contudo, o conceito de RAD tem evoluído e, de acordo com (Gartner, 2021), corresponde a uma abordagem de desenvolvimento de aplicações com recurso a *Joint Application Design* (JAD) e prototipagem iterativa para implementação de aplicações de baixa a média complexidade num período de 60 a 120 dias.

Tal como referido, as ferramentas ocupam um papel determinante no desenvolvimento rápido de aplicações, uma vez que se espera que estas facilitem a sua conceção e construção, contribuindo para a redução do tempo de desenvolvimento. Atualmente, os avanços tecnológicos têm permitido o desenvolvimento de ferramentas cada vez mais autónomas e com elevados níveis de abstração desde *frameworks* como Microsoft .NET ou Spring até plataformas de desenvolvimento *no/low-code* [1](Cuba Platform, 2018).

As plataformas de desenvolvimento *low-code* são fornecidas na *cloud* através de um modelo *Platform-as-a-service* (PaaS) e permitem o desenvolvimento e instalação de aplicações através de procedimentos com pouco ou nenhum código (Sahay et al., 2020). Na base destas plataformas estão os princípios de *model-driven engineering* adotados em vários contextos, baseando-se na automação, análise e abstração através de modelos e meta modelos (Sahay et al., 2020).

[1] Ao longo do documento estas plataformas serão referidas simplesmente como *low-code*

1.2 Problema

As mudanças proporcionadas pelo aparecimento de novas tecnologias têm desafiado as organizações, que entendem que têm de ter as melhores ferramentas para poderem responder de forma adequada às transformações tecnológicas (PMI, 2020).

Apesar disso, hoje em dia verifica-se que, embora haja um aumento significativo no investimento de projetos de software, existem projetos que não são entregues dentro do prazo previsto ou de acordo com o orçamento estabelecido e não oferecem valor ao cliente (Ahimbisibwe et al., 2015).

No contexto de mudança contínua, as organizações relacionadas com as Tecnologias de Informação (TI), em particular ao desenvolvimento de sistemas e/ou aplicações, estão cada vez mais pressionadas a minimizar o denominado *time-to-market*, sem comprometer a qualidade da solução. O *time-to-market* corresponde ao tempo que decorre entre uma nova ideia ou necessidade de uma solução e o momento em que essa solução chega ao mercado.

Tal como foi referido na secção anterior, a abordagem RAD considera que as ferramentas são elementos essenciais para a adoção da abordagem de desenvolvimento rápido. Mais recentemente, neste âmbito, têm surgido um conjunto de abordagens, ferramentas e tecnologias, com destaque para as plataformas *low-code*. Por contraponto com os modelos e métodos de desenvolvimento mais atuais e tradicionais, tem sido sugerido que este tipo de abordagens e ferramentas permite que as organizações de TI reduzam significativa o *time-to-market* (Sanchis et al., 2020).

Contudo, ao nível de uma organização de TI a decisão de alterar métodos e procedimentos bem estabelecidos e enraizados no sentido de adotar uma abordagem de desenvolvimento rápido, através de plataformas *low-code*, apresenta algumas dúvidas, reservas e até resistências naturais, tais como: que plataformas existem e quais as mais adequadas ao tipo de aplicações que a organização desenvolve; qual a adequação destas plataformas à metodologia de desenvolvimento utilizada e quais as vantagens e limitações da adoção destas plataformas, em termos de tempo, custos, qualidade e competências técnicas.

O trabalho reportado neste documento visa identificar, sistematizar e procurar obter respostas ou recomendações relevantes para algumas destas dúvidas.

1.3 Objetivos

O principal objetivo deste projeto de dissertação é, por um lado, identificar e sistematizar os principais modelos e metodologias de desenvolvimento de software e as principais ferramentas *low-code* existentes. Por outro lado, pretende-se avaliar a aplicabilidade de plataformas *low-code* nas organizações em diferentes cenários e/ou tipos de projetos, tendo em conta as seguintes questões de investigação:

- **Q1:** Qual a adequabilidade das ferramentas *low-code* para suportar o desenvolvimento de aplicações com diferentes arquiteturas (e.g. monolítica e orientada a serviços)? E, por outro lado, quais são as principais limitações na adoção de cada uma dessas ferramentas?
- **Q2:** Qual o impacto da adoção de uma ferramenta *low-code* ao nível de custo, tempo e qualidade em relação à adoção de uma ferramenta de desenvolvimento tradicional?
- **Q3:** Tendo em conta os seguintes fatores ao nível organizacional: (i) metodologia de desenvolvimento e (ii) perfil de competências da equipa de desenvolvimento, qual o nível de adaptação requerido para a utilização de uma ferramenta *low-code* no processo de desenvolvimento de uma organização?

Assim, espera-se que, no final deste trabalho, seja possível responder às questões de investigação identificadas.

1.4 Abordagem

Neste projeto pretende-se estudar e avaliar a aplicabilidade da adoção de plataformas *low-code* nas organizações. Para tal, pretende-se, numa fase inicial, realizar uma análise da literatura e sistematização da mesma, através do estudo de: (i) principais conceitos e abordagens de desenvolvimento rápido de soluções; (ii) principais métodos e metodologias de desenvolvimento de software existentes e (iii) principais ferramentas *low-code* existentes, bem como os conceitos associados.

Após esta análise pretende-se especificar alguns casos de estudo, focados num determinado domínio generalista, como base para a construção de soluções/aplicações informáticas. Apesar da implementação destes casos de estudo ser através de plataformas *low-code*, a sua definição não deve ter em conta as características ou limitações das plataformas, uma vez que se

pretende validar a aplicabilidade destas num cenário possível de desenvolver através de metodologias e ferramentas tradicionais.

De seguida, pretende-se desenvolver os casos de estudo nas plataformas *low-code* mais promissoras – Outsystems e Mendix – de forma a potenciar a sua comparação e verificar o nível de adequação à arquitetura definida, bem como ao tipo de projeto, através da experiência pessoal nessas plataformas.

Por fim, pretende-se comparar as plataformas em análise ao nível da adequabilidade de arquiteturas, curva de aprendizagem, usabilidade e custos. Para além disso, com recurso à literatura existente e tendo em conta a experiência pessoal com as ferramentas já citadas pretende-se analisar, de forma crítica possíveis respostas para as questões de investigação Q2 e Q3.

1.5 Estrutura do documento

O presente documento apresenta-se dividido em sete capítulos:

- **Introdução:** apresenta e enquadra o projeto de dissertação ao nível do contexto e problema, bem como os seus principais objetivos e a abordagem seguida para a concretização dos objetivos;
- **Modelos e metodologias de desenvolvimento de software:** apresenta uma breve descrição e um enquadramento histórico e evolutivo de alguns dos principais modelos e metodologias de desenvolvimento existentes;
- **Ferramentas de desenvolvimento *low-code*:** apresenta uma breve descrição dos princípios de ferramentas *low-code*, bem como uma descrição da sua arquitetura e do processo de desenvolvimento comum nestas ferramentas. Para além disso, este capítulo descreve as principais plataformas *low-code* existentes no mercado, tendo em conta as suas características;
- **Casos de estudo:** apresenta a descrição dos casos de estudo definidos no âmbito deste projeto, através da descrição do negócio do problema, dos requisitos funcionais e não-funcionais e da descrição da arquitetura do sistema;
- **Implementação:** apresenta a descrição da implementação e experiência das soluções nas plataformas em estudo - Outsystems e Mendix;

- **Comparação das plataformas:** apresenta a comparação das plataformas em estudo, tendo em conta metodologia de desenvolvimento, arquitetura de soluções, aprendizagem e custos e subscrições;
- **Conclusão:** apresenta um breve sumário do trabalho desenvolvido, bem como dos objetivos alcançados, aas limitações identificadas e do trabalho futuro.

Adicionalmente, no Anexo 5 - Análise de Valor apresenta-se o capítulo Análise de Valor que apresenta uma análise dos cinco elementos de *New Concept Development* (NCD) no contexto do projeto, o valor do projeto para o cliente e a análise de negócio;

2 Modelos e metodologias de desenvolvimento de software

Este capítulo apresenta uma breve descrição e enquadramento histórico de algumas das principais metodologias de desenvolvimento de software. Este enquadramento é realizado de forma que sejam apresentadas não só as vantagens de cada modelo ou metodologia, mas também as principais limitações que estimularam o aparecimento de novas perspetivas.

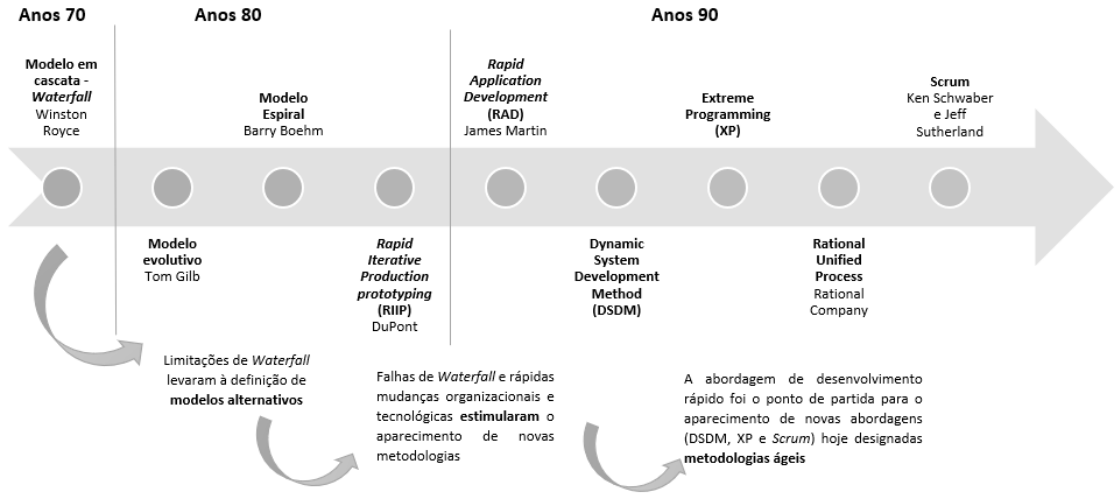


Figura 1 Vista cronológica da evolução de métodos e metodologias de desenvolvimento

A Figura 1 apresenta, por ordem cronológica, os métodos e metodologias descritos nas subsecções seguintes. Assim, o enquadramento inicia-se com a descrição do modelo em cascata, modelo atualmente considerado metodologia tradicional de desenvolvimento. Segue-se a descrição dos modelos alternativos que surgiram numa tentativa de colmatar as falhas do modelo em cascata. De seguida, descreve-se a metodologia RAD. O aparecimento desta abordagem contribuiu para a evolução de novos abordagens e métodos (e.g. DSDM, XP e *Scrum*) conhecidos, atualmente, como metodologias ágeis. Desta forma, o enquadramento final pretende descrever a evolução que sucedeu depois de RAD.

2.1 Modelo em cascata

O modelo em cascata, também conhecido por *Waterfall*, surgiu em meados dos anos 70 através da publicação (Rovce, 1970). Este modelo popular segue uma abordagem clássica do ciclo de

vida do desenvolvimento de *software*, através da utilização de um método de desenvolvimento linear e sequencial (Zaminkar and Reshadinezhad, 2013).

Neste modelo cada etapa do ciclo de vida de desenvolvimento produz resultados e começa apenas quando a anterior estiver completa (Despa, 2014), com ênfase no planeamento nas etapas iniciais (Zaminkar and Reshadinezhad, 2013). O modelo apresenta-se dividido por etapas, sendo que a sua apresentação varia, conforma a interpretação de vários autores, tal como se pode verificar por comparação da Figura 2 com a Figura 3.

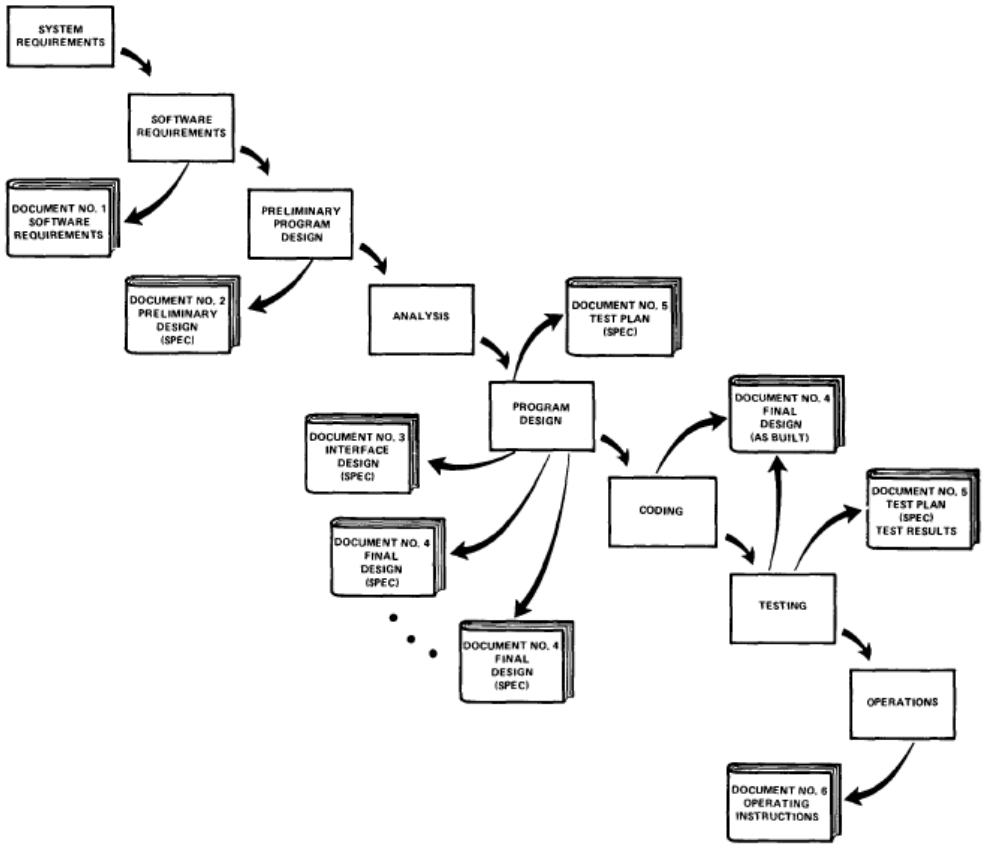


Figura 2 Modelo em cascata definido por (Rovce, 1970)

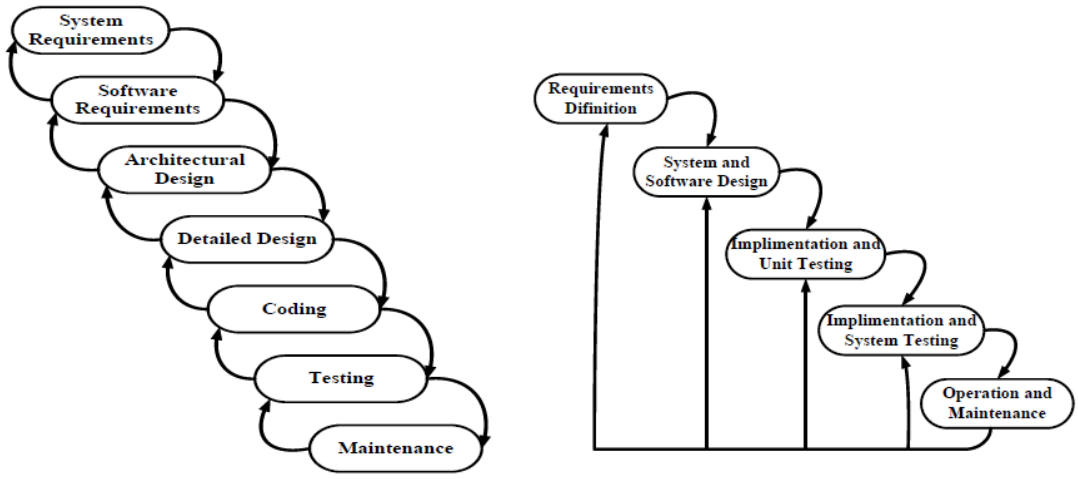


Figura 3 Modelos em cascata definidos por *National Instruments Corporation (2006)* e por *CTG (1998),* respetivamente, citados em (Zaminkar and Reshadinezhad, 2013)

Seguindo a orientação do primeiro modelo definido na Figura 3, as etapas que completam este modelo são:

- ***System Requirements***: são definidos os componentes necessários para o desenvolvimento do projeto, tais como ferramentas de *software* e requisitos de *hardware*;
- ***Software Requirements:*** especificação de requisitos e definição das interações necessárias com outras aplicações e bases de dados, por exemplo;
- ***Architecture design***: especificação dos detalhes da arquitetura da solução;
- ***Detailed design:*** especificação dos detalhes de implementação de cada componente;
- ***Coding:*** implementação das especificações definidas anteriormente:
- ***Testing:*** validação de eventuais erros no código da solução, assim como de validação do cumprimento das especificações definidas na solução;

  Maintenance: ocorre depois da entrega da solução e tem como objetivo o tratamento de problemas e pedidos de melhoria da solução.

Segundo (Boehm, 1988) o modelo de cascata contribuiu para ajudar a ultrapassar as dificuldades encontradas nos projetos de *software*, tornando-se um modelo base para o desenvolvimento de software na indústria e ao nível governamental. Contudo, apesar das revisões e aperfeiçoamentos do modelo, (Boehm, 1988) considera que uma das dificuldades deste modelo é o ênfase dado à elaboração de documentos como critério para concluir uma

etapa, uma vez o autor entende que a necessidade destes documentos pode variar dependendo dos componentes de software.

As limitações deste modelo levaram à definição de modelos alternativos ao existente. Um dos modelos alternativos foi o *evolutionary development model* (Gilb, 1988) que define incrementos no ciclo de vida do modelo de cascata, sendo que as direções da evolução do desenvolvimento do produto são determinadas pela experiência (Boehm, 1988). Este modelo está adaptado para situações em que o cliente não sabe e não define todos os requisitos necessários (Boehm, 1988). Outro modelo alternativo que surgiu foi o *transform model* que se apresentou como uma solução para os problemas de "*spaghetti code*" do desenvolvimento evolutivo e modelos *code-and-fix* (Boehm, 1988). Este modelo assume a capacidade de converter especificações formais em código que satisfaça as especificações de forma automática (Boehm, 1988). A definição destes dois modelos alternativos e as suas limitações levaram (Boehm, 1988) a definir o modelo espiral (cf. 2.2 Modelo Espiral).

(Patel and Jain, 2013) apresentam uma proposta de modelo em cascata, representada na figura seguinte:

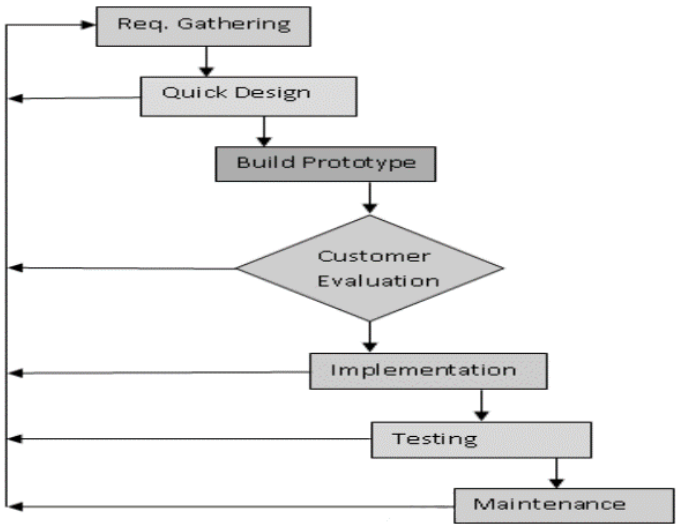


Figura 4 Proposta de modelo em cascata por (Patel and Jain, 2013)

Para (Patel and Jain, 2013) o sucesso deste modelo adaptado está na rigorosa validação de requisitos e a verificação de cada versão de acordo com os requisitos no final de cada fase. Um dos pontos fortes deste modelo é o facto do utilizador ver um protótipo da solução, antes do

desenvolvimento final da mesma, sendo que o utilizador dá o seu *feedback* e, qualquer alteração pode ser realizada antes da fase de desenvolvimento (Patel and Jain, 2013).

Apesar da definição de novos modelos alternativos, o modelo em cascata apresenta-se como um modelo simples, fácil de entender e de gerir (Despa, 2014) (Zaminkar and Reshadinezhad, 2013), sendo um dos modelos de desenvolvimento de software mais utilizados nas organizações (Vijayasarathy and Butler, 2016).

No entanto, vários autores têm apresentado outra visão deste modelo, detetando as principais limitações e desafios do mesmo. Em (van Vliet, 2007) o autor faz referência a *McCracken and Jackson* (1981) que comparam o modelo em cascata com uma loja em que o cliente é obrigado a comprar quando entra, sem ter oportunidade de comparar preços, olhar em volta ou mudar de ideias. O mesmo autor refere ainda que o modelo em cascata é irrealista, justificando que, na maioria dos projetos de desenvolvimento, as fases definidas pelo modelo não são respeitadas (van Vliet, 2007). Por outro lado, (Patel and Jain, 2013) afirmam que a fixação de requisitos antes da fase de design e implementação é demasiado idealista e impraticável.

(Petersen et al., 2009) apresenta as principais limitações do modelo em cascata descritas na literatura existente, sendo as mais citadas as seguintes:

- Dificuldade em responder à mudança;
- Quando o cliente descobre problemas numa fase tardia, já com a solução a ser utilizada, sendo que esta não satisfaz os requisitos;
- Elevado esforço e custo de produção e aprovação de documentos em cada fase de desenvolvimento e integração *big-bang* e testes no final do projeto podem originar problemas de qualidade, custos elevados e atrasos.

Para além disso, quando são detetados componentes em falta durante a implementação há um aumento da necessidade de desenvolvimento e, consequentemente, aumento de custos (Patel and Jain, 2013). Por outro lado, quando o cliente apenas pode ver a solução depois da fase de implementação (Patel and Jain, 2013), sendo que, se pretender alterar um dos requisitos já não o fará no processo de desenvolvimento atual (Balaji, 2012). Outra limitação está relacionada ao facto dos problemas não serem resolvidos imediatamente numa etapa, havendo pouca tolerância para erros de design e planeamento (Despa, 2014), o que resulta numa solução mal estruturada (Balaji, 2012).

2.2 Modelo Espiral

Tal como descrito na secção 2.1 Modelo em cascata as limitações do modelo em cascata levaram ao desenvolvimento de modelos alternativos, como *evolutionary development model* e *transform model*. Na sua publicação (Boehm, 1988) contextualiza a definição do modelo espiral com as limitações do modelos em cascata e dos modelos alternativos. Para o autor o *evolutionary development model* é facilmente confundido com o modelo *code-fix*, cujas limitações – *spaghetti code* e falta de planeamento – foram as principais motivações para a definições do modelo em cascata (Boehm, 1988). Este modelo e o *transform model* são, na ótica de (Boehm, 1988), irrealistas na forma de se considerar que o sistema será suficientemente flexível para evoluir para direções não planeadas.

A identificação destas limitações permitiu a definição do modelo espiral. Este modelo foi definido de forma a melhorar o modelo em cascata, com base na experiência dos vários aperfeiçoamentos ao modelo aplicados em projetos de *software* e com base na combinação dos modelos anteriores (Boehm, 1988). Este modelo apresenta uma abordagem orientada para o risco - *risk-driven* - com foco na análise de alternativas e na identificação de objetivos (Natarajan et al., 2013). O ciclo de vida de desenvolvimento do modelo espiral apresenta quatro fases e encontra-se representado na Figura 5.

De acordo com (Boehm, 1988) cada ciclo é iniciado com a identificação dos objetivos do produto a implementar, definição de alternativas de implementação e as suas limitações, em termos de custo, por exemplo. De seguida, são avaliadas as alternativas em relação aos objetivos e identificadas as áreas que constituem um risco para o projeto. Segue-se o desenvolvimento evolutivo, com o planeamento e desenvolvimento de protótipos que permitam resolver as questões de maior risco. Assim que os protótipos resolvam os riscos identificados seguem-se os passos da abordagem ao modelo em cascata, sendo que, cada nível de especificação é seguido de uma etapa de validação e preparação para o ciclo seguinte.

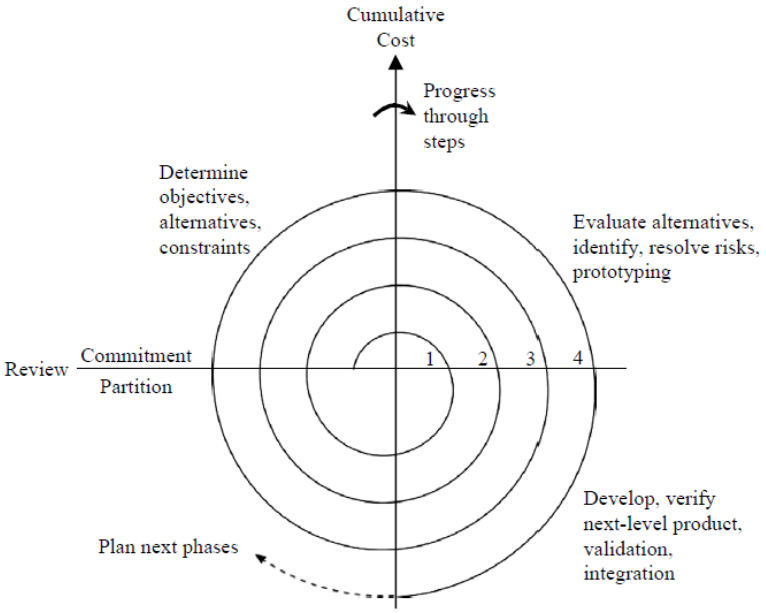


Figura 5 Fases do modelo espiral (Nilsson and Wilson, 2012)

As principais vantagens deste modelo estão relacionadas com documentação sólida, minimização de riscos (Despa, 2014), eliminação de erros e alternativas poucos credíveis numa fase inicial, foco para a reutilização de software, estimulada na identificação e avaliação de alternativas (Boehm, 1988). Para além disso, este modelo é relevante pelo ser valor conceptual, sendo que acredita que desenvolvimentos futuros podem validar estar abordagem (Nilsson and Wilson, 2012).

Por outro lado, este modelo apresenta limitações, tais como: custos gerados pela gestão de riscos; dependência da precisão da análise de risco (Despa, 2014), sendo que é depositada confiança na capacidade da equipa para identificar e gerir os riscos do projeto e necessidade de maior elaboração das etapas, ao nível de especificações e definição de prazos (Boehm, 1988). Para além disso, este modelo descreve um processo, não estando adaptado para controlar um projeto (Nilsson and Wilson, 2012).

2.3 Rapid Application Development

As falhas do modelo em cascata, juntamente com as rápidas mudanças organizacionais e tecnológicas, foram um estímulo para a definição de RAD (Jones and King, 1998). As definições de modelos como modelo evolutivo, modelo espiral (Boehm, 1988) e modelo de protótipos levaram à definição de uma metodologia designada *Rapid Interative Production Prototyping*

(RIIP) (Kerr and Hunter, 1994). Esta metodologia surgiu em meados dos anos 80 e foi formalizada pela *DuPont*, tendo sido definidas, pela primeira vez, técnicas que, coletivamente, vieram a ser conhecidas como RAD (Kerr and Hunter, 1994).

Baseado no trabalho desenvolvido na DuPont, James Martin publica, em 1991, o livro *Rapid Application Development* (RAD) (Martin, 1991), onde surgiu o termo RAD pela primeira vez. Para (Martin, 1991) RAD "refere-se ao ciclo de vida de desenvolvimento projetado para fornecer um desenvolvimento muito mais rápido e resultados de maior qualidade do que o ciclo de vida tradicional. É projetado para tirar o máximo proveito de poderoso software de desenvolvimento que evoluiu, recentemente"[2].

Figura 6 Relação dos principais objetivos de RAD baseada em (Martin, 1991)

(Martin, 1991) realça que o termo RAD se refere a uma metodologia e tem como principais objetivos: (i) alta velocidade, (ii) alta qualidade e (iii) baixo custo. A Figura 6 apresenta a relação desses três objetivos que estão interligados durante a implementação desta metodologia. Esta relação leva (Martin, 1991) a referir que "esta é uma situação verdadeiramente vantajosa para

[2] Tradução livre da autora. No original "RAD (Rapid Application Development) referes to a development lifecycle designed to give much faster development and higher quality results than the tradicional lifecycle. I tis designed to take maximum advantage to powerful development software that has evolved recently. There are variations on the RAD lifecycle, depending on the nature of the system."

ambas as partes. Os sistemas assim desenvolvidos satisfazem melhor as necessidades dos utilizadores e têm custos de manutenção mais baixos"[3].

Para o desenvolvimento rápido considera-se fundamental a participação de pessoas bem preparadas e com competências adequadas, bem como uma gestão adequada de forma a evitar burocracias que atrasem o desenvolvimento. Para além disso, as ferramentas, combinadas com adequadas metodologias, são um aspeto importante a considerar, uma vez que estas serão necessárias para a criação das aplicações e/ou sistemas. Assim, (Martin, 1991) considera pessoas, gestão, metodologia e ferramentas os quatro aspetos essenciais para a metodologia RAD.

A definição desta metodologia assenta em alguns conceitos fundamentais para a implementação da mesma, tais como:

- ***Timebox***: o termo *timebox* foi utilizado a primeira vez pela *DuPont* sendo que, após o sucesso da sua aplicação foram utilizadas outros técnicas derivadas de *timebox* em diversas organizações (Martin, 1991). Ao nível de RAD o *timebox* é aplicado na fase de construção e representa o intervalo de tempo, tipicamente curto, no qual o sistema deverá ser implementado (ver Figura 7) (Martin, 1991). O principal objetivo desta técnica de gestão de tempo é a divisão dos objetivos da implementação em *timeboxes* (Simao, 2011). Contudo, caso se verifiquem atrasos em relação ao previsto há uma redução e ajuste dos requisitos, de forma a não se aumentar o *timebox* (Beynon-Davies et al., 1999).
- **Protótipo**: corresponde a uma técnica para implementar uma versão rápida de um determinado sistema que permite que o cliente tenha uma interação com o sistema, de forma a serem revistas as interações do utilizador com o sistema (Martin, 1991) . O protótipo é utilizado entre os profissionais de TI e os utilizadores, de forma a serem discutidas eventuais alterações (Beynon-Davies et al., 1999). Esta técnica poderá ser utilizada em qualquer uma das fases de desenvolvimento, no entanto, é utilizada, de uma forma mais intensa, na fase *User Design*, bem como na fase *Requirements Planning*, de forma a estimular as ideias do utilizador (Martin, 1991).

[3] Tradução livre da autora. No original "This is truly a win-win situation. Systems so developed *meet the needs of the users better ans have lower maintenance costs*."

- ***Joint Requirements Planning (JRP):*** corresponde a um *workshop* onde se definem os requisitos e se detalham as funcionalidades do sistema a implementar, sem detalhes técnicos, onde participam os representantes do cliente que têm conhecimento geral das necessidades do negócio (Martin, 1991).
- ***Joint Application Design (JAD):*** corresponde a um workshop onde são definidos os detalhes de design do sistema, tais como modelo de dados, especificações detalhadas, interfaces gráficas e protótipos (Martin, 1991). Neste workshop participam os utilizadores que serão proprietários ou que trabalharão com o sistema e os profissionais de TI (Martin, 1991) (Beynon-Davies et al., 1999). Durante o desenvolvimento de um projeto podem ser agendados workshops deste tipo para cada entrega prevista (Beynon-Davies et al., 1999)

Segundo (Martin, 1991), o ciclo de vida de RAD apresenta-se dividido em quatro fases:

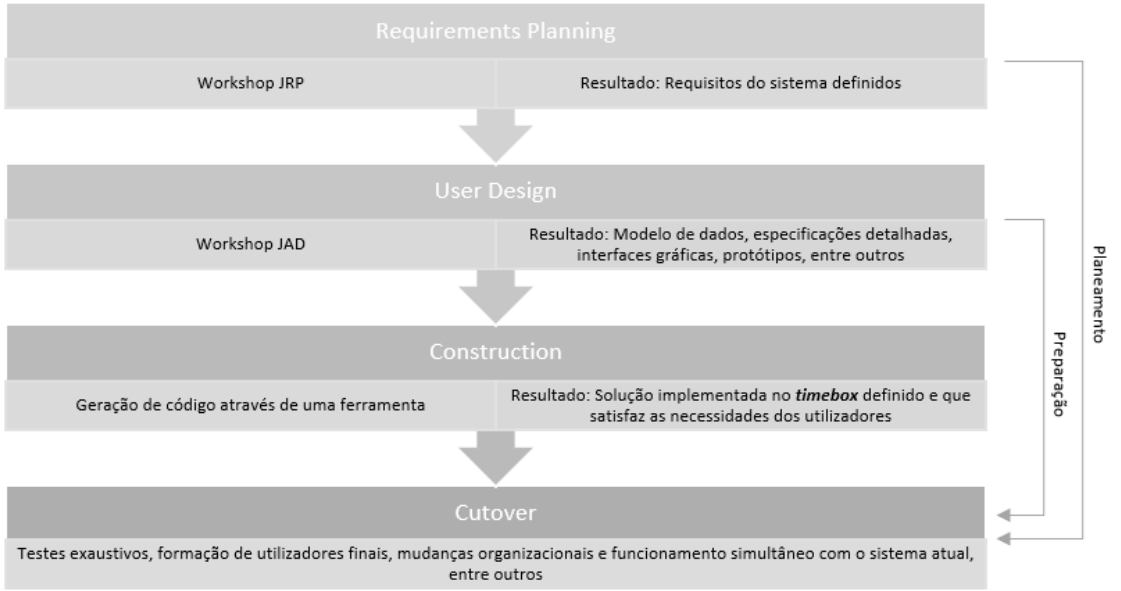


Figura 7 Etapas do ciclo de vida de RAD baseadas em (Martin, 1991)

- ***Requirements Planning*****:** nesta fase é realizado o workshop *Joint Requirements Planning* (JRP) onde o cliente, com o acompanhamento dos profissionais de TI, expõe as suas necessidades que resultarão em requisitos do sistema a desenvolver;
- ***User Design*****:** nesta fase é realizado o workshop *Joint Application Design* (JDA) e são utilizadas ferramentas I-CASE e protótipos. Os profissionais de TI recorrem às ferramentas I-CASE para captarem requisitos sob a forma de protótipo, de forma que os utilizadores interajam com os mesmos e contribuam para a especificação dos

requisitos. No final desta fase deve resultar um desenho apresentado numa ferramenta I-CASE, que satisfaça as necessidades de todos os intervenientes. Depois, o desenho final deverá ser direcionado para a fase seguinte, onde ocorre a geração de código;

- ***Construction*:** nesta fase os profissionais de TI realizam a geração de código através da ferramenta I-CASE. Com a utilização deste tipo de ferramenta o desenho detalhado é elaborado no ecrã e o código é gerado a partir do mesmo. Desta forma, verifica-se que com RAD a relação entre o design e a implementação de código é diferente da relação verificada num ciclo de vida de desenvolvimento tradicional, onde a fase de design é separada da fase de implementação.

  Durante esta fase são realizados testes ao sistema gerado. Para além disso, há envolvimento contínuo dos utilizadores finais que validam os ecrãs, por exemplo, e são informados caso se verifiquem alterações significativas nos requisitos. Este envolvimento permite assegurar que o produto final satisfaz as necessidades do cliente;
- ***Cutover*:** durante esta fase realizam-se algumas atividades, tais como: testes exaustivos, formação de utilizadores finais, mudanças organizacionais e funcionamento em paralelo com o sistema atual. Uma parte do planeamento de *cutover* é realizado na fase de *Requirements Planning* e os detalhes de preparação de *cutover* começam a ser definidos na fase de *User Design*.

Após a publicação de James Martin seguiram-se vários autores que, baseados no seu trabalho, apresentaram novas abordagens para RAD. Uma dessas abordagens é dos autores (Kerr and Hunter, 1994) que referem que RAD "é uma abordagem à construção de sistemas informáticos que combina ferramentas CASE e técnicas, protótipos orientados para o utilizador e rigorosos prazos de entrega de projetos (…). RAD aumenta drasticamente a qualidade dos sistemas desenvolvidos, enquanto reduz o tempo necessário para os construir"[4] .

Para estes autores o ciclo de vida de RAD apresenta cinco fases (Kerr and Hunter, 1994): (i) *Business Modeling*, (ii) *Data Modeling*, (iii) *Process Modeling*, (iv) *Application Generation* e (v) *Testing and turnover*. Na primeira fase há o entendimento e modelação de todos os processos do negócio, bem como o levantamento de requisitos (Simao, 2011). Na segunda fase é definido

[4] Tradução livre da autora. No original "RAD is an approach to building computer systems which combines Computer-Assisted Software Engineering (CASE) tools and techniques, user-driven prototyping, and stringent project delivery time limits (…). Rad drastically raises the quality of finished systems while reducing the time it takes to build them"

o modelo de dados que suporta o negócio, baseado no resultado da fase anterior (Kerr and Hunter, 1994). A fase seguinte corresponde à definição da lógica a utilizar para o criação, eliminação, atualização e recuperação dos dados definidos no modelo anterior (Kerr and Hunter, 1994). De seguida, a fase *Application Generation* corresponde à implementação da solução através de ferramentas que permitem tornar a implementação mais rápida e reduzir o aparecimento de erros, o que aumentará a qualidade do produto final (Kerr and Hunter, 1994). Para além disso, destaca-se a importância de reutilizar componentes de *software*, de forma a reduzir a quantidade de testes necessários (Kerr and Hunter, 1994). Por fim, na fase *Testing and Turnover* são testados todos os componentes e são feitas alterações necessárias após a realização dos testes finais antes da entrega da solução (Kerr and Hunter, 1994).

Para (McConnell, 1996) RAD é "uma combinação de ferramentas CASE, envolvimento intenso do utilizador e *timeboxes* rigorosos" [5] e apresenta quatro dimensões: pessoas, processo, tecnologia e produto. Este autor define processo como um conjunto de gestão e metodologias, contrariamente a (Martin, 1991) que definiu cada uma delas como uma dimensão de RAD. Para além disso, (McConnell, 1996) considera que o foco na dimensão e nas características de um produto podem contribuir para a redução do tempo previsto à sua implementação.

Esta abordagem mostra que existem diferentes níveis de compromisso com a velocidade de desenvolvimento – existem casos em que se pretende aumentar a velocidade sem custos adicionais ou impacto na qualidade do produto (McConnell, 1996). Assim, (McConnell, 1996) mostra que uma oportunidade para um desenvolvimento mais rápido é seguir uma abordagem de desenvolvimento eficiente, focado no equilíbrio entre custo, tempo e funcionalidade. A Figura 8 mostra o caminho das organizações até ao desenvolvimento rápido e, tal como se pode verificar, o caminho onde estavam a maioria das organizações até ao desenvolvimento eficiente é linear, sendo que a partir deste último ponto o caminho diverge entre desenvolvimento rápido, menores custos de desenvolvimento, aumento de satisfação do utilizador e menor número de defeitos (McConnell, 1996). Este autor defende que seria benéfico para as organizações definirem, primeiramente, um desenvolvimento eficiente, de forma a, só mais tarde divergirem para um desenvolvimento rápido (McConnell, 1996).

[5] Tradução livre da autora. No original "it is RAD – a combination of CASE tools, intensive user involvement, and tight timeboxes"

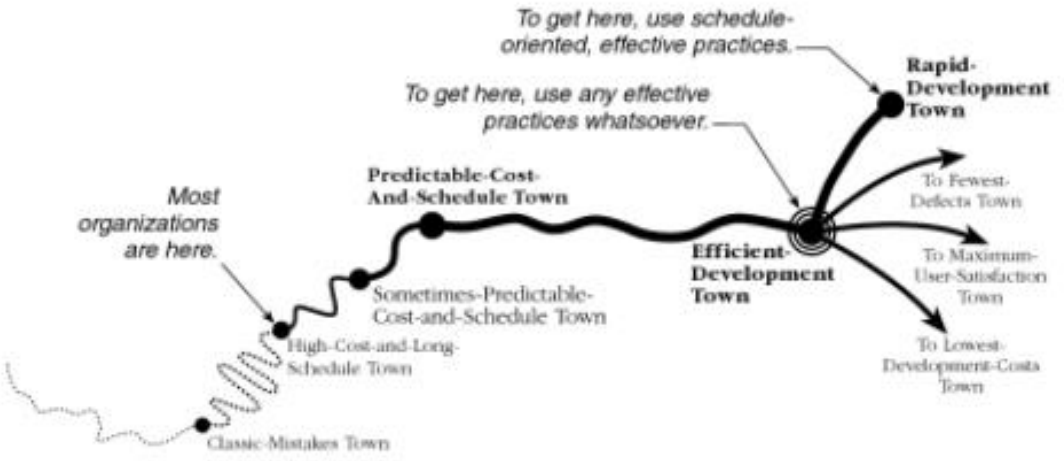


Figura 8 Caminho para o desenvolvimento rápido (McConnell, 1996)

Como citado em (Jones and King, 1998) os autores Ljubic e Stefancic descrevem, em 1994, que o sucesso de RAD depende também do conhecimento dos utilizadores e não apenas da capacidade dos profissionais de TI. Para além disso, estes autores descrevem alguns problemas na implementação de RAD, tais como:

- Dificuldade dos utilizadores em identificarem os requisitos necessários e em dedicarem tempo ao projeto;
- Os profissionais de TI não estão adaptados às ferramentas utilizadas;
- Os módulos desenvolvidos e entregues separadamente podem ser difíceis de integrar.

(Jones and King, 1998) afirmam que esta metodologia não tem uma visão a longo prazo, havendo apenas foco nos requisitos necessários numa determinada altura. Análises a metodologias de desenvolvimento mais atuais mostram que, para além das limitações enunciadas, RAD carece de documentação consistente e pode apresentar elevados custos de desenvolvimento (Despa, 2014).

Numa abordagem mais atual, a (Gartner, 2021) define que RAD é uma abordagem de desenvolvimento de aplicações que inclui equipas pequenas que recorrem a JAD e prototipagem iterativa para desenvolverem aplicações de baixa a média complexidade num período entre 60 a 120 dias.

2.4 Dynamic System Development Method

No seguimento da definição de RAD apresentada por (Martin, 1991) surge, em 1994, a definição do método *Dynamic System Development Method* (DSDM).

DSDM corresponde a um método ágil desenvolvido pelo consórcio DSDM para projetos dinâmicos que, atualmente, incluí também gestão de projetos através de *Agile PM Framework* (Hohl et al., 2018). Este método surgiu com o objetivo de criar um método *standard* que permitisse eliminar abordagens pouco formais associadas a RAD (Beynon-Davies et al., 1999), permitindo dar resposta às necessidade de alguns profissionais da área de desenvolvimento que pretendem aumentar a qualidade dos seus desenvolvimentos com utilização de processos de RAD (Agile Business, 2014). De facto, a utilização da metodologia RAD permitiu o desenvolvimento rápido de soluções. No entanto a qualidade destas soluções não correspondia ao esperado (Agile Business, 2014). Para a definição deste método foram analisados não só os pontos fortes e limitações de RAD, mas também os pontos fortes e limitações da abordagem de desenvolvimento tradicional, o que resultou numa fusão dos pontos fortes destas duas abordagens (Agile Business, 2014).

A filosofia de DSDM refere que "o melhor valor de negócio surge quando os projetos estão alinhados para objetivos claros de negócio, têm entregas frequentemente e envolvem a colaboração de pessoas motivadas e capacitadas"[6] e está assente em oito princípios (Agile Business, 2014): foco na necessidade do negócio, entrega dentro do prazo, colaboração, nunca comprometer a qualidade, implementar de forma incremental, desenvolver iterativamente, comunicar de forma contínua e clara e demonstrar controlo.

O DSDM corresponde a um modelo iterativo e incremental de desenvolvimento que utiliza *timebox*, priorização de tarefas *MoSCow* (Agile Business, 2014) e realiza testes contínuos (Despa, 2014). Este método apresenta-se dividido em seis fases, tal como se pode observar na Figura 9.

[6] Tradução livre da autora. No original "best business value emerges when projects are aligned to clear business goals, deliver frequently and involve the collaboration of motivated and empowered people."

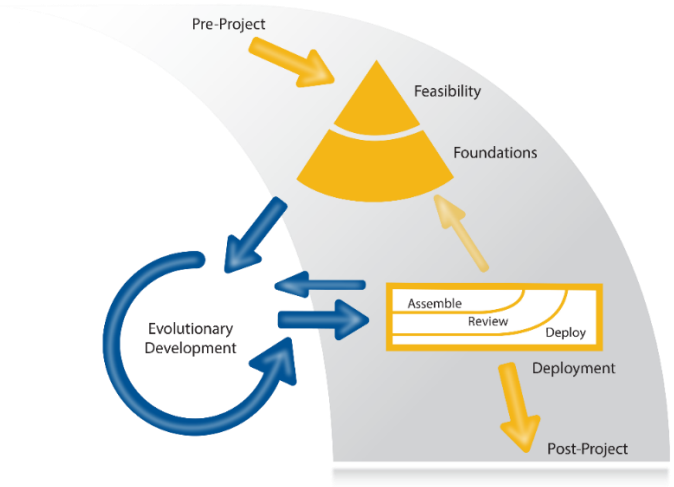


Figura 9 Fases do método DSDM (Agile Business, 2014)

Na fase *pre-project* assegura-se que os projetos são iniciados, com base nos objetivos definidos, seguindo-se a fase *feasibility* onde se determina a viabilidade de um projeto do ponto de vista técnico e de rentabilidade (Agile Business, 2014). Na fase *foundations* é esclarecido o âmbito do projeto – quem o realiza, detalhes de uma possível solução e detalhes da gestão do projeto (Agile Business, 2014). Segue-se a fase *evolutionary development* onde são implementados incrementos de soluções, aplicando práticas de *timebox* e *MoSCoW* (Despa, 2014) (Agile Business, 2014). Na fase de *deployment* a solução final ou apenas uma parte dela é instalada para uso operacional (Agile Business, 2014). Por fim, na fase *post-project* são elaboradas avaliações de benefícios ao nível de negócio (Agile Business, 2014).

A documentação consistente, avaliação no final do projeto e envolvimento do utilizador são alguns dos principais pontos fortes do método DSDM (Despa, 2014). Por outo lado, este método exige profissionais bem capacitados e requer equipas numerosas (Despa, 2014).

2.5 Extreme Programming

A metodologia *Extreme Programming* (XP) (Beck, 2000) surgiu no final dos anos 90 como uma nova forma de desenvolver *software*, destinando-se a pequenas e médias equipas, com foco nas necessidades do cliente e numa eventual mudança rápida de requisitos (Beck, 2000) (Wood and Kleb, 2002). Esta metodologia caracteriza-se pelo elevado envolvimento do cliente durante o processo de desenvolvimento (Matharu et al., 2015).

(Beck, 2000) referiu que "o desenvolvimento de software não é entregue a tempo e não apresenta valor. (…) Temos de encontrar uma nova forma de desenvolver software"[7]. Foi então que surgiu a necessidade da definição de XP, tendo como problema base o risco no desenvolvimento de software associado, por exemplo, a falhas na gestão de tempo, produto entregue sem qualidade e alterações de negócio (Beck, 2000). Para além disso, este modelo foi idealizado tendo por base quatro variáveis: custo, tempo, qualidade e âmbito (Beck, 2000).

O ciclo de vida de desenvolvimento da abordagem XP apresenta-se na Figura 10.

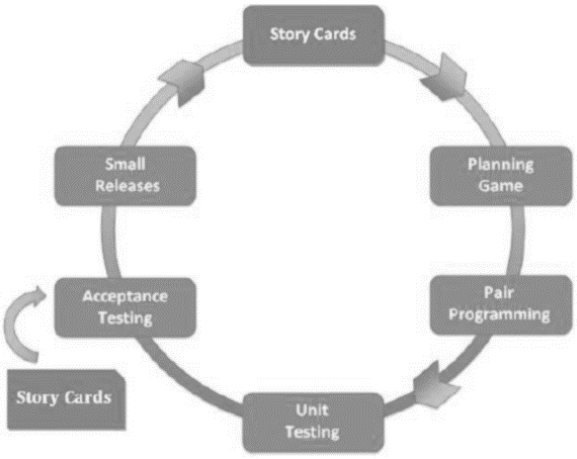


Figura 10 Ciclo de vida de desenvolvimento em Extreme Programming (XP) (Matharu et al., 2015)

Nas fases deste ciclo considera-se apenas o desenvolvimento de uma parte do projeto, permitindo que, mais rapidamente, uma solução parcial seja instalada em produção, evoluindo, num ciclo seguinte, numa direção que seja mais vantajosa (Beck, 2000) (Despa, 2014). Inicialmente, os requisitos são representados como cenários pelos utilizadores e são definidos nas *Story Cards* (Matharu et al., 2015). Em XP é utilizada a abordagem *test driven development* (Beck, 2000) e, dessa forma, os casos de testes são definidos antes da implementação do código, que são executados durante toda a fase de desenvolvimento (Beck, 2000) (Matharu et al., 2015). Na implementação é adotado o *pair programming* onde uma dupla de programadores trabalha de forma dinâmica, resultando numa redução da carga de trabalho e, por consequência, do número de horas trabalhadas (Despa, 2014) (Matharu et al., 2015). Para um melhor design e

[7] Tradução livre da autora. No original "Software development fails to deliver, and fails to deliver value. (…) We need to fin a new way to develop software"

qualidade das soluções, esta metodologia apela ao uso de *refactoring* nas soluções existentes, de forma a diminuir a complexidade e aumentar a fiabilidade do código (Matharu et al., 2015).

(Beck, 2000) afirma que esta metodologia obriga a uma nova visão na forma como é desenvolvido o software. De facto, esta nova visão criou alguma controvérsia, uma vez que, por exemplo, num ciclo de desenvolvimento de XP não se realiza uma análise e definição de design completa do projeto, assim como não existe elaboração de documentação dos desenvolvimentos realizados, uma vez que há um maior foco para a implementação de testes eficientes e código com qualidade (Beck, 2000) (Despa, 2014).

Para além disso, esta metodologia apresenta um desafio para os programadores, uma vez que estes terão de se adaptar ao *pair programming*, assim como ao desenvolvimento de casos de teste antes da implementação do código (Despa, 2014).

2.6 Rational Unified Process

Em meados dos anos 90 a *Rational Company* decidiu uniformizar as várias metodologias existentes orientadas a objetos, análise e design num único método. Este procedimento começou pela definição e publicação de *Unified Modeling Language* (UML), notação para a modelação de *software*, seguindo-se a definição de *Rational Unified Process* (RUP) (Zaminkar and Reshadinezhad, 2013). Atualmente, este processo é suportado pelo *Rational Method Composer* (IBM, 2009).

Este processo tem como objetivo garantir o desenvolvimento de uma solução, dentro do prazo e do orçamento estabelecido, que satisfaça as necessidades dos utilizadores finais (IBM, 2007). Para tal, o RUP apresenta uma abordagem disciplinada de atribuição de tarefas e responsabilidades dentro de uma organização (IBM, 2007). De forma a seguir as melhores práticas, este processo baseia-se nos seguintes princípios: adaptação do processo, equilíbrio entre as prioridades das partes interessadas, colaboração entre as equipas, demonstração de valor de forma iterativa, aumentar o nível de abstração e concentração contínua na qualidade (IBM, 2007). A aplicação destes princípios apresenta alguns benefícios, tais como: eficiência do ciclo de vida, aumento da agilidade do projeto, otimização do valor de negócio, produtividade da equipa, redução do risco precocemente, redução de complexidade e maior qualidade (IBM, 2007).

A Figura 11 apresenta o diagrama do ciclo de vida de RUP.

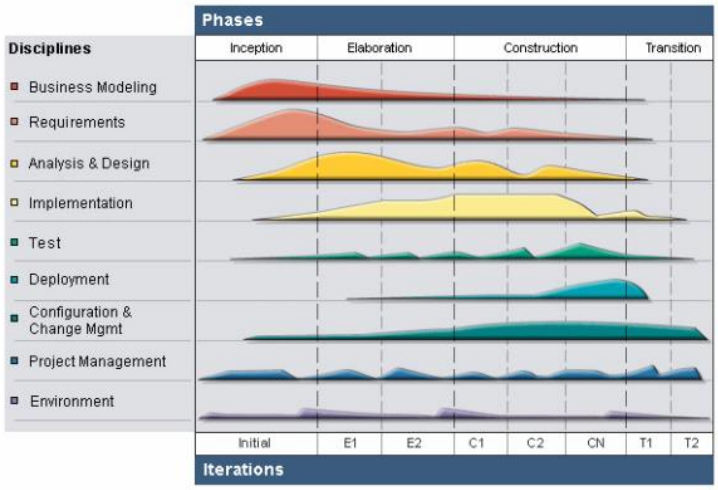


Figura 11 Diagrama do ciclo de vida de RUP (IBM, 2007)

Tal como se pode observar este diagrama apresenta duas dimensões: disciplinas e fases. As disciplinas correspondem às atividades realizadas que resultam na elaboração de artefactos durante o processo, tais como: modelação de negócio, requisitos, análise e conceção, implementação, testes, *deployment*, entre outros (IBM, 2007) (Zaminkar and Reshadinezhad, 2013). RUP é um processo iterativo e incremental, sendo que cada fase do seu ciclo de vida é dividida em iterações que correspondem a intervalos de tempo definidos de acordo com o tipo de projeto (IBM, 2007). Este processo apresenta as seguintes fases (IBM, 2007) (IBM, 2009):

- ***Inception***: nesta fase são definidos, em concordância entre as partes interessadas, os objetivos do ciclo de vida de um projeto, bem como as estimativas de custo e planeamento do projeto e estimativa de potenciais riscos;
- ***Elaboration***: nesta fase é definida e validada a arquitetura, são criados os planos de iteração para a fase seguinte e é posto em prático o ambiente de desenvolvimento;
- ***Construction***: nesta fase é realizado o desenvolvimento e são implementados testes de acordo com os critérios definidos. Para além disso é avaliada a solução final tendo com conta os critérios de aceitação;
- ***Transition***: nesta fase é feito o *deployment* da solução e é entregue uma versão ao cliente que dá o seu feedback, sendo possível a realização de melhorias na solução antes desta estar disponível para os utilizadores.

Esta metodologia dá ênfase à documentação objetiva e permite a resolução de riscos do projeto relativos à evolução dos requisitos pelo cliente, exigindo uma gestão rigorosa dos pedidos de alteração (Zaminkar and Reshadinezhad, 2013). Por outro lado, esta metodologia apresenta algumas limitações, tais como: elevado nível de conhecimento dos elementos da equipa para desenvolverem segundo esta metodologia, o processo de desenvolvimento é complexo e a integração ao longo do processo de desenvolvimento, em projetos grandes e com vários fluxos de desenvolvimento, causa problemas durante a fase de testes (Despa, 2014) (Zaminkar and Reshadinezhad, 2013).

2.7 Scrum

A abordagem *Scrum* (Schwaber and Sutherland, 2020) surgiu em meados dos anos 90 e foi desenvolvida por *Ken Schwaber* e *Jeff Sutherland*. *Scrum* faz parte das metodologias ágeis de desenvolvimento, mas contrariamente a outra abordagens, esta assume-se como uma *framework* que tem como objetivo organizar e gerir trabalho, bem como ajudar pessoas, equipas e organizações a criarem valor através de soluções para problemas complexos (Rubin, 2012) (Schwaber and Sutherland, 2020).

Esta abordagem surge na necessidade de criar uma alternativa a metodologias tradicionais, como modelo em cascata que, segundo (Rubin, 2012) funciona bem quando aplicado a problemas bem definidos, previsíveis e com baixa possibilidade de sofrerem alterações significativas. Contudo, o problema está relacionado ao facto da maioria dos desenvolvimentos não serem previsíveis, sobretudo numa fase inicial e ao facto desta abordagem tradicional se basear em ideais que não dão resposta à incerteza presente em grande parte dos projetos de desenvolvimento (Rubin, 2012).

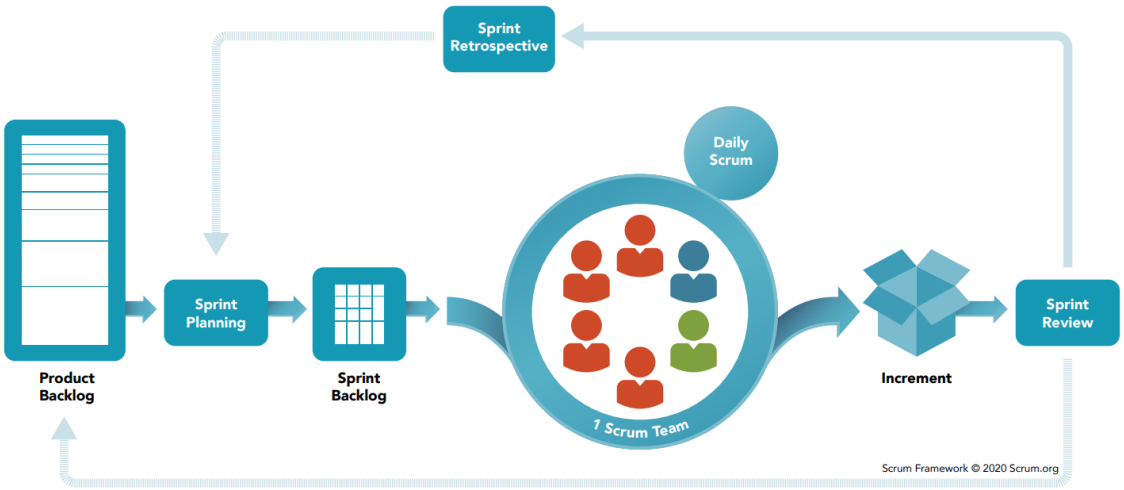


Figura 12 Representação gráfica de Scrum (Scrum.org, 2020)

Tal como se pode observar na Figura 12, o *Scrum* é composto pelos seguintes elementos (Matharu et al., 2015) (Schwaber and Sutherland, 2020):

- ***Scrum Team***: corresponde a uma equipa pequena composta por um *Scrum Master*, um *Product Owner* e desenvolvedores. Estes últimos são responsáveis pela criação do *Sprint Backlog* e pela realização das tarefas atribuídas. O *Scrum Master* é responsável pelo cumprimento dos princípios de *Scrum* e ajuda a eliminar impedimentos no progresso da equipa. O *Product Owner* é responsável pela gestão do *Backlog* e por maximizar o valor do produto realizado pela equipa;
- ***Scrum Events***: correspondem a eventos realizados durante o ciclo de desenvolvimento, tais como:
    - ***Sprint:*** é um evento de duração fixa de um mês ou menos onde se realizam as seguintes atividades: *Sprint Planning*, *Daily Scrum*, *Sprint Review* e *Sprint Retrospective*;
    - ***Sprint Planning:*** evento que determina o início de uma *sprint* e define o trabalho a realizar durante a mesma;
    - ***Daily Scrum:*** evento de quinze minutos realizada, idealmente, no mesmo horário e durante todos os dias úteis de uma *sprint*. O objetivo deste evento é acompanhar o progresso do trabalho, com recurso ao *Sprint Backlog*;
    - ***Sprint Review:*** evento onde são revistos os resultados da *sprint* e onde se determina o que fazer a seguir;

- ***Sprint Retrospective:*** evento que tem como objetivo a reflexão sobre a última *sprint* e o planeamento de novas formas de aumentar a qualidade e a eficácia.

- ***Product Backlog****:* corresponde a uma lista ordenada das tarefas a realizar que podem ser, por exemplo, requisitos funcionais e não funcionais ou *bug fixes*;
- ***Sprint Backlog****:* é composto pelo objetivo da *sprint*, pelo conjunto de elementos do *product backlog* selecionados para um *sprint* e pelo plano para a entrega de uma *sprint*.

O *Scrum* é, atualmente, a metodologia ágil mais adotada pelos desenvolvedores (Waheed et al., 2018). Esta escolha deve-se ao facto desta metodologia conseguir dar uma resposta rápida às necessidades do mercado, apresentando soluções em ciclos de desenvolvimento curtos, permitindo economizar custos (Despa, 2014) (Waheed et al., 2018). Para além disso, a implementação é incremental e iterativa e facilita a adaptação a mudanças o que, juntamente com a participação do cliente durante o todo o processo, aumenta a satisfação das necessidades do cliente (Waheed et al., 2018).

Por outro lado, a gestão de equipas com *Scrum* pode revelar-se uma tarefa difícil, uma vez que, não havendo compromisso de todos os elementos o sucesso do projeto fica comprometido (Waheed et al., 2018). Para além disso, as equipas requerem elementos tecnicamente experientes (Despa, 2014). Outra limitação associada ao *Scrum* está relacionada com a dificuldade em estimar esforço e custo, numa fase inicial, sobretudo para projetos de maior dimensão (Despa, 2014).

2.8 Comparação de modelos e metodologias de desenvolvimento de software

O aparecimento do modelo em cascata contribuiu para se estabelecer um modelo base para o desenvolvimento de software na época da sua definição (Boehm, 1988). Contudo e apesar de ser ainda, atualmente, um dos modelos mais utilizados (Vijayasarathy and Butler, 2016) apresenta limitações que têm sido exploradas, resultando no aparecimento de novos modelos e metodologias. De facto, a descrição nas subsecções anteriores mostra que, na sua maioria, os modelos e metodologias definidos depois do modelo em cascata tiveram como ponto de partida os pontos fortes e os pontos fracos desse modelo tradicional de desenvolvimento. Para além disso, o aumento do ritmo das transformações sociais, económicas, políticas e tecnológicas também contribuiu e continua a contribuir para o aparecimento de novas abordagens no desenvolvimento de software.

Um dos primeiros modelos definidos para dar resposta às limitações do modelo em cascata foi o modelo espiral que foi definido seguindo uma abordagem orientada ao risco, com foco na análise de alternativas. Com base no modelo espiral e outros modelos alternativos surgiu a metodologia RAD que focou os seus objetivos em três atributos: **custo**, **tempo** e **qualidade**.

É possível verificar que os modelos e metodologias definidos depois de RAD foram, direta ou indiretamente, influenciados por essa abordagem. Esta influência está, por exemplo, no foco nos atributos custo, tempo e qualidade no processo de desenvolvimento.

O método DSDM formaliza as práticas de RAD e define como principal objetivo a melhoria da qualidade das soluções entregues. Segue-se a metodologia XP cuja principal preocupação foi a entrega de soluções a tempo e com valor. Numa perspetiva semelhante, RUP foi estabelecido de forma que o desenvolvimento de soluções ocorra dentro de um prazo e orçamento estabelecidos, com foco contínuo na qualidade e nas necessidades do cliente. Por fim, o *Scrum* foca-se no desenvolvimento em ciclos curtos e em dar resposta às incertezas existentes durante o processo de desenvolvimento.

Tal como se pode verificar, as abordagens de cada modelo descrito variam entre si, quer em termos de princípios e objetivos, quer em termos da sua aplicação. Para além disso, é possível verificar que existe um conjunto de características que, com maior ou menor ênfase, tipicamente, estão presentes nestes modelos e metodologias, tais como:

- **Gestão de Projeto:** caracterização do ciclo de vida do projeto, bem como relação com o cliente, ao nível de comunicação e do seu envolvimento no projeto;
- **Requisitos:** a especificação de requisitos pode realizar-se para a totalidade do projeto, ou apenas para uma iteração. Para além disso, consideram-se as validações e entregas ao cliente, assim como a flexibilidade e a capacidade de resposta à mudança dos requisitos durante o processo de desenvolvimento;
- **Documentação:** a documentação nos projetos pode ser utilizada para uma adequada orientação de novos elementos;
- **Desenvolvimento:** no processo de desenvolvimento são utilizados diferentes tipos de ferramentas. Para além disso, em alguns modelos considera-se a utilização de protótipos e reutilização de componentes;
- **Gestão de Equipa:** o nível de competências técnicas necessárias varia, havendo modelos e metodologias que exigem elementos com elevadas competências técnicas;

- **Qualidade:** realização de testes e acompanhamento na correção e tratamento de *bugs* durante o ciclo de vida de desenvolvimento do projeto;
- **Gestão de risco:** a exposição ao risco difere nos modelos descritos, na medida em que, em alguns casos, existe uma orientação ao risco, havendo a análise e minimização de eventuais riscos;
- **Tempo:** duração do processo de desenvolvimentos;
- **Custo:** o custo associado ao processo de desenvolvimento, podendo variar, sobretudo quando são detetados atrasos ao longo do projeto;
- **Simplicidade:** o nível de simplicidade do modelo está diretamente associado às técnicas, procedimento e abordagens associadas ao mesmo e à sua adoção, que pode ser considerada simples ou não.

A Tabela 1 apresenta a comparação dos modelos e metodologias descritos segundo as características mencionadas anteriormente. Esta comparação é o resultado da interpretação do estudo destes modelos e metodologias.

Tabela 1 Comparação de modelos e metodologias de desenvolvimento de *software*

Características	**Modelo em cascata**	**Modelo Espiral**	**RAD**	**DSDM**	**XP**	**RUP**	**Scrum**
Gestão de Projeto							
Ciclo de vida do projeto	Preditivo	Iterativo	Adaptativo	Adaptativo	Adaptativo	Adaptativo	Adaptativo
Comunicação com cliente	Formal	Formal	Informal	Informal	Informal	Informal	Informal
Envolvimento do cliente	Início e fim	Início e fim	Contínuo	Contínuo	Contínuo	Contínuo	Contínuo
Requisitos							
Especificação de requisitos	Etapa inicial (todo o projeto)	Etapa inicial	Etapa inicial (JDA)	Etapa inicial	Etapa inicial	Etapa inicial	Etapa inicial
Validação de requisitos pelo cliente	Final	Final	Contínua	Contínua	Contínua	Contínua	Contínua
Resposta às necessidades	Baixa	Média	Alta	Alta	Alta	Alta	Alta
Flexibilidade/resposta à mudança	Baixa	Média	Alta	Alta	Alta	Alta	Alta
Entregas ao cliente	Final	Final	Contínuas	Contínuas	Contínuas	Contínuas	Contínuas

Características	Modelo em cascata	Modelo Espiral	RAD	DSDM	XP	RUP	Scrum
Documentação							
Tipo de documentação	Sólida	Sólida	Pouca consistente	Consistente	Não existe	Consistente e objetiva	Pouco consistente
Desenvolvimento							
Ênfase em ferramentas de automação/geração de código	Não	Não	Sim	Não	Não	Não	Não
Protótipos	Não	Sim	Sim	Não	Não	Não	Não
Reutilização de componentes	Não	Sim	Sim	Não	Não	Não	Não
Gestão de Equipa							
Tipo	Individual	Individual	Individual	Individual	*Pair-programming*	Individual	individual
Nível de competências técnicas	Elevado	Elevado	Baixa	Elevado	Elevado	Elevado	Elevado
Qualidade							
Testes	Após a implementação	Após a implementação	Contínuos	Contínuos	*Test-driven development*	Contínuos	Contínuos
Correções e tratamento de *bugs*	Etapa de manutenção	Etapa de manutenção	Contínuo	Contínuo	Contínuo	Contínuo	Contínuo
Gestão de Risco							
Exposição ao risco	Elevada	Baixa	Baixa	Baixa	Baixa	Baixa	Baixa
Tempo							
Duração de desenvolvimento	Elevada	Elevada	Baixa	Baixa	Baixa	Baixa	Baixa
Custo							
Elevado ou Baixo	Elevado	Elevado	Baixo	Baixo	Baixo	Médio	Baixo
Simplicidade							
Nível de simplicidade do modelo	Alto	Médio	Alto	Médio	Médio	Baixo	Médio

A interpretação desta comparação permite obter as seguintes conclusões para cada característica apresentada:

- **Gestão de Projeto:** o modelo em cascata e o modelo espiral apresentam uma relação mais formal com o cliente e este apenas está envolvido na fase inicial e na fase final do projeto. Nos restantes modelos verifica-se que a comunicação com o cliente é informal e há uma participação contínua dele durante o projeto;
- **Requisitos:** verifica-se que no modelo em cascata os requisitos de todo o projeto são especificados na etapa inicial, sendo que a entrega e a validação dos requisitos pelo cliente são realizadas na etapa final do ciclo de vida de desenvolvimento. Esta abordagem, juntamente com o escasso envolvimento do cliente, contribui para uma baixa resposta às necessidades do cliente e baixa flexibilidade. No modelo espiral os requisitos são definidos para uma iteração, não sendo considerado todo o contexto do projeto. Apesar de apresentar a mesma abordagem de validação e entrega ao cliente do modelo em cascata, o modelo em espiral apresenta melhor resposta às necessidades do cliente e resposta à mudança, uma vez que o seu processo é iterativo, não havendo períodos demasiado longos entre a especificação de requisitos e a sua entrega. Em RAD a especificação de requisitos realiza-se num workshop JDA, com contínua participação do cliente, o que favorece a entrega de soluções que dão resposta às necessidades do cliente, bem como a resposta à mudança. Á semelhança de RAD, com a exceção da não adoção de JDA, os restantes modelos apresentam boa capacidade de resposta ao cliente e elevada flexibilidade em responder às mudanças.
- **Documentação:** o modelo em cascata, modelo espiral e o RUP são os modelos que dão mais ênfase à documentação, resultando a elaboração de documentos consistentes e objetivos. Nos restantes modelos e metodologias a documentação é pouco ou quase nada considerada;
- **Desenvolvimento:** ao nível do desenvolvimento, mais concretamente ao nível do detalhe das ferramentas utilizadas, a metodologia RAD é a única que prevê a utilização de ferramentas que permitam automatizar o desenvolvimento. Para além disso, RAD juntamente com o modelo espiral consideram a utilização de protótipos e a reutilização de componentes no desenvolvimento;
- **Gestão de Equipa:** o XP adota a abordagem de *pair-programming*, contrariamente aos restantes modelos em análise que, tipicamente, responsabilizam individualmente os desenvolvedores pela implementação de tarefas. Em relação às competências técnicas

todos os modelos, à exceção de RAD, exigem níveis elevados de competências. Por outro lado, RAD exige um menor nível técnico por considerar que as ferramentas adotadas serão adequadas para qualquer desenvolvedor, independentemente do seu conhecimento técnico;

- **Qualidade:** o modelo em cascata prevê a implementação de testes na etapa imediatamente a seguir à implementação e o tratamento de possíveis falhas realiza-se na etapa de manutenção, correspondente à etapa final. Em XP segue-se a abordagem *test-driven development*, o que permite considerar que existe, neste modelo, foco na qualidade do código das soluções implementadas. Os restantes modelos e metodologias seguem uma abordagem de testes e correções contínuas durante o processo de desenvolvimento;
- **Gestão de risco:** o modelo em espiral é orientado ao risco, o que permite que o nível de exposição ao risco seja mais baixo. Para além disso, RAD, DSDM, XP, RUP e *Scrum* também apresentam baixo níveis de exposição ao risco, uma vez que se considera que a flexibilidade na resposta à mudança e o envolvimento contínuo do cliente permite reduzir os riscos nas tomadas de decisão, uma vez que, idealmente, estão em conformidade com o cliente. Por outro lado, o modelo em cascata apresenta uma elevada exposição ao risco por não existir possibilidade de mudança e adaptação do que é inicialmente previsto, aumentando o risco em não corresponder às necessidades reais do cliente.
- **Tempo:** no modelo em cascata e no modelo espiral a duração de desenvolvimento é elevada. Em teoria, na metodologia RAD, o tempo de desenvolvimento é mais baixo do que nos restantes modelos, uma vez que se espera que as ferramentas utilizadas permitam facilitar e aumentar a rapidez do desenvolvimento;
- **Custo:** o modelo em cascata e o modelo espiral apresentam custos variáveis e elevados devido aos atrasos durante o processo de desenvolvimento;
- **Simplicidade:** ao nível da simplicidade da sua compreensão o modelo em cascata, considerado preditivo, é orientado para o planeamento antecipado e segue um conjunto de etapas bem delineadas, o que contribui para tornar o modelo mais simples de entender. A esse nível, também RAD se apresenta como uma metodologia simples de entender e com etapas, procedimento e ferramentas bem definidas o que facilita a sua compreensão e adaptação. Os restantes modelos não são considerados simples,

por apresentarem abordagens, técnicas e procedimentos mais exigentes e difíceis de compreender e/ou adotar.

Esta comparação permite identificar as principais diferenças entre os modelos de desenvolvimento de software em estudo. Por sua vez, estas diferenças permitem que se questione quais são as alterações e os impactos resultantes de uma possível mudança de modelo ou metodologia de desenvolvimento ao nível de uma organização.

As transformações tecnológicas e as mudanças rápidas ao nível das necessidades dos clientes têm pressionado as organizações a adaptarem os seus processos de forma a conseguirem dar resposta aos seus clientes. Para tal, o desenvolvimento rápido de aplicações surge como uma abordagem a considerar neste novo panorama. Contudo, organizações que utilizam outros modelos e metodologias de desenvolvimento terão que adaptar os seus processos, de forma a conseguirem adotar os princípios e abordagens de RAD. Tal como mencionado na secção 2.3 Rapid Application Development o caminho para o desenvolvimento rápido seria mais adequado se as organizações seguirem, primeiramente, um desenvolvimento eficiente através de práticas eficazes e só posteriormente seguirem para o desenvolvimento rápido com a adoção de práticas eficazes e orientadas para a gestão de tempo. O autor mencionado na secção referida apresentava uma perspetiva das organizações nos anos 90, sendo que considerava que a maioria das organizações adotavam procedimentos que apresentavam como consequências elevados custos e planeamentos muito extensos. Contudo, o estudo da evolução dos modelos e metodologias permite verificar que, atualmente, existem novas abordagens que permitem que as organizações diminuam essas consequências.

3 Ferramentas de desenvolvimento rápido de soluções

Este capítulo apresenta uma descrição dos princípios de ferramentas *low-code*, da sua arquitetura, do processo de desenvolvimento adotado nestas ferramentas e das principais plataformas *low-code* existentes no mercado.

Ao longo das últimas décadas estudos realizados no âmbito de software têm vindo a focar-se na criação de abstrações que permitam maior foco na modelação de *software* e não em conceitos de memória ou dispositivos de rede (Schmidt, 2006). Esta abstração permite reduzir a complexidade dos processos de desenvolvimento (Schmidt, 2006).

As abstrações destinam-se tanto para linguagens de programação como ferramentas, sendo que a evolução de software permite verificar que as linguagens de programação evoluíram de forma a abstrair os programadores da complexidade de programação direta com o *hardware* (Schmidt, 2006).

Ao nível de abstração em ferramentas a evolução tem início nos anos 80 com a abordagem CASE definida para implementação de ferramentas e métodos que facilitem o desenvolvimento de *software* (Schmidt, 2006). As ferramentas CASE foram abordadas por (Martin, 1991) quando definiu a metodologia RAD. De facto, tal como se verificou na secção 2.3 Rapid Application Development as ferramentas são consideradas um elemento essencial no processo de desenvolvimento rápido de aplicações. No entanto, essas ferramentas têm evoluído considerando-se, atualmente, que as ferramentas que suportam o desenvolvimento rápido são designadas por ferramentas *low-code*.

3.1 Classificação de Ferramentas de Desenvolvimento

Segundo (Cuba Platform, 2018) a escolha entre as melhores ferramentas para desenvolvimento baseia-se nos seguintes critérios: aplicabilidade da ferramenta, velocidade de desenvolvimento e flexibilidade e capacidade de gestão. A aplicabilidade da ferramenta corresponde ao domínio de aplicação ou negócio e com o propósito e características técnicas.

Com base nestes critérios e em ferramentas de referência (Cuba Platform, 2018) apresenta na Figura 13 uma possível classificação para as mesmas.

Sendo a ferramenta *Cuba Platform* uma das ferramentas visadas nesta classificação, procedeu-se a uma análise crítica, de forma a verificar se a classificação seria pouco independente ou pouco relevante. Desta análise, como resultado considera-se que, no geral, esta classificação é adequada e imparcial, uma vez que apenas se limita a categorizar ferramentas.

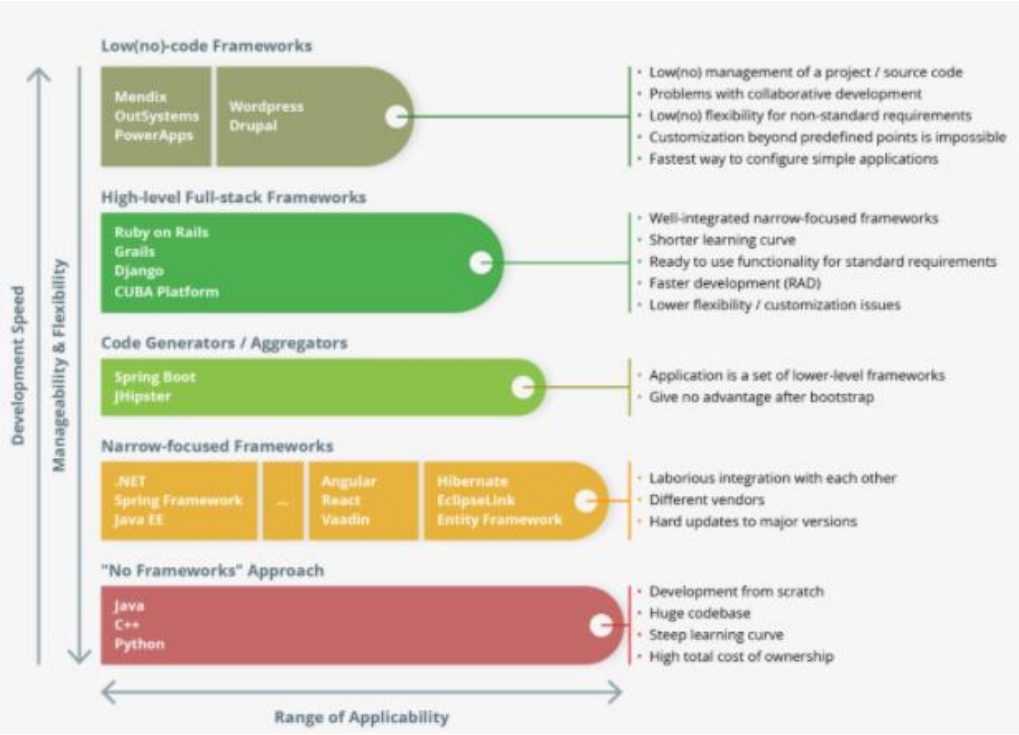


Figura 13 Classificação de ferramentas segundo (Cuba Platform, 2018)

Esta classificação divide as ferramentas em cinco categorias (Cuba Platform, 2018):

- ***"No framework" Approach:*** esta categoria contempla as ferramentas/linguagens que apresentam uma maior aplicabilidade e a mais elevada capacidade de gestão e flexibilidade, no entanto é a forma mais lenta de desenvolvimento. São exemplos da sua aplicabilidade soluções de baixo nível, tais como reconhecimento de imagem e algoritmos de processamento de dados.
- ***Narrow-focused frameworks:*** esta categoria contempla *frameworks* que permitem aumentar o nível de abstração no desenvolvimento. Estas apresentam uma velocidade de desenvolvimento e um nível de flexibilidade satisfatórios.
- ***Code generators/aggregators***: esta categoria contempla ferramentas que aceleram a fase inicial do processo de desenvolvimento, tendo como base as *narrow-focused frameworks*.

- ***High-level full-stack frameworks:*** as ferramentas nesta categoria apresentam um nível de abstração acima do nível de *narrow-focused frameworks*, dispondo de infraestrutura para implementar soluções mais rapidamente. São exemplos da sua aplicabilidade soluções ERP, BPMS e soluções personalizadas com requisitos específicos.
- ***Low/no-code frameworks:*** nesta categoria apresentam-se ferramentas que permitem um desenvolvimento de soluções mais rápido, desde que essas ferramentas suportem os requisitos necessários. A utilização destas ferramentas envolve um menor controlo sobre o código, sendo utilizadas para soluções como BPMS e aplicações de CRUD, por exemplo.

A classificação destas ferramentas permite verificar que a escolha de ferramentas ao nível de *low-code* resulta numa maior rapidez de desenvolvimento. Ao nível das organizações de TI, a escolha deste tipo de ferramenta estará, em parte, associada à procura da redução do tempo de desenvolvimento das suas aplicações.

3.2 Princípios das Ferramentas *Low-Code*

O termo *low-code* foi introduzido pela *Forrester Research* em 2014 que afirma que as organizações optam por alternativas de baixo código para aumentar a rapidez das entregas (Sanchis et al., 2020).

As plataformas de desenvolvimento *low-code* (LCDPs) são fornecidas na *cloud* através de um modelo *Platform-as-a-Service* (PaaS) que permitem o desenvolvimento e instalação de aplicações através de procedimentos que requerem pouco ou nenhum código (Sahay et al., 2020). (Sanchis et al., 2020) definem que as plataformas *low-code* são ecossistemas nos quais é possível desenvolver aplicações a partir de interfaces que permitem que pessoas sem *background* tecnológico criem aplicações.

De facto, o recurso a estas plataformas permite que utilizadores finais sem conhecimento prévio de programação, designados *citizen developers* no contexto das LCDPs, contribuam para o desenvolvimento de aplicações (Sahay et al., 2020). Para além disso, a pouca implementação de código permite que os desenvolvedores se concentrem na conceção das especificações das funcionalidades e na lógica de negócio, reduzindo o tempo de adaptação à sintaxe de uma

linguagem de programação e resolução de erros de código verificados no desenvolvimento tradicional (Waszkowski, 2019).

Estas plataformas fornecem um ambiente para os utilizadores criarem aplicações, através da interação com interfaces gráficas dinâmicas, diagramas e linguagens declarativas, invés do tradicional ambiente de desenvolvimento (Tisi et al., 2019) (Lefort and Costa, 2020) (Sahay et al., 2020). Na essência das LCDPs consideram-se os princípios de *model-driven engineering* (MDE) que têm sido adotados em vários contextos, baseando-se na automação, análise e abstração através de modelos e meta modelos (Sahay et al., 2020).

Model-Driven Engineering (MDE) é uma abordagem que determina que o principal foco do desenvolvimento de software são os modelos (Selic, 2003). Esta abordagem é um subconjunto de MBE, abordagem onde os modelos não conduzem o processo, e um superconjunto de MDD (ver Figura 14), que corresponde a um paradigma de desenvolvimento que recorre a modelos como elemento principal no processo de desenvolvimento (Brambilla et al., 2012). O MDA corresponde a um subconjunto de MDD, apresentando uma visão particular desta abordagem proposta por *Object Management Group* (OMG) (Brambilla et al., 2012).

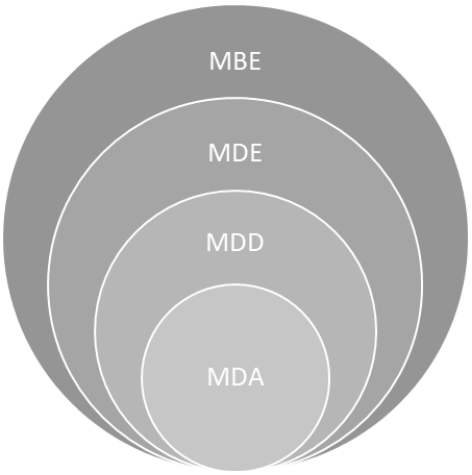


Figura 14 Esquema de conjuntos *model-driven* baseado em (Brambilla et al., 2012)

Nesta abordagem as soluções e outros artefactos são automaticamente gerados a partir dos seus modelos (Neis et al., 2019). Um modelo é uma abstração da realidade que tem a capacidade de prever o comportamento de um processo, estando presente, nesta abordagem, durante todo o ciclo de vida de desenvolvimento (Neis et al., 2019).

MDE aborda os seguintes conceitos:

- ***Domain-specific modeling languages (DSMLs, DSLs):*** correspondem a linguagens definidas especificamente para um determinado domínio ou contexto (Brambilla et al., 2012). Estas linguagens utilizam metamodelos que definem a estrutura, a semântica, as relações entre os conceitos de um domínio e as suas restrições (Schmidt, 2006) (Boussaïd et al., 2017). Os metamodelos representam mais uma camada de abstração (Brambilla et al., 2012) que, por sua vez, são construídos a partir de meta-metamodelos (ver Figura 15) (Boussaïd et al., 2017). Para a definição de metamodelos recorre-se à linguagem padronizada *Meta Object Facility* (MOF) que define linguagens de modelação baseadas nos conceitos dos diagramas de classe UML (Unified Modeling Language) (Boussaïd et al., 2017).
- ***Transformation engines and generators:*** estas ferramentas permitem analisar e sintetizar artefactos como código e representações de modelos alternativos (Schmidt, 2006). O processo de transformação baseia-se em "*correct-by-construction*", contrariamente aos processos de desenvolvimento tradicionais que se baseiam em "*construction-by-correction*" (Schmidt, 2006). A transformação de modelos utiliza-se para traduzir *source models* para um ou mais *target models*, tendo em conta um conjunto de regras (ver Figura 16) (Boussaïd et al., 2017). As transformações podem ser *Model-To-Model*, se os artefactos forem modelos ou *Model-to-text* no caso de transformação de modelos para texto como código-fonte, por exemplo (Boussaïd et al., 2017).

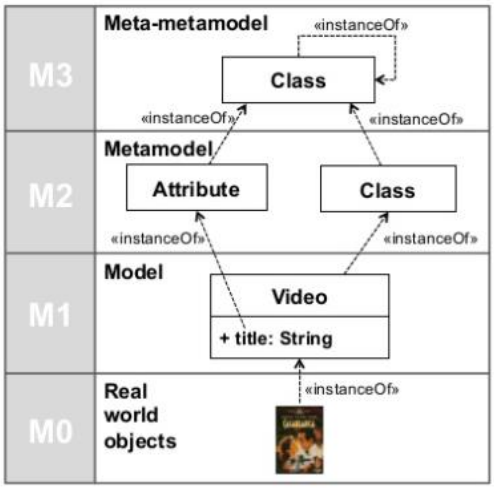


Figura 15 Camadas de abstração de modelos e metamodelos (Brambilla et al., 2012)

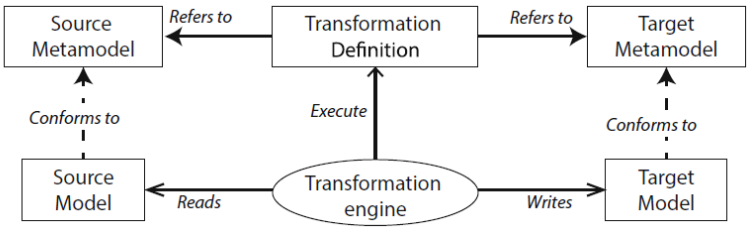


Figura 16 Conceitos de transformação (Boussaïd et al., 2017)

3.3 Adoção de plataformas *low-code*

Em 2019, com o propósito de compreender o estado do desenvolvimento de aplicações, a *Outsystems*, uma das líderes no mercado low-code, realizou um inquérito a mais de 3300 profissionais de TI de diferentes continentes, tendo questionado os inquiridos sobre as principais razões para adotarem ou não plataformas de desenvolvimento *low-code* (Outsystems, 2019a).

De acordo com esse inquérito (Outsystems, 2019a), ao nível das principais razões para adoção deste tipo de plataforma obteve-se os seguintes resultados (cf. Figura 17): 66% aceleração da inovação/transformação digital e aumentar a capacidade de resposta ao negócio; 45% reduzir a dependência de competências técnicas difíceis de contratar; 28% escapar de *legacy debt*; 22% proteção contra agitação tecnológica; 20% permitir que os denominados *citizen developers* melhorem os processos internos; 2% outra razão.

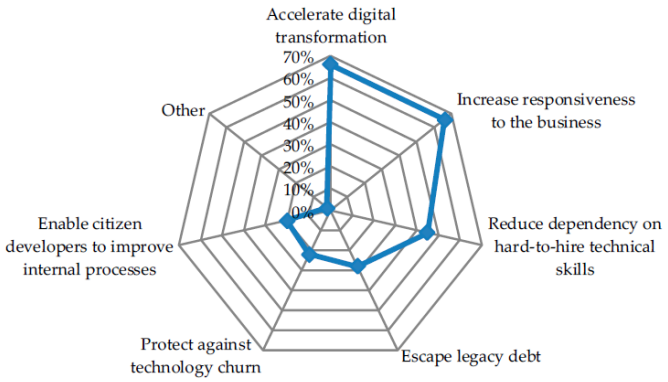


Figura 17 Principais razões para adoção de plataformas *low-code* (Outsystems, 2019a) (Sanchis et al., 2020)

Por outro lado, ao nível das principais razões para a não adoção de plataformas de desenvolvimento *low-code* obteve-se os seguintes resultados (cf. Figura 18): 47% falta de conhecimento das plataformas *low-code*; 37% preocupação com *lock-in* com o fornecedor da plataforma *low-code*; 32% não acreditam que consigam construir o tipo de aplicações que necessitam; 28% preocupação acerca da escalabilidade das aplicações criadas; 25% preocupação com a segurança das aplicações criadas; 10% outra razão.

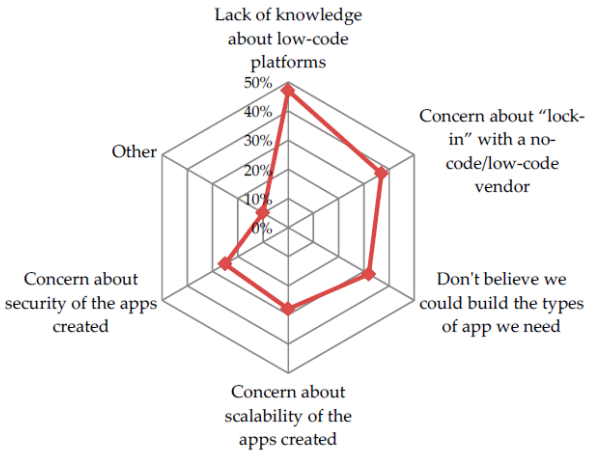


Figura 18 Principais razões para a não adoção de plataformas *low-code* (Outsystems, 2019a) (Sanchis et al., 2020)

Estas razões de adoção e não adoção são também considerados pela Forrester em (Rymer, 2017).

3.4 Processo de desenvolvimento em plataformas low-code

De acordo com (Sahay et al., 2020) (Alexander, 2021), durante o processo de desenvolvimento através de plataformas *low-code* ocorrem, tipicamente, as seguinte fases:

- **Definição dos requisitos:** fase inicial do processo de desenvolvimento que tem como objetivo o levantamento e definição dos requisitos da solução;
- **Definição de especificações da aplicação:** nesta fase a solução é implementada através de:
    - **Modelação de dados:** através das interfaces gráficas define-se, geralmente através de opções de *drag-and-drop*, o esquema de dados da aplicação com a criação de entidades, relações e dependências;
    - **Definição de interfaces gráficas:** define-se os formulários, páginas e gerem-se os papéis dos utilizadores ao nível dos formulários, páginas, entidades e componentes;
    - **Especificação de regras de negócio e *workflows*:** definem-se as regras de negócio através de *workflows*, tais como *visual-based workflow* ou com notação BPMN, entre os formulários ou páginas.
- **Integração com serviços externos através de *third-party* APIs:** nesta fase realizam-se as comunicações aos serviços externos, se necessário, sendo que estas plataformas fornecem formas de estabelecer estas comunicações;
- **Customização da aplicação:** nesta fase realizam-se personalizações na aplicação, se necessário, através da implementação de código no *front-end* ou com a customização de *queries*, por exemplo;
- **Testes de aceitação:** nesta fase realizam-se os testes de aceitação do utilizador, de forma a validar os requisitos implementados;
- ***Deployment* da aplicação:** nesta fase realiza-se o *deploy* da aplicação que, na maioria das plataformas, consegue-se através de poucos cliques.

3.5 Arquitetura

As plataformas de desenvolvimento de desenvolvimento rápido de aplicações apresentam, tipicamente, uma arquitetura em camadas (Sahay et al., 2020):

- **Apresentação:** esta camada apresenta o ambiente gráfico que permite que os utilizadores construam as interfaces gráficas, bem como especifiquem o comportamento da aplicação através da construção de modelos;
- **Integração de Serviços:** corresponde à camada de comunicação a serviços através de APIs e mecanismos de autenticação;
- **Integração de dados:** camada que permite a manipulação de dados, bem como a sua integração a diferentes fontes;
- ***Deployment:*** nesta camada efetua-se o *deploy* da aplicação na *cloud* ou em ambiente *on-premise*. Para além disso, nesta camada são tratadas a contentorização e orquestração das aplicações, em conjunto com integração contínua e *deployment* que colaboram com a camada de integração de serviços.

Tendo como base esta arquitetura (Sahay et al., 2020) detalharam os principais componentes destas plataformas e dividiram-nos em três níveis, tal como se pode observar Figura 19.

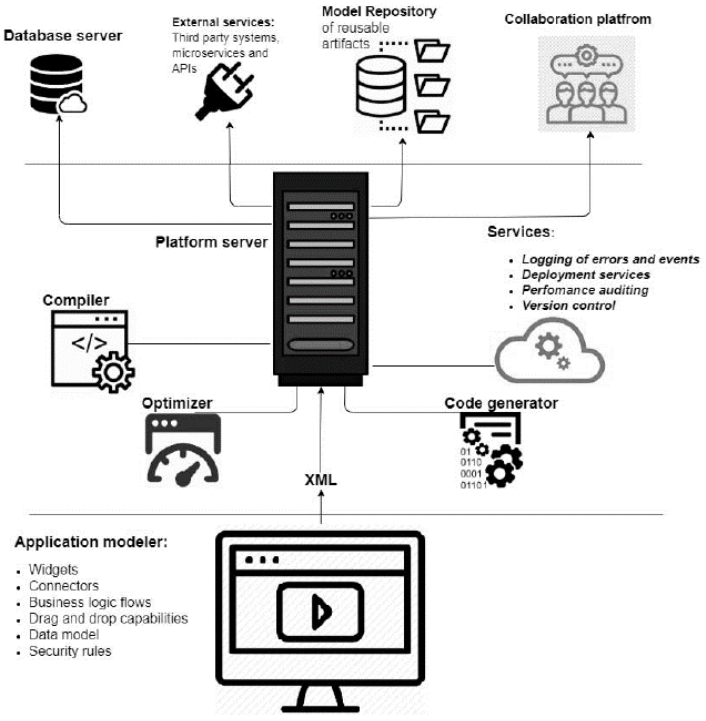


Figura 19 Componentes das LCDPs

No nível da modelação da aplicação, tal como referido anteriormente para a camada de apresentação, os desenvolvedores especificam e constroem as suas aplicações através dos modelos e das abstrações fornecidas, podendo correr essas aplicações localmente, antes da instalação (Sahay et al., 2020). O nível seguinte trata o modelo recebido do nível anterior, prosseguindo com geração de código, otimizações e considerando os serviços envolvidos, tais como sistema de base de dados, conetores de APIs, micro serviços e modelos de repositórios reutilizáveis (Sahay et al., 2020). Por sua vez, estes serviços apresentam-se no último nível. Ao nível de servidor de base de dados, os utilizadores e programadores não têm controlo sobre ele, uma vez que não se espera que haja preocupação com o tipo de base de dados ou com a integridade dos dados (Sahay et al., 2020). Também os micro serviços são criados e orquestrados no *back-end*, não havendo intervenção dos desenvolvedores da aplicação (Sahay et al., 2020).

Para permitir a reutilização de artefactos estas plataformas fornecem repositórios para armazenamento destes artefactos (Sahay et al., 2020). Para além disso, estas ferramentas facilitam a integração de metodologias de desenvolvimento, de forma a que as equipas possam visualizar e acompanhar o processo de desenvolvimento da solução (Sahay et al., 2020).

Como exemplo, a figura seguinte apresenta a arquitetura de *Outsystems*, uma das plataformas líderes no mercado (Outsystems, 2021a):

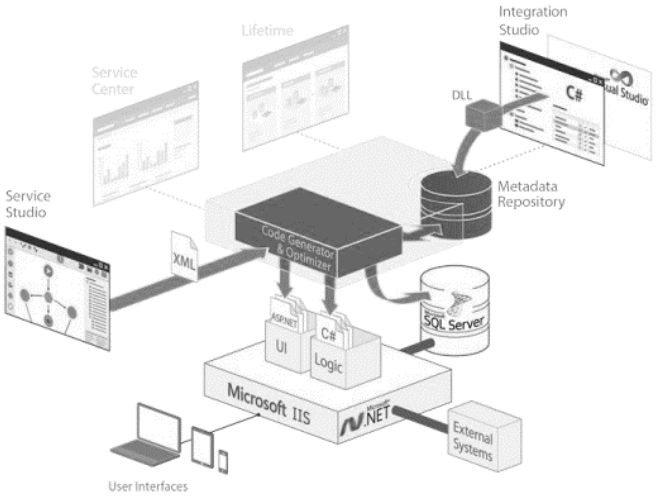


Figura 20 Arquitetura de Outsystems (Outsystems, 2021a)

Nesta arquitetura é possível ver os componentes nos três níveis definidos anteriormente: (i) no primeiro nível apresenta-se o *Service Studio*, onde é realizada a modelação de dados e construção da aplicação; (ii) no segundo nível apresenta-se o *code generator and optimizer* que, recebendo o modelo definido no nível anterior, gera todos os componentes; (iii) no último nível é feita a gestão de integração de serviços através da ferramenta *Integration Studio* (Outsystems, 2021a). A maioria das plataformas *low-code* incorporam e automatizam as operações *DevOps*, que permite simplificar o ciclo de vida de desenvolvimento das aplicações (Vincent, 2019).

Segundo (Gartner, 2021) *DevOps* "representa uma mudança na cultura de TI, focando-se na rápida prestação de serviços de TI através da adoção de práticas ágeis no contexto de uma abordagem orientada para o sistema. *DevOps* enfatiza as pessoas (e a cultura) e procura melhorar a colaboração entre as operações e as equipas de desenvolvimento. As implementações *DevOps* utilizam tecnologia – especialmente ferramentas de automação que podem alavancar uma infraestrutura cada vez mais programável e dinâmica, numa perspetiva de ciclo de vida"[8]

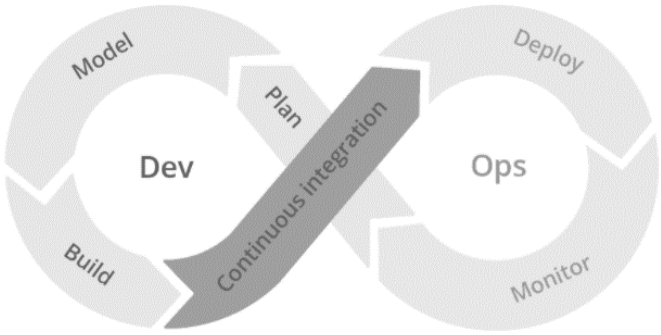


Figura 21 *DevOps* em *Mendix* (Mendix, 2021a)

No contexto das plataformas *low-code*, a plataforma *Mendix*, por exemplo, apresenta uma base na sua plataforma para que as equipas de desenvolvimento adotem as práticas *DevOps* (Mendix, 2021a).Para além disso, *Mendix* fornece ferramentas *DevOps out-of-the-box* para integração e

[8] Tradução livre da autora. No original "DevOps represents a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach. DevOps emphasizes people (and culture), and it seeks to improve collaboration between operations and development teams. DevOps implementations utilize technology — especially automation tools that can leverage an increasingly programmable and dynamic infrastructure from a life cycle perspective"

entrega contínua (CI/CD), automatização de testes e monitorização e permite a utilização de APIs da plataforma para integração com *pipelines* CI/CD existentes (Mendix, 2021a).

Desta forma, é possível criar, testar e manter aplicações em ambientes de produção de forma autónoma, assegurando o cumprimento dos requisitos (ver Figura 21) (Mendix, 2021a).

3.6 Principais plataformas low-code

As plataformas *low-code* são desenvolvidas, geridas e comercializadas por organizações que se responsabilizam pela definição de estratégias adequadas para que a plataforma tenha impacto no mercado.

Anualmente, a Gartner (Vincent et al., 2020) e a Forrester Research (Rymer and Koplowitz, 2019) publicam relatórios de análise do mercado das plataformas *low-code*. Estes relatórios avaliam as organizações que detêm as plataformas segundo critérios de negócio e estratégia definidos. Esta avaliação baseia-se, essencialmente, no produto, nas estratégias de venda e marketing, nos modelos de negócio e na compreensão do mercado. Desta avaliação resultam os pontos fortes e pontos fracos de cada organização, bem como um gráfico que permite apurar as organizações líderes, visionárias, desafiadoras e as mais contidas no mercado.

A Figura 22 e Figura 23 apresentam os resultados dos relatórios da Gartner e Forrester Research, respetivamente. Tal como se pode observar (Vincent et al., 2020) identifica os seguintes líderes: *Microsoft*, *Salesforce*, *Outsystems*, *Mendix*, *Appian* e *Service Now*. Por outro lado, (Rymer and Koplowitz, 2019) identifica *Microsoft*, *Salesforce*, *Outsystems*, *Mendix* e *Kony* como líderes.

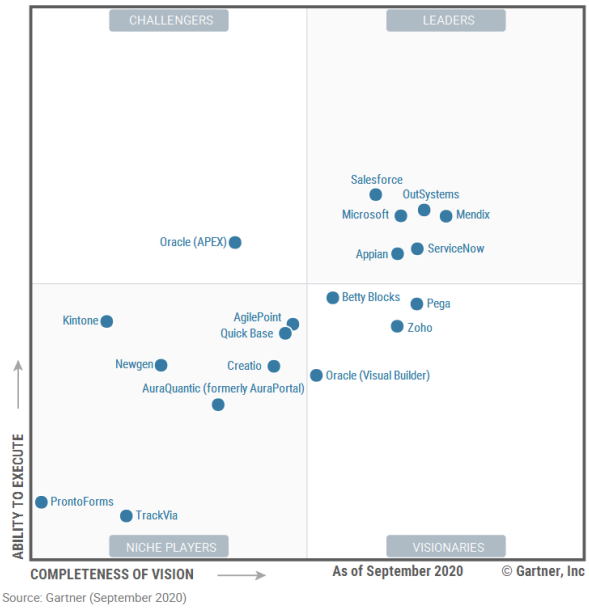


Figura 22 Quadrante para plataformas *low-code* (Vincent et al., 2020)

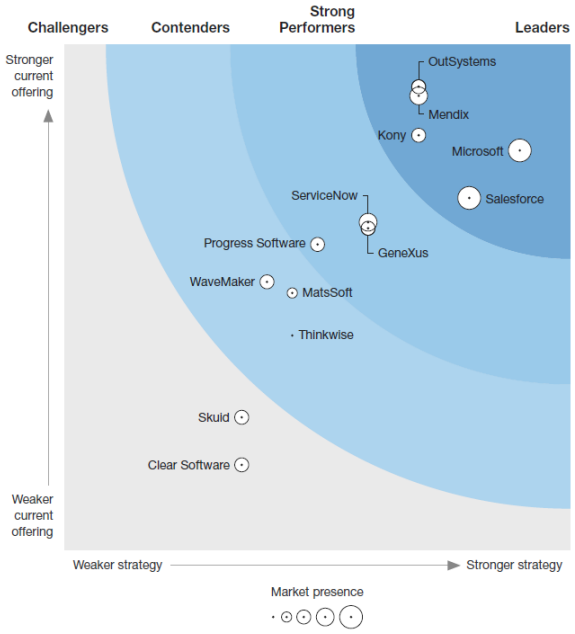


Figura 23 Principais fornecedores de plataformas *low-code* (Rymer and Koplowitz, 2019)

De seguida, apresentam-se as plataformas identificadas como líderes em ambos os relatórios analisados – *Microsoft* (com a plataforma PowerApps), *Salesforce*, *Outsystems* e *Mendix*.

3.6.1 Microsoft Power Apps

A plataforma *Power Apps*, que funciona em conjunto com a *Power Automate* (lógica de negócio e integrações) e com *Common Service Data*, foi desenvolvida e é gerida pela *Microsoft* (Microsoft, 2021). Esta plataforma, em conjunto com *Power BI* (análise de negócios) e *Power Virtual Agents* (*chatbots* de baixo código), fazem parte da plataforma *Power* (Microsoft, 2021).

Power Apps é "um conjunto de aplicações, serviços, conectores e plataformas de dados que fornece um ambiente de programação rápida para que possa criar aplicações personalizadas para as suas necessidades empresariais" (Microsoft, 2021). Para além disso, com esta plataforma é possível, nas aplicações criadas, incorporar dados armazenados na plataforma *Microsoft Dataverse*, *Sharepoint*, *Microsoft 365*, *Dynamics 365* e *SQL Server*, por exemplo (Microsoft, 2021).

Esta plataforma permite a criação de três tipos aplicações: aplicações de tela, *model-driven* e portal (Microsoft, 2021). As aplicações de tela baseiam-se no *drag-and-drop* de elementos numa tela, permitindo, também, a criação de uma aplicação a partir de fontes de dados, como *Dataverse*, *SharePoint*, *Excel*, entre outros (Microsoft, 2021). As aplicações baseadas em *model-driven* permitem a criação de aplicações orientadas a modelos composta por componente, que consiste, essencialmente, na modelação de dados de negócio, definição de processos de negócio e composição da aplicação (Microsoft, 2021). Os portais permitem uma experiência mais avançada, permitindo que sites com acesso externo possibilitem que os utilizadores externos iniciem sessão com uma diversidade de identificação, bem como tenham acesso à gestão de dados no *Microsoft Dataverse*, por exemplo (Microsoft, 2021).

Ao nível de APIs e serviços de integração a *Power Apps* e *Common Data Service* apresentam um conjunto de APIs e *endpoints OData* disponíveis para gestão de dados, permitindo a integração com outros serviços externos (Vincent et al., 2020). Ao nível de inteligência artificial, esta plataforma apresenta extensões como *AI builder* que permitem o consumo de modelos de IA para extrair texto de imagens, identificar linguagens ou analisar sentimentos, por exemplo (Vincent et al., 2020) (Microsoft, 2021). Para além disso, o *Power Virtual Agents*, integrado em *Power Apps,* permite a criação de *chatbots* utilizando uma interface sem código (Microsoft, 2021).

Em (Vincent et al., 2020) são identificadas algumas limitações desta plataforma e da sua utilização. A primeira limitação está relacionada com *workflows* que, apesar de suportados em

PowerApps, a interface para fluxos de negócio é separada e está apenas disponível nas aplicações orientadas a modelos (Vincent et al., 2020). Para além disso, a notação BPMN em *Power Automate* é apenas suportada através de uma integração com o *Microsoft Visio* (Vincent et al., 2020).

Outra limitação desta plataforma está associada à prática de preços, sendo que para uma funcionalidade como *AI builder* os clientes têm de realizar um pagamento adicional (Vincent et al., 2020). Perante estes custos adicionais durante a utilização da plataforma, os clientes têm de estar conscientes das alterações de licenças e, consequentemente, dos preços (Vincent et al., 2020).

3.6.2 Salesforce Cloud

A plataforma *Salesforce Cloud*, desenvolvida e gerida pela *Salesforce*, permite a construção e publicação de aplicações, seguras e escaláveis, baseadas na *cloud* (Sahay et al., 2020). Esta plataforma apresenta ferramentas *out-of-the-box* e permite a integração com serviços externos (Sahay et al., 2020).

Uma das funcionalidades presentes nesta ferramenta é o *Lightning Flow* que permite a construção, gestão e execução de fluxos e processos, através de dois construtores *point-and-click* – *Flow Builder* e *Process Builder* (Salesforce, 2021). O *Flow Builder* destina-se à criação de processos com lógica complexa, enquanto o *Process Builder* destina-se a automação de ações, baseada em critérios (Salesforce, 2021). Para além disso, o *Lightning Flow* permite a comunicação dentro e fora de *Salesforce*, através da subscrição e publicação de eventos na plataforma, com recurso a *MuleSoft APIs* e classes *Apex* (Salesforce, 2021).

Uma das particularidades de *Salesforce Cloud* é a *AppExchange* que contém uma variada oferta de aplicações com finalidade de gerir parcerias ou integrar e-mail, por exemplo. Para além disso, são disponibilizados componentes gratuitos prontos a utilizar e personalizar (Salesforce, 2021). Esta oferta de aplicações e componentes permite facilitar a criação de soluções.

(Vincent et al., 2020) descreve que uma das limitações desta plataforma está relacionada com a sua arquitetura, uma vez que esta difere das mais atuais infraestruturas *distributed-cloud-native*. Para além disso, a falta de suporte *low-code* para integração e gestão de API também representa uma das principais limitações desta plataforma (Vincent et al., 2020).

3.6.3 Outsystems

A Outsystems é uma plataforma que permite o desenvolvimento visual e orientado a modelos de aplicações que podem ser executadas na *cloud* ou em infraestruturas locais (Sahay et al., 2020) (Outsystems, 2021a). Para além disso, permite a integração com IA e *DevOps*, bem como a extensão de componentes desenvolvidos em código tradicional (Outsystems, 2021a).

Esta plataforma apresenta a *framework Ousystems UI* que permite o desenvolvimento rápido de interfaces gráficas de uma aplicação *web* ou *mobile* a partir de um *template* existente ou a partir de uma página vazia (Outsystems, 2021a). Estas interfaces são criadas a partir de *drag-and-drop* de componentes para os ecrãs (Outsystems, 2021a). Em *Outsystems* o desenvolvimento de aplicações realiza-se no *Service Studio* - ambiente de desenvolvimento visual e de baixo código da plataforma (Outsystems, 2021a). Contudo, é possível criar protótipos de interfaces gráficas ou criar *workflows* através de *Experience Builder* ou *Workflow Builder*, respetivamente (Oustystems, 2018).

Para integração com serviços externos é utilizada a ferramenta *Integration Studio* que permite a gestão de *extensions* que correspondem ao conjunto de estruturas, entidades e ações (Outsystems, 2021a).

Para além disso, a *Outsystems* dá suporte ao ciclo de vida de desenvolvimento de soluções, através de (Oustystems, 2018): (i) *deployment* de todos os componentes da solução e componentes para integração através de um clique; (ii) monitorização automática de aplicações com apresentação de *dashboards* que possibilitam o levantamento de questões crítica ou problemas através da sua análise; (iii) gestão de configurações de aplicações em execução, simplificando o *deployment* de aplicações em produção e garantindo uma entrega contínua.

Esta plataforma tem sido cada vez mais usada para automação de processos de negócio, contudo segundo (Vincent et al., 2020) a plataforma apresenta alguns desafios na modelação de decisões mais complexas, o que permite aferir que ainda não é competitiva neste contexto. Outro aspeto desafiante da plataforma está relacionado à estratégia, uma vez que (Vincent et al., 2020) referem que Outsystems favorece os programadores, invés dos *citizen developers*, sendo que, as plataformas concorrentes tendem a estimular os dois perfis.

3.6.4 Mendix

Mendix é uma plataforma de *no-code* e *low-code* orientada a modelos e baseada numa arquitetura *cloud-native* (Vincent et al., 2020). Esta plataforma suporta o desenvolvimento através de integração, *workflows*, processamento de eventos e utilização de IA (Vincent et al., 2020). Adicionalmente, esta plataforma suporta *chatbots* e casos de uso de aplicações de *Internet of Things* (IoT). Para além disso, esta plataforma incorpora o *Mendix Assist* que fornece sugestões em tempo real para as próximas etapas de implementação (Mendix, 2021a).

Ao nível de ambiente de desenvolvimento esta plataforma apresenta dois ambientes – *Mendix Studio* e *Mendix Studio Pro* – um para *citizen developers* e outros para programadores profissionais, respetivamente (Vincent et al., 2020). O *Mendix Studio* é um ambiente visual, sem código e que permite, por exemplo, o desenvolvimento de interfaces gráficas, modelação de dados e modelação do fluxo da aplicação (Mendix, 2021a). O *Mendix Assist* disponibiliza recomendações e sugestões de forma a garantir qualidade e acelerar o processo de aprendizagem (Mendix, 2021a).

Por outro lado, o *Mendix Studio Pro é* um ambiente visual *low-code*, orientado por modelos que disponibiliza aos programadores profissionais as ferramentas para a criação de aplicações, possibilitando, também, a personalização da aplicação através da implementação de código em editores incorporados (Mendix, 2021a). Neste ambiente, o *Mendix Assist* disponibiliza recomendações e atua como um par, sendo um programador virtual, com o programador (Mendix, 2021a).

Mais recentemente, esta plataforma lançou o *Data Hub Catalog* que é um repositório de meta data que permite que todos os desenvolvedores gerir dados dentro do ecossistema de dados conectados (Mendix, 2021a).

Esta plataforma apresenta algumas limitações, principalmente ao nível da estratégia e política de preços. (Vincent et al., 2020) consideram que esta plataforma se atrasou na oferta de soluções *industry-specific* e, ao nível de preços e flexibilidade de contratos as pontuações são baixas, comparativamente aos seus concorrentes.

3.6.5 Comparação das funcionalidades das plataformas

A Tabela 2 apresenta a comparação de funcionalidades entre as plataformas descritas previamente. Esta comparação é apresentada em (Sahay et al., 2020), sendo que os autores definiram uma taxonomia para facilitar a categorização de funcionalidades. A tabela que descreve esta taxonomia encontra-se no Anexo 1 – Taxonomia para plataformas *low-code*.

A partir desta tabela é possível concluir que em algumas categorias de funcionalidades as quatros principais ferramentas no mercado *low-code* apresentam, de uma forma geral, uma oferta idêntica. Ao nível de suporte de comunicação com serviços externos, segurança e escalabilidade as quatro plataformas apresentam as mesmas funcionalidades. Sendo assim, em todas elas existe: apoio à integração de serviços externos e fontes de dados; apoio à segurança através da confidencialidade, integridade e autenticação e escalabilidade em termos de número de utilizadores, tráfego de dados e armazenamento de dados.

Tabela 2 Comparação de funcionalidades entre plataformas (Sahay et al., 2020)

Funcionalidade	**Microsoft Power Apps**	**Salesforce Cloud**	**Outsystems**	**Mendix**
Graphical user interface				
Drag-and-drop designer		✓	✓	✓
Point and click approach	✓			
Pre-built forms/reports	✓	✓	✓	✓
Pre-built dashboards	✓	✓	✓	
Forms	✓			
Progress tracking	✓	✓	✓	✓
Advanced reporting				
Built-in workflows		✓		
Configurable workflows		✓		
Interoperability support				
Interoperability with external service	✓	✓	✓	✓
Connection with data sources	✓	✓	✓	✓
Security Support				
Application security	✓	✓	✓	✓

Funcionalidade	Microsoft Power Apps	Salesforce Cloud	Outsystems	Mendix
Platform security	✓	✓	✓	✓
Collaborative development support				
Off-line collaboration	✓	✓	✓	✓
On-line collaboration		✓	✓	✓
Reusability support				
Built-in workflows		✓		
Pre-built forms/reports	✓	✓	✓	✓
Pre-built dashboards	✓	✓	✓	
Scalability				
Scalability on number of users	✓	✓	✓	✓
Scalability on data traffic	✓	✓	✓	✓
Scalability on data storage	✓	✓	✓	✓
Business logic specification mechanisms				
Business rules engine	✓	✓	✓	✓
Graphical workflow editor		✓	✓	✓
AI enabled business logic		✓	✓	
Application build mechanisms				
Code generation			✓	
Models at run-time	✓	✓		✓
Deployment support				
Deployment on cloud	✓	✓	✓	✓
Deployment on local infrastructures		✓	✓	✓
Kinds of supported applications				
Event monitoring	✓	✓	✓	✓
Process automation	✓		✓	
Approval process control				
Escalation management				
Inventory management	✓	✓	✓	✓
Quality management	✓	✓		✓
Workflow management	✓	✓	✓	✓

Por outro lado, esta comparação permite verificar algumas diferenças entre as funcionalidades, tais como:

- *Power Apps* não suporta as seguintes funcionalidades: design *drag-and-drop*; colaboração online de forma que diferentes pessoas possam colaborar em simultâneo na mesma aplicação; geração de código; lógica de negócio ativada por IA; *deployment* em infraestruturas locais; Por outro lado, é a única plataforma a permitir a criação de formulários e a seguir a abordagem *point and click* em que é necessário apontar sobre o item e clicar na interface, invés do *drag-and-drop* do item (Sahay et al., 2020);
- *Salesforce Cloud* é a única plataforma em análise que suporta *workflows* incorporados e configuráveis;
- *Outsystems* apresenta o mecanismo de geração de código, sendo que, ao contrário das restantes plataformas, o código fonte da aplicação é gerado e posteriormente implantado antes da sua execução (Sahay et al., 2020). Nas restantes plataformas o modelo da aplicação é implementado e utilizado em *run-time* durante a execução da aplicação, sem geração de código;
- *Mendix* não suporta mecanismos de lógica de negócio suportada por IA para aprendizagem do comportamento de um atributo, de forma a reproduzir esses comportamentos num mecanismo de aprendizagem (Sahay et al., 2020);

A comparação de funcionalidades permite analisar as plataformas em relação ao que estas oferecem os seus utilizadores. Contudo, a escolha de uma ferramenta *low-code* ao nível de uma organização é também influenciada por outros fatores, tais comos, dimensão da organização, tipo de soluções que desenvolvem e custos da plataforma (Sahay et al., 2020). Assim, estes fatores requerem atenção para a decisão de adoção ou não deste tipo de plataforma, sendo que, o fator tipo de solução contribui para a complexidade dessa decisão, uma vez que têm de ser considerados todos os aspetos do processo de desenvolvimento, bem como a metodologia de desenvolvimento seguida pela organização.

4 Casos de estudo

Este capítulo apresenta os casos de estudo utilizados no âmbito deste projeto, bem como a justificação dos aspetos considerados essenciais na sua definição.

No âmbito deste projeto de dissertação, o desenvolvimento de soluções através de plataformas *low-code* assentou na definição de dois casos de estudo que simulam domínios de um contexto profissional. De forma a simplificar o entendimento do contexto dos casos de estudo, estes foram definidos tendo em conta o mesmo domínio. Assim, optou-se por um domínio que, na sua generalidade é familiar para um vasto conjunto de pessoas, permitindo facilitar a sua compreensão. Para além disso, considerou-se o desenvolvimento de uma solução para gestão de atividades – *Business to Business* (B2B) e uma solução para suportar as necessidades de um cliente final – *Business to Consumer* (B2C) (Sahay et al., 2020). A consideração destes tipos de soluções pretende apresentar um domínio aplicacional abrangente, permitindo simplificar o entendimento do problema.

Neste contexto, cada caso de estudo, apresentado em cada subcapítulo, compreende: (i) uma descrição breve do domínio de negócio; (ii) especificação de requisitos de negócio considerados relevantes e (iii) o desenho da arquitetura da solução para dar resposta ao problema e aos respetivos requisitos. Relativamente à definição destas arquiteturas, em cada caso de estudo definiram-se diferentes arquiteturas das soluções, de forma a ser possível dar resposta à questão de investigação Q1 (cf. 1.3 Objetivos). A construção das soluções, em conformidade com o desenho definido, será realizada recorrendo às plataformas *low-code* em análise – Outsystems e Mendix.

A definição destes casos de estudo foi suportada tendo em conta os principais desafios no desenvolvimento de aplicações, que atrasam a sua entrega, independentemente das tecnologias escolhidas (Rymer, 2017) (Outsystems, 2019a): (i) dificuldade em responder aos requisitos de negócio a tempo; (ii) falta de flexibilidade; (iii) dificuldade de redução de custos; (iv) aumento de *cyber attacks* o que resulta na necessidade de garantir a segurança das aplicações; (v) dificuldade de desenvolvimento UX/UI adequado e (vi) dificuldade de coordenação para *deployment*. Para além disso, para esta definição e de forma a ser possível avaliar de forma crítica as características de cada uma das plataformas, os casos de estudo não tiveram em consideração as principais características dessas plataformas.

Os subcapítulos que se seguem apresentam a estrutura descrita anteriormente. Ao nível da descrição do desenho das arquiteturas das soluções recorreu-se à representação das vistas consideradas pelo *View Model* 4+1 (Kruchten, 1995), mais precisamente à vista lógica, de processo e de implantação. Adicionalmente, combinou-se o modelo C4 (C4 Model, 2021) com a vista lógica, de forma a representar a arquitetura em dois níveis: (i) nível 2: considera a representação de *containers* (e.g. aplicações e bases de dados) e (ii) nível 3: considera a representação dos componentes que compõe um *container*.

4.1 Caso de estudo 1

Este subcapítulo apresenta a descrição do domínio de negócio deste caso de estudo e a descrição dos principais aspetos a considerar para a sua implementação.

No contexto deste projeto, para além dos objetivos associados à definição da arquitetura, pretende-se que com este caso de estudo seja possível construir e analisar soluções, cujo seu desenvolvimento deve, essencialmente, focar-se nos detalhes de modelação de dados e base de dados, lógica de negócio e interfaces do sistema. Assim, será possível analisar e comparar, em detalhe, a capacidade de resposta de cada plataforma em estudo face a esses detalhes.

4.1.1 Descrição de negócio

A empresa fictícia Alfa Lda. dedica-se à produção de roupa hospitalar sendo que, nos últimos meses, teve um aumento da procura dos seus produtos, em Portugal e no resto da Europa. Este aumento da procura levou a empresa a procurar formas mais eficientes e rápidas de gerir o seu inventário - melhorar a gestão do seu stock e a sua produção, bem como melhorar a gestão de encomendas a fornecedores e clientes. Desta forma, a empresa Alfa Lda. pretende uma solução para gestão das suas encomendas, considerando o seu processo de produção atual.

A empresa Alfa Lda. apresenta uma lista de clientes, sendo que estes podem realizar encomendas de produtos. Nestas encomendas é necessário validar o stock de matérias-primas para a produção dos itens encomendados. Esta validação de stock ocorre atualmente num sistema externo[9], sendo necessário considerá-lo na integração no processo descrito. Em caso

[9] Por se tratar de um caso fictício, este sistema externo corresponde a um serviço *mock*, utilizado para simular esse sistema.

de não existirem matérias-primas disponíveis, os colaboradores do departamento de vendas realizam um pedido de encomenda de matérias-primas (tecido e outros materiais) aos fornecedores. Assim que estas encomendas são entregues no ponto de produção regista-se a sua entrega e o novo material é considerado na lista de stock de matérias-primas disponíveis. Segue-se o processo de produção das encomendas para os clientes, sendo este processo da responsabilidade dos colaboradores do departamento de produção. O acompanhamento deste processo desde o início é essencial para evitar a produção duplicada e o desperdício associado. Assim que a encomenda estiver finalizada e embalada regista-se o seu estado – pronto para entrega – sendo que a equipa de distribuição será responsável pela sua entrega aos clientes. O pagamento destas encomendas é feito à cobrança, pelo que, no processo atual não se pretende considerar outros formas de pagamento.

Para uma melhor compreensão deste processo, dos seus conceitos e da relação entre eles apresenta-se, na Figura 24, um modelo de domínio que ilustra os principais conceitos relevantes para o domínio do problema.

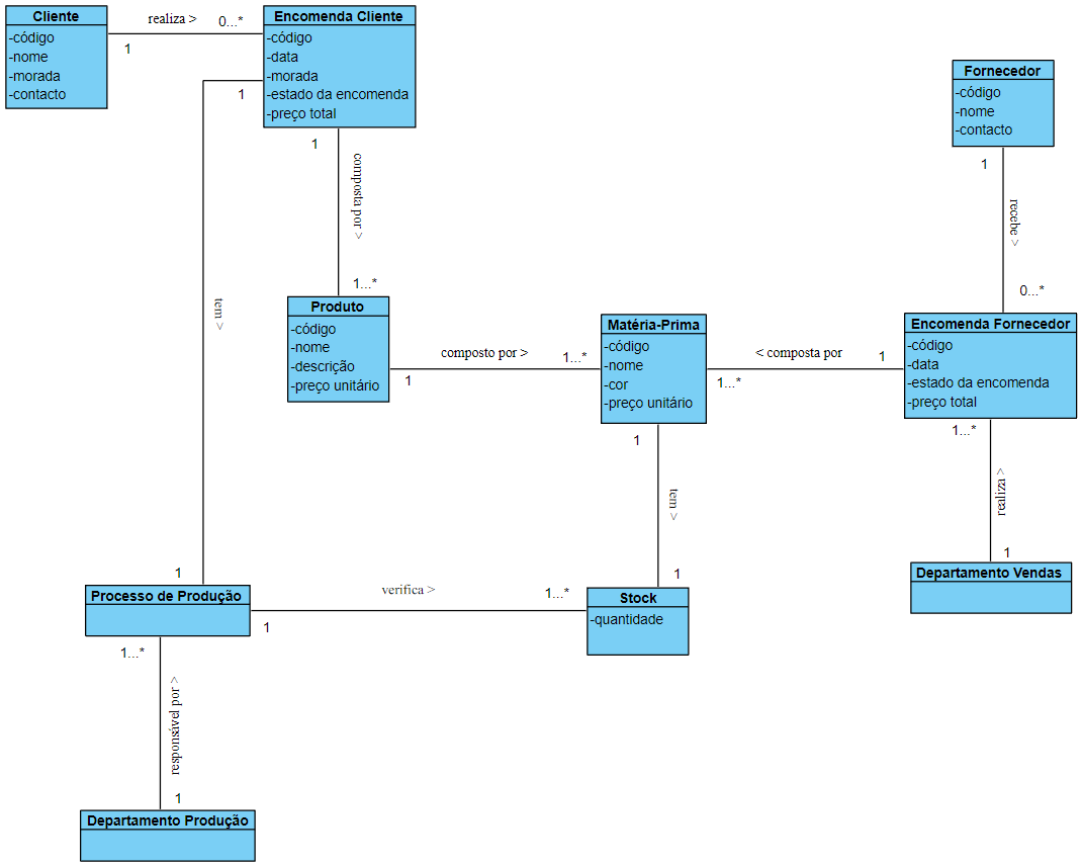


Figura 24 Modelo de domínio do processo de gestão de encomendas Alfa Lda

Assim, a empresa Alfa Lda. pretende um sistema de gestão de encomendas que informatize o processo descrito anteriormente, disponibilizando duas aplicações web acedidas via *browser* – uma para os seus clientes e outra para os seus colaboradores envolvidos no processo. Estes colaboradores pertencem ao departamento de produção e ao departamento de vendas, sendo que a empresa considera necessário que, pelo menos um colaborador do departamento de produção seja responsável por gerir a informação de produtos e matérias-primas no novo sistema.

Pretende-se ainda que o sistema de gestão de encomendas considere os seguintes aspetos:

- O código de matérias-primas, produtos, fornecedores, encomendas de fornecedores e encomendas de clientes deverá seguir o seguinte formato, respetivamente: 10210X, 11210X, 12210X, 13210X e 14210X, sendo X um número a incrementar na sua criação;
- Quando o cliente cria uma nova encomenda, o estado desta deve ser definido tendo em conta a disponibilidade ou não de stock (*Started* ou *Waiting Stock*).
- Relativamente ao cálculo dos preços totais das encomendas, deve-se considerar:
    - Preço total encomenda de cliente:

$$\sum (\text{preço unitário do produto X quantidade})$$

    - Preço total encomenda a fornecedor:

$$\sum (\text{preço unitário da matéria} - \text{prima X quantidade})$$

Para além disso, a empresa pretende que o novo sistema seja intuitivo, adaptado e personalizado às suas necessidades. Considera também que este sistema deve estar capacitado para eventuais melhorias e alterações, a curto ou longo prazo, conforme as necessidades da empresa e dos seus colaboradores e clientes. Uma das principais preocupações da empresa para a nova implementação de uma solução relaciona-se com a segurança e a disponibilidade do sistema. Uma vez que se pretende que o novo sistema dê suporte em todas as fases deste processo, a sua indisponibilidade ou perda de informação compromete todo o processo de produção.

4.1.2 Requisitos Funcionais

Considerando o contexto exposto, para este caso de estudo identificaram-se, os seguintes atores:

- **Administrador:** responsável pela gestão de toda a informação da plataforma;

- **Operador de produção:** pertencente ao departamento de produção e responsável pelo processo de produção;
- **Operador de vendas:** pertencente ao departamento de venda e responsável pela gestão de fornecedores e clientes e respetivas encomendas;
- **Cliente:** entidade que realiza encomendas de produtos na Alfa Lda.

Para cada um destes atores identificou-se os principais requisitos funcionais que estão representados no diagrama de casos de uso apresentado na Figura 25. Salienta-se que da descrição anterior podem existir outros (implícitos ou explícitos). Contudo, estes são aqueles que serão considerados na fase de construção.

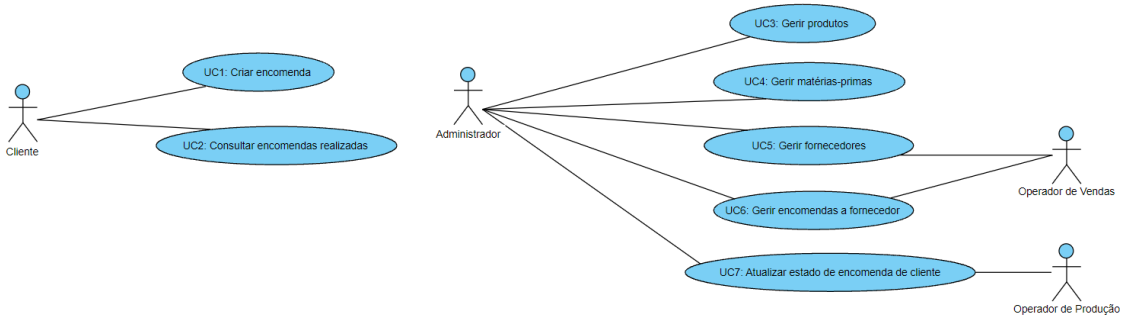


Figura 25 Diagrama de casos de uso

Tal como é possível verificar na figura apresentada, consideram-se os seguintes casos de uso:

- **UC1 - Criar Encomenda:** o cliente cria uma nova encomenda selecionando os produtos e a respetiva quantidade pretendida e introduzindo o endereço de entrega pretendido. Para além disso, esta encomenda apresenta uma data e o preço total. O sistema é responsável por, após a criação de uma nova encomenda, validar o stock de matérias-primas para a produção dos produtos encomendados e definir o estado inicial da encomenda;
- **UC2 - Consultar encomendas realizadas:** o cliente consulta as encomendas que realizou, verificando os detalhes das mesmas, bem como o seu estado;
- **UC3 – Gerir produtos:** o administrador pode consultar, criar, editar ou eliminar produtos, sendo estes caracterizados por código, nome, descrição, preço unitário e imagem;
- **UC4 – Gerir matérias-primas:** o administrador pode consultar, criar, editar ou eliminar matérias-primas, sendo estes caracterizados por código, nome, cor e preço unitário;

- **UC5 – Gerir fornecedores:** o administrador ou operador de vendas pode consultar, criar, editar ou eliminar fornecedores, sendo estes caracterizados por código, nome, email e número de telefone;
- **UC6 – Gerir encomendas a fornecedores:** o administrador ou operador de vendas pode criar ou editar o estado de encomendas a fornecedores, sendo estas caracterizadas por código, estado, fornecedor, matérias-primas a encomendar e respetiva quantidade e preço total;
- **UC7 – Atualizar estado de encomenda de cliente:** o administrador ou operador de produção pode consultar todas as encomendas dos clientes e editar o estado de encomenda de cliente, tendo em conta a progressão da encomenda no processo de produção.

Para o contexto do estudo a realizar, pretende-se que os requisitos a implementar apresentem níveis de complexidade diferentes. Isto porque, tipicamente, um sistema é composto por um conjunto, geralmente elevado, de operações de baixa/média complexidade e um conjunto, mais reduzido, de operações mais complexas.

Assim, consideram-se os requisitos UC1 e UC7 como os mais significativos, ao nível da definição da lógica do problema. Os restantes requisitos correspondem, essencialmente, ao desenvolvimento de operações CRUD básicas ou apenas consulta de informação, sendo estes considerados essenciais para o domínio, mas mais simples ao nível de lógica de negócio, logo mais simples de implementar. Contudo, importa realçar que, a este nível, existem operações mais simples do que outras, tendo em conta as relações existentes entre conceitos. Por exemplo, o CRUD de matéria-prima é mais simples do que o de produto, uma vez que este é composto por uma ou mais matérias-primas. Considera-se a implementação de mais do que um requisito deste tipo, de forma a ser possível avaliar a evolução de aprendizagem em cada ferramenta. Isto é, espera-se que, a implementação de um requisito semelhante a um já implementado seja mais rápida, uma vez que já existe conhecimento prévio.

4.1.3 Requisitos Não Funcionais

Os requisitos não-funcionais foram definidos seguindo a abordagem FURPS+ e tendo em conta a análise considerada em 4.1.1 Descrição de negócio. A especificação destes requisitos permitirá analisar as ferramentas em estudo ao nível dos aspetos considerados.

Funcionalidade

Este requisito corresponde aos aspetos funcionais do sistema, descritos anteriormente em 4.1.2 Requisitos Funcionais. Contudo, importa realçar a necessidade de considerar os seguintes aspetos:

- **Autenticação:** garantir que acedem ao sistema apenas utilizadores autenticados e que pertencem à Alfa Lda ou têm vínculo de cliente;
- **Autorização:** o sistema a desenvolver tem a participação de vários utilizadores, sendo que cada um deles tem um papel diferente (cf. 4.1.2 Requisitos Funcionais). Desta forma, dependendo do seu papel, o utilizador deverá ter acesso apenas às funcionalidades às quais tem permissão.

Usabilidade

Este requisito corresponde à interação do sistema com o utilizador final e, no contexto deste caso de estudo, é pertinente ter em conta o seguinte aspeto:

- **Padrões de *user* interface:** as interfaces gráficas apresentadas ao utilizador final devem, todas elas, seguir o mesmo esquema e disposição semelhante de informação, de forma a facilitar o processo de aprendizagem do utilizador no novo sistema.

Confiabilidade

Este requisito corresponde à disponibilidade ou, por exemplo, a recuperação a falhas. No contexto deste caso de estudo deve-se ter em conta o seguinte aspeto:

- **Disponibilidade:** o sistema deve apresentar alta disponibilidade sobretudo porque, como referido em 4.1.1 Descrição de negócio a indisponibilidade do sistema pode comprometer o processo de trabalho.

Desempenho

Este requisito permite avaliar o desempenho de um sistema ao nível de tempo de resposta, capacidade e escalabilidade. Para este caso de estudo devem-se considerar os seguintes aspetos:

- **Escalabilidade:** o sistema deve suportar um possível aumento de número de utilizadores e, consequentemente um aumento no número de pedidos ao sistema.
- **Tempo de resposta:** o sistema deve apresentar tempos de resposta baixos.

Suportabilidade

Este requisito considera algumas características como testabilidade, compatibilidade, manutenibilidade entre outros. Neste caso, deve-se ter em conta os seguintes aspetos:

- **Testabilidade:** deve ser possível garantir uma adequada cobertura de testes ao sistema;
- **Flexibilidade:** deve garantir-se a possibilidade de realizar alterações ao sistema existente.

4.1.4 Arquitetura

Tendo em conta o problema do presente caso de estudo e os objetivos deste projeto de dissertação (mais concretamente Q1), considera-se adequado adotar uma arquitetura cliente-servidor, combinada com uma arquitetura monolítica no lado do servidor.

Tal como é possível entender em 4.1.1 Descrição de negócio, a empresa fictícia deste caso pretende um sistema de gestão de encomendas que, nesta fase inicial, realize as funcionalidades mais simples, mas essenciais ao funcionamento adequado do seu processo de produção. A arquitetura pretendida, para além de ser a mais comum para este tipo de desenvolvimento apresenta, tipicamente, maior simplicidade e rapidez de desenvolvimento, em relação, por exemplo, a arquitetura orientada a microserviços (Fowler, 2015). Este tipo de arquitetura continua, pelo menos empiricamente, a ser um dos mais adotados no desenvolvimento de sistemas de informação como o pretendido. Para além disso, esta simplifica a implantação do servidor, ao mesmo tempo que permite que este seja desenvolvido considerando várias camadas (padrão de arquitetura em camadas), cada uma com uma responsabilidade específica.

Para a representação da vista lógica do sistema de nível 2, recorreu-se ao diagrama ilustrado na Figura 26 que apresenta os *containers* que constituem o sistema.

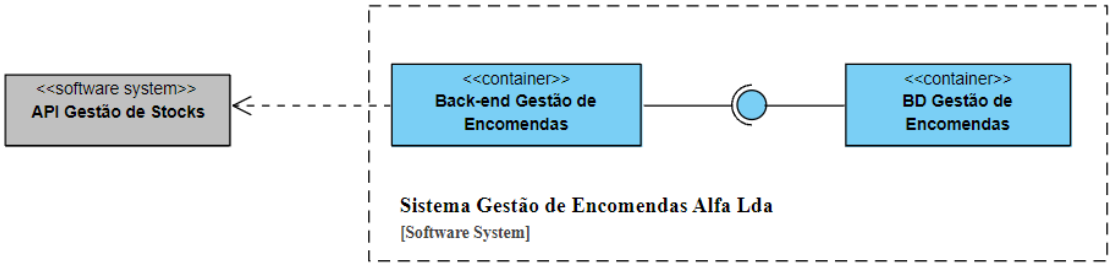


Figura 26 Vista lógica de nível 2 do caso de estudo 1

Pretende-se que o sistema seja composto pela base de dados, responsável pela gestão e armazenamento de toda a informação do sistema e pelo *back-end*, que fornece todas as funcionalidades do sistema. Externamente ao sistema, considera-se a API Gestão de Stocks[10] (cf. 4.1.1 Descrição de negócio). Para compreender a composição do *back-end*, apresenta-se na Figura 27 um diagrama de componentes, que representa a vista lógica de nível 3. Tal como se pode verificar no diagrama apresentado, no *back-end* deste sistema pretende-se adotar uma arquitetura em camadas. A definição desta arquitetura permite, ao nível do estudo das plataformas em análise, entender como estas suportam a implementação de soluções seguindo esta abordagem. Espera-se que, sendo estas plataformas direcionadas para *citizen developers*, apresentem níveis de abstração consideráveis.

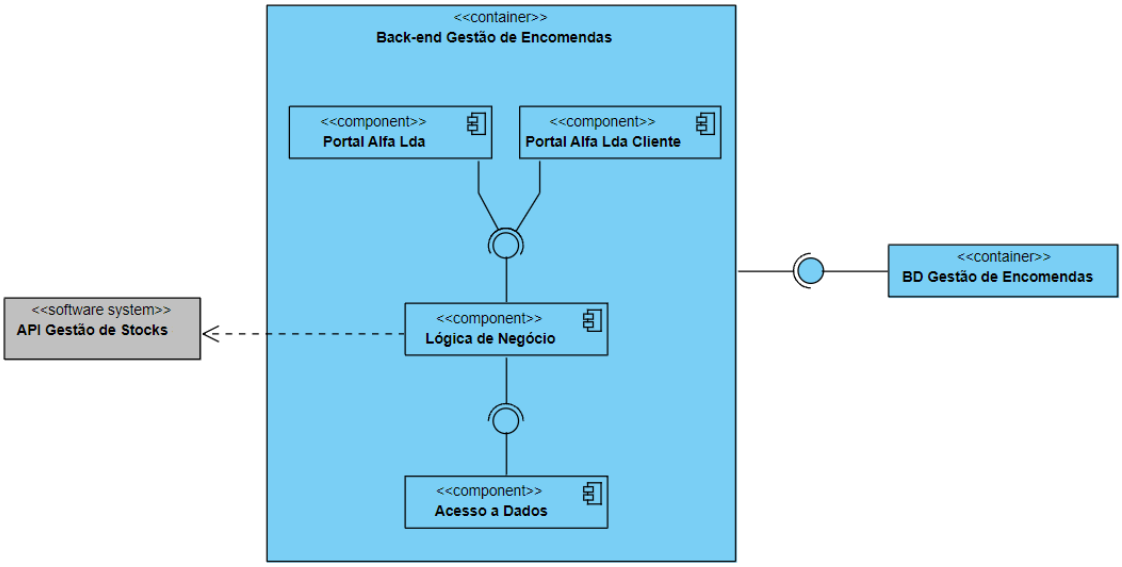


Figura 27 Vista lógica de nível 3 do caso de estudo 1

[10] Comunicação será realizada através de um pedido HTTP GET que recebe um parâmetro *path* com o código da matéria-prima que se pretende verificar o stock

O *container Back-end* Gestão de encomendas é composto pelos seguintes componentes: (i) Portal Alfa Lda, (ii) Portal Alfa Lda Cliente, (iii) Lógica de Negócio e (iv) Acesso a Dados. Estes componentes representam as várias camadas do servidor do sistema. A especificação do sistema desta forma, permite que cada camada tenha uma responsabilidade única, sendo que cada uma delas comunica com a camada imediatamente inferior à sua.

Os componentes Portal Alfa Lda e Portal Alfa Lda Cliente são responsáveis pelas páginas web apresentadas aos utilizadores com permissões de colaboradores ou clientes, respetivamente. Tendo em conta este caso de estudo, um exemplo de página apresentada é o formulário para criação de uma nova encomenda pelo cliente ou de um novo produto. Para além disso, estes componentes são responsáveis por gerir as permissões dos utilizadores, garantindo que os utilizadores têm acesso apenas às páginas autorizadas. Por esta razão, alternativamente, poderá considerar-se apenas um componente destinado a todos os utilizadores do sistema, mas que deve garantir acesso apenas a páginas e menus permitidos tendo em conta as permissões do utilizador.

O componente de lógica de negócio é responsável pela implementação de regras e lógica de negócio. Este componente comunica com os componentes descritos anteriormente, sendo responsável por processar todos os pedidos desse componente, tais como retornar informação ou criar um novo produto, tendo em conta as regras de negócio. Por sua vez, este componente comunica com o componente de acesso a dados. Este é responsável por simplificar o acesso aos dados da base de dados, evitando, assim, a utilização de comandos de base de dados para aceder às suas tabelas. De forma a uniformizar, no processo de implementação, a definição de entidades, definiu-se um diagrama de classes (ver Anexo 2 – Diagrama de classes) que apresenta as classes, os seus atributos e relações entre elas consideradas para o problema.

Alternativamente à arquitetura descrita podia considerar-se a arquitetura representada na Figura 28.

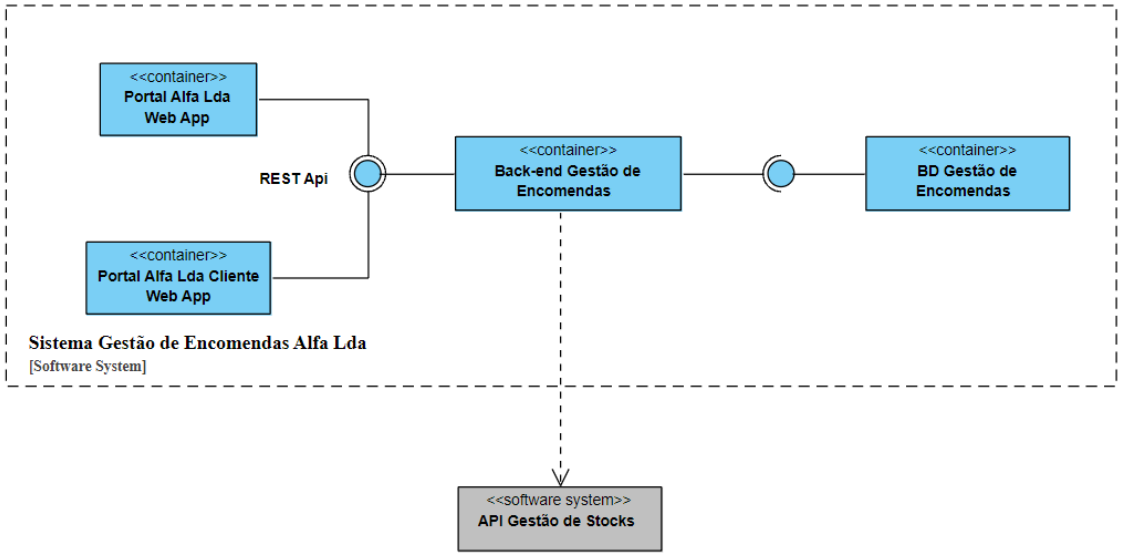


Figura 28 Vista lógica de nível 2 alternativa para o caso de estudo 1

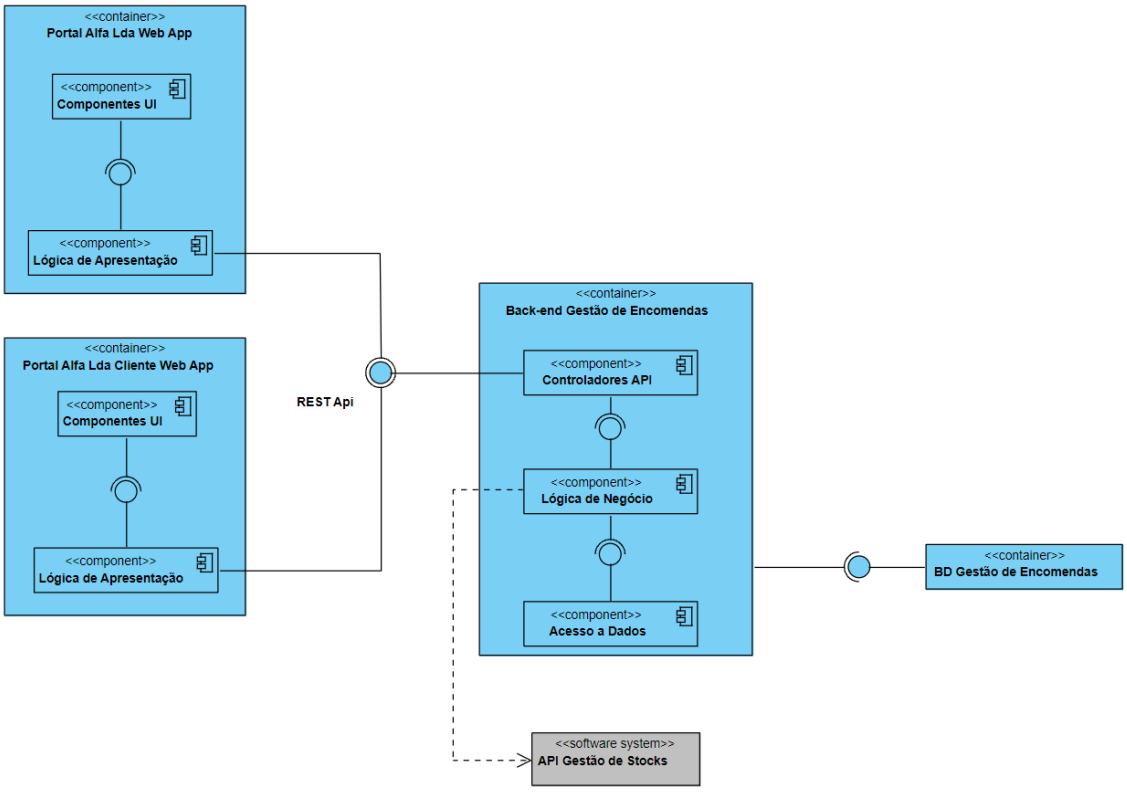


Figura 29 Vista lógica de nível 3 alternativa para o caso de estudo 1

Esta alternativa apresenta duas aplicações *web* que comunicam com o *back-end* através de pedidos REST Api. Esta abordagem encontra-se detalhada na Figura 29. Nesta alternativa o *back-end* é composto pelo componente de controladores API, lógica negócio e acesso a dados, sendo que o primeiro terá que considerar a implementação de controladores para estabelecer

a comunicação com as aplicações *web*. Cada uma destas aplicações é responsável pela implementação de interfaces e lógica de apresentação, sendo esta necessária para garantir a comunicação com o *back-end*.

Para uma melhor compreensão da interação entre os *containers* que constituem o sistema, apresenta-se, na Figura 30 o diagrama de sequência para o requisito funcional UC3: Gerir Produtos (cf. 4.1.2 Requisitos Funcionais), considerando a criação de um novo produto. Este diagrama permite representar a vista de processo da arquitetura.

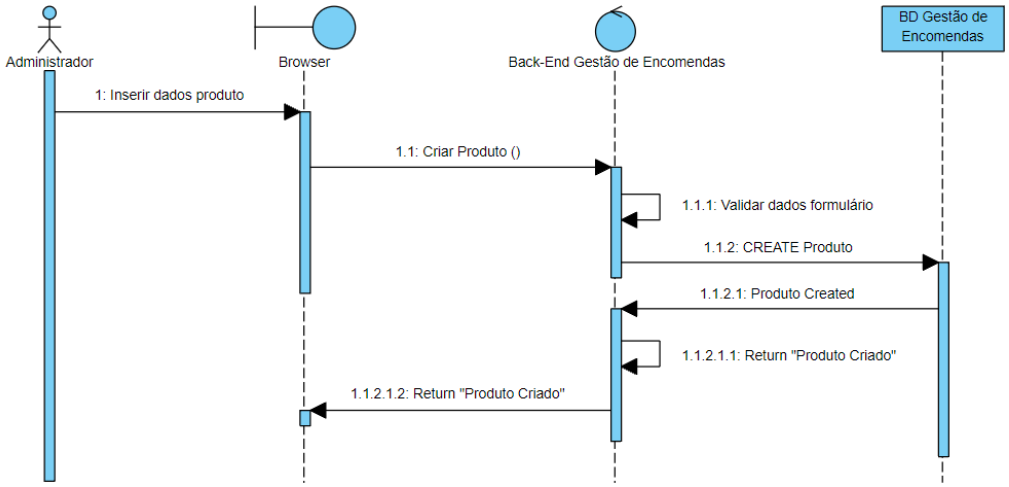


Figura 30 Vista de processo para UC3 - Gerir Produtos (criar produto)

O processo de criação de um novo produto inicia-se quando o administrador insere os dados do produto. O *back-end* do sistema recebe a informação do produto, valida-a e, em caso de os dados serem válidos, é criado e guardado o novo produto na base de dados. Por fim, o *back-end* retorna uma mensagem de sucesso.

No caso do requisito UC1: Criar Encomenda de Cliente (cf. 4.1.2 Requisitos Funcionais), uma vez que se pretende verificar o stock de matérias-primas, o fluxo entre os *containers* do sistema considera a interação com o sistema externo API Gestão de Stock. Este fluxo apresenta-se na Figura 31.

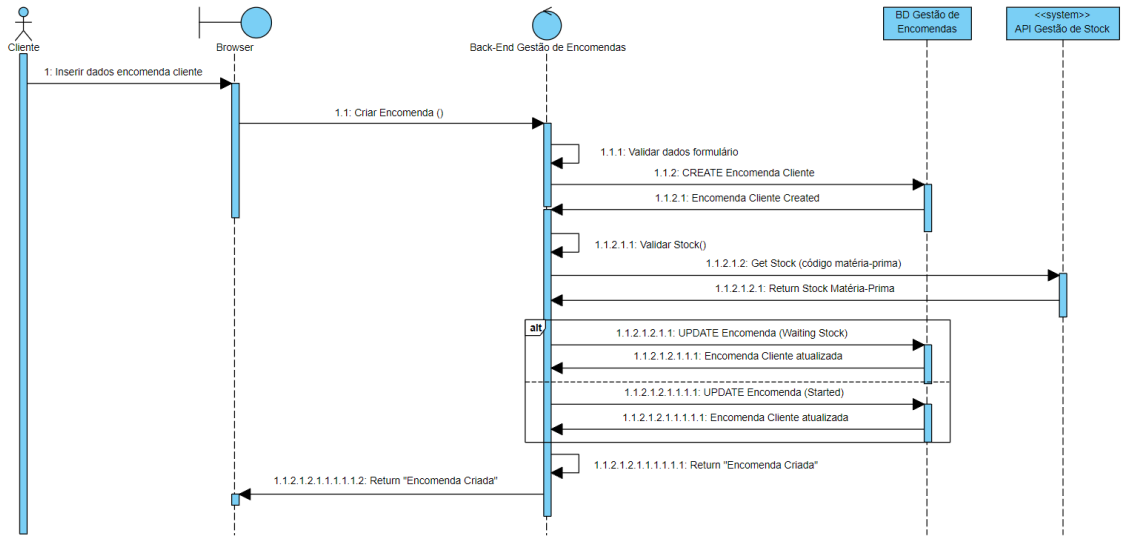


Figura 31 Vista de processo para UC1: Criar Encomenda

O processo de criar encomenda inicia-se quando o cliente insere os dados da encomenda. O *back-end* recebe essa informação, valida-a e, no caso desta ser válida, a encomenda é criada e guardada na base de dados, considerando o estado *Open*. Depois, o *back-end* inicia o processamento da encomenda, enviando um pedido para o sistema externo, de forma a obter o stock disponível. No caso de não existir stock disponível, o estado da encomenda é alterado para *Waiting Stock*, caso contrário é alterado para *Started*. Por fim, o *back-end* retorna uma mensagem de sucesso.

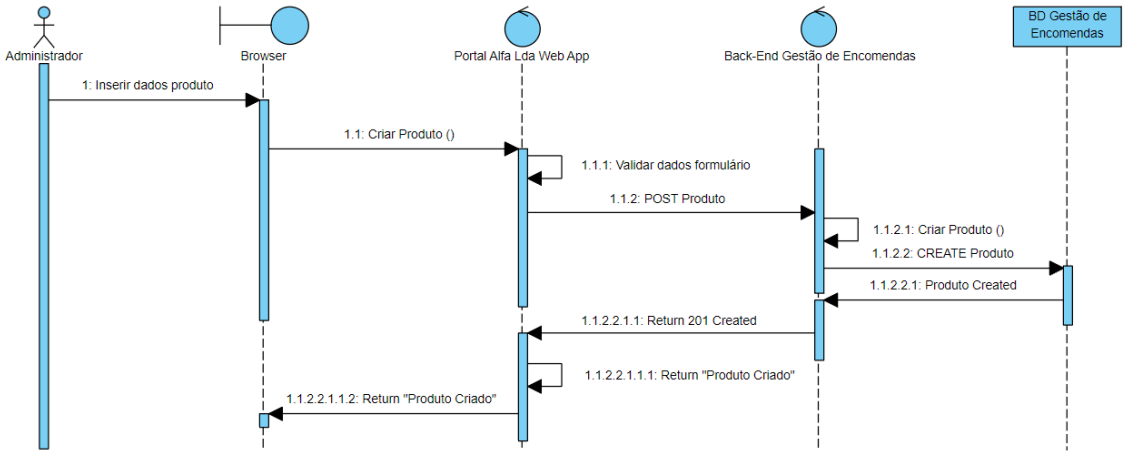


Figura 32 Vista de processo alternativa para UC3: Gerir Produtos (criar produto)

Considerando a alternativa descrita anteriormente apresenta-se, na Figura 32, o diagrama de sequência para o requisito funcional UC3: Gerir Produtos (cf. 4.1.2 Requisitos Funcionais),

considerando a criação de um novo produto. É possível observar que, nesta alternativa, a validação dos dados inseridos pelo utilizador acontece em Portal Alfa Lda Web App, seguindo-se o envio do pedido para criar produto para o *back-end* do sistema. Este recebe e processa a informação, sendo, posteriormente, criado o produto na base de dados. No caso de criação do produto com sucesso, o *back-end* envia uma resposta ao Portal Alfa Lda Web App que, por sua vez processa-a e retorna uma mensagem de sucesso para o utilizador.

A representação da vista lógica de nível 2 permitiu representar, numa vista de implantação, os principais elementos do modelo físico do sistema e a comunicação entre eles. Para a representação dessa vista recorreu-se ao diagrama de implantação apresentado na Figura 33.

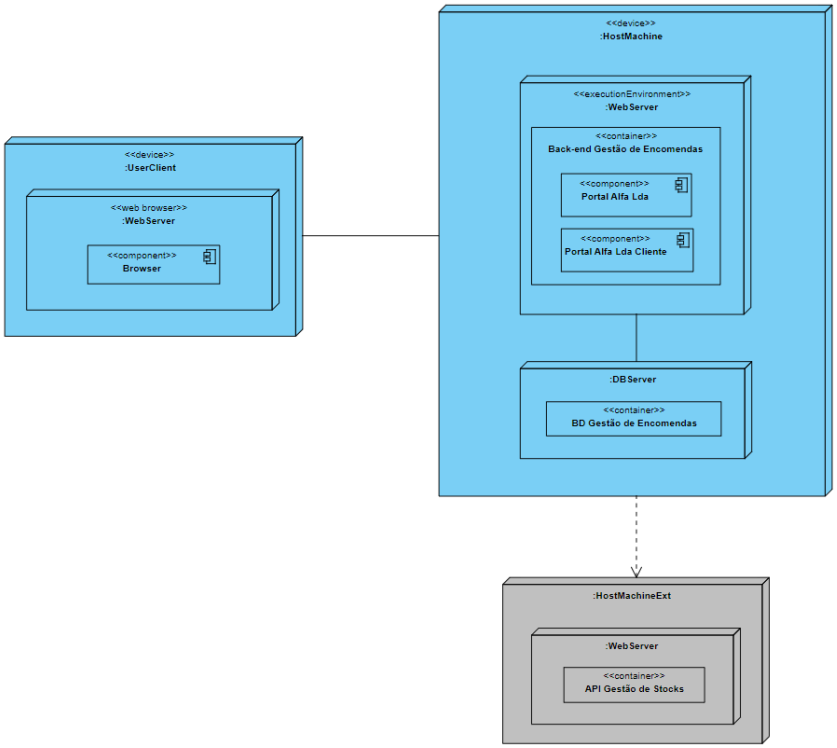


Figura 33 Vista de implantação do caso de estudo 1

Tal como se pode verificar no diagrama apresentado o modelo físico do sistema é composto pelo nó *HostMachine* e pelo nó *UserClient*. O nó *HostMachineExt* representa a implantação do sistema externo. O nó *HostMachine* representa a instalação do servidor do sistema sendo que, neste caso pretende-se que este seja instalado na mesma máquina. Sendo assim, considera-se que nesta abordagem serão desenvolvidas aplicações SSR (*Server Side Render)*, uma vez que as aplicações Portal Alfa Lda e Portal Alfa Lda Cliente são executadas no servidor, sendo que este fornece ao cliente - *UserClient* - a interface das páginas.

Por outro lado, considerando a alternativa descrita anteriormente apresenta-se na Figura 34 o diagrama de implantação para essa abordagem.

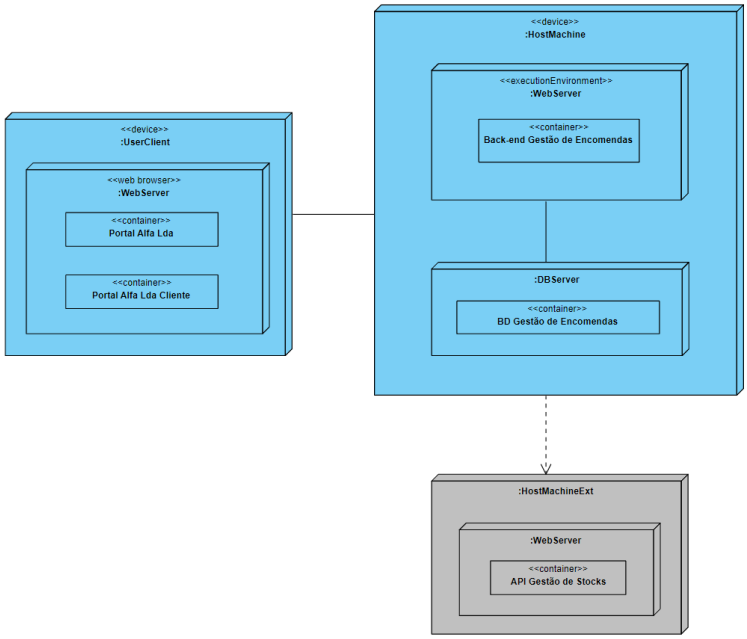


Figura 34 Vista de implantação alternativa para o caso de estudo 1

Nesta alternativa, o modelo físico é composto pelos nós *UserClient* e *HostMachine*. O nó *HostMachine* representa a instalação do servidor do sistema e o nó *UserClient* representa a instalação das aplicações web destinadas aos utilizadores, que são acedidas via browser. Nesta abordagem, é possível verificar que todo o *front-end* das aplicações é executado, de uma só vez, no lado do cliente, sendo que este comunica com o servidor, enviando e recebendo apenas os dados que necessita. Neste caso, as aplicações descritas são consideradas SPAs (*Single Page Applications*), sendo que, quando solicitam novos dados ao servidor, no lado do cliente são atualizadas apenas algumas partes de uma página, não se considerando a atualização de uma página inteira.

A consideração desta vista permite que se estude e compare o processo de implantação do sistema nas plataformas em estudo. Mais uma vez, espera-se que, a este nível, estas ferramentas apresentem um elevado nível de abstração, simplificando este processo. Para além disso, deve-se considerar também a alternativa descrita e, neste nível, será oportuno entender se as plataformas permitem e adotam uma abordagem mais tradicional - SSR - ou se, por outro lado, adotam uma abordagem mais atual – SPA.

4.2 Caso de estudo 2

Este subcapítulo apresenta a descrição do domínio de negócio deste casos de estudo e a descrição dos principais aspetos de design a considerar para a sua implementação.

Tal como referido em 4.1.4 Arquitetura a arquitetura adotada permite o desenvolvimento de soluções num menor intervalo de tempo, numa fase inicial, uma vez que o desenvolvimento e instalação são mais simples (*deploy* num único servidor). Contudo, (Fowler, 2015) refere que à medida que a complexidade do projeto aumenta a arquitetura adotada no caso de estudo 1 tende a não responder de forma satisfatória, sendo que a manutenção da solução torna-se cada mais complexa e demorada. O mesmo autor refere que a adoção de uma arquitetura orientada a microserviços apesar de, inicialmente, exigir mais tempo de desenvolvimento, tende a responder melhor a alterações e consequente aumento de complexidade dos projetos. Neste sentido, o caso de estudo 2 evolui o contexto anterior de forma a, entre outros, conduzir à adoção de uma arquitetura orientada a microserviços. Desta forma tornará possível, por um lado, avaliar a resposta a evoluções arquiteturais por parte das ferramentas em estudo e, por outro lado, avaliar se, de facto, esta arquitetura permitirá simplificar a manutenção e implementação de futuros desenvolvimentos nestas ferramentas.

Com este caso de estudo pretende-se que, para além da adoção de uma nova arquitetura, seja possível construir e analisar soluções, sendo que o seu desenvolvimento deve focar-se, essencialmente, na manutenção e adaptação do sistema implementado no caso de estudo anterior. Desta forma, será possível analisar e comparar a resposta de cada plataforma face a estas alterações e exigências da nova arquitetura.

4.2.1 Descrição de negócio

No seguimento do caso de estudo 1, a empresa fictícia Alfa Lda. solicitou feedback dos utilizadores finais do sistema implementado. Ao nível do portal destinado a tarefas de produção e venda, o feedback dos utilizadores foi positivo, tendo estes considerado o sistema adequado para as suas tarefas. Por outro lado, ao nível do portal destinado aos clientes, apesar destes considerarem o sistema apropriado, solicitaram a hipótese de o portal apresentado passar a estar disponível numa aplicação móvel, facilitando, assim, o processo de criação de uma nova encomenda.

Assim, a Alfa Lda. pretende manter o sistema de gestão de encomenda, continuando a disponibilizar aos seus colaboradores e cliente uma aplicação *web* acedida via *browser*. Adicionalmente, a empresa pretende disponibilizar aos seus clientes uma aplicação móvel, com as mesmas funcionalidades da aplicação *web*.

Mais uma vez, a empresa pretende que o sistema se mantenha intuitivo e personalizado às suas necessidades, bem como esteja capacitado para eventuais melhorias e alterações, conforme as necessidades da empresa, dos seus colaboradores e dos seus clientes. Para esta nova versão do sistema, uma das principais preocupações da empresa está relacionada com a disponibilidade do sistema. Isto porque, a tendência é o número de cliente aumentar, aumentando, consequentemente, o número de encomendas, sendo assim o sistema será solicitado mais vezes, podendo a sua indisponibilidade comprometer os novos pedidos dos clientes.

4.2.2 Requisitos Funcionais

Para este caso de estudo consideram-se os mesmos atores e requisitos funcionais identificados em 4.1.2 Requisitos Funcionais.

4.2.3 Requisitos Não-Funcionais

Para este caso de estudo consideram-se os mesmos aspetos identificados em 4.1.3 Requisitos Não Funcionais e, adicionalmente:

Funcionalidade

No contexto deste caso de estudo deve-se ter em conta o seguinte aspeto:

- **Autenticação:** garantir que acedem à aplicação apenas utilizadores registados.

Usabilidade

No contexto deste caso de estudo, é pertinente ter em conta o seguinte aspeto:

- **Facilidade de utilização:** as interfaces gráficas fornecidas ao utilizador devem ser intuitivas, devem seguir um padrão comum que permita o utilizador memorizar com mais facilidade as ações que realiza na aplicação. Para além disso, deve garantir-se o menor número de cliques possível entre as várias ações.

Confiabilidade

No contexto deste caso de estudo deve-se ter em conta o seguinte aspeto:

- **Disponibilidade:** o sistema deve apresentar alta disponibilidade sobretudo porque, como referido em 4.2.1 Descrição de negócio a aplicação pode apresentar uma maior procura e a indisponibilidade da aplicação compromete o pedido dos utilizadores.

4.2.4 Arquitetura

O caso de estudo 2, contrariamente ao anterior, resulta numa melhoria do sistema implementado previamente (cf. 4.2.1 Descrição de negócio). Esta melhoria prevê a implementação de uma aplicação móvel que complementará a aplicação *web* existente para clientes. Para este novo sistema considera-se a adoção de uma arquitetura orientada a microserviços.

A decomposição de um sistema em micro serviços nem sempre é linear, pois esta depende de vários fatores, tais como a complexidade do negócio e a granularidade que se pretende. Neste caso de estudo, considera-se que a solução a desenvolver é de pequena dimensão e apresenta baixa complexidade. Para entender o processo de decomposição de microserviços no âmbito deste caso de estudo é necessário retomar a 4.1.1 Descrição de negócio e considerar a representação do modelo de domínio que permitiu compreender as principais entidades do negócio, considerando entidades que evoluirão para classes de domínio e outras que apenas estão representadas para clarificar o contexto do domínio (e.g. Stock, Processo de Produção, Departamento de Produção e Departamento de Vendas).

Tendo como ponto de partida o modelo de domínio apresentado utilizou-se o padrão *aggregate* de DDD (*Domain-Driven Design*). Um agregado corresponde a um conjunto de objetos de domínio que podem ser tratados como uma unidade e é composto por *aggregate root* e por *value objects* (Richardson, 2019). Na prática, este padrão permite decompor um modelo de domínio em vários partes, considerando os limites de cada uma delas (Richardson, 2019). Contudo para esta decomposição é necessário considerar algumas regras, por exemplo, manter as relações entre agregados apenas a partir de *aggregate root* e considerar a utilização de chaves primárias para estas referências (Richardson, 2019). Assim, para este caso de estudo apresenta-se na Figura 35 o modelo de domínio (apenas com as entidades de domínio mais

relevantes para este contexto) estruturado, tendo em conta os agregados identificados e os limites destes.

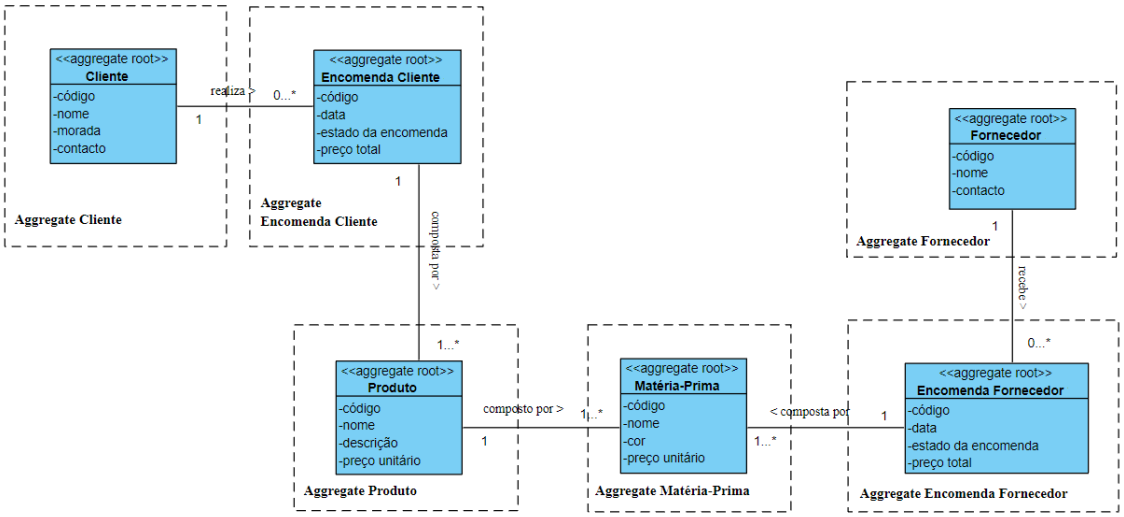


Figura 35 Decomposição do modelo de domínio em agregados, segundo o padrão *aggregate.*

Tal como se pode verificar foram identificados seis agregados, tendo em conta o seu contexto de domínio e as referências entre as entidades, estando cada um deles relacionado com outros, através de *aggregate root*. Idealmente, os agregados devem ser de pequena dimensão, para melhorar a escalabilidade e, neste sentido, considera-se que os agregados identificados apresentam uma dimensão adequada, para além de terem limites bem definidos.

A decomposição segundo o padrão *aggregate* resulta em vários agregados, podendo cada um destes tornar-se num microserviço. De facto, idealmente cada um dos agregados identificados resultaria num microserviço, sendo assim o sistema seria decomposto em seis microserviços. Contudo, esta decomposição, alinhada com a baixa complexidade do sistema, levantou algumas questões, nomeadamente ao nível dos ganhos de uma arquitetura deste tipo. Isto porque, decompor um sistema em microserviços demasiado pequenos, pode, facilmente, sobrecarregar o sistema e aumentar a sua complexidade, o que coloca em causa as vantagens desta arquitetura (IBM, 2021). Desta forma, optou-se por considerar serviços de maior dimensão, resultantes da união de agregados identificados anteriormente, tendo em conta o seu domínio no âmbito do problema, isto é: (i) Cliente e Encomenda Cliente; (ii) Produto e Matérias-Primas; (iii) Fornecedor e Encomenda Fornecedor. Para simplificar a sua designação consideram-se as abreviaturas Serviço CEC, Serviço PMP e Serviço FEF, respetivamente, que serão consideradas ao longo deste documento. No entanto, importa salientar que estes microserviços deixam de

apresentar uma única responsabilidade. Apesar disso, nesta abordagem considera-se que, ao longo do desenvolvimento e evolução de um sistema, os microserviços podem ser separados, à medida que apresentem características que o justifiquem e quando as mudanças no sistema já são mais lentas e difíceis.

Assim, pretende-se que o novo sistema considere a desenvolvimento dos microserviços identificados cuja comunicação com as aplicações finais deverá ser realizada através de uma API *Gateway*. Como aplicações finais consideram-se o Portal Alfa Lda, Portal Alfa Lda Cliente e Aplicação Móvel Alfa Lda Cliente, sendo estes *containers* responsáveis pelas aplicações que serão apresentadas aos utilizadores finais. Apesar de ser possível que estas aplicações cliente e os serviços comuniquem entre si diretamente, o recurso a uma API *Gateway* permite criar uma camada intermediária entre eles, sendo possível desacoplar a interface do cliente com a implementação do *back-end*. A Figura 36 apresenta o diagrama que representa a vista lógica de nível 2 do sistema a desenvolver.

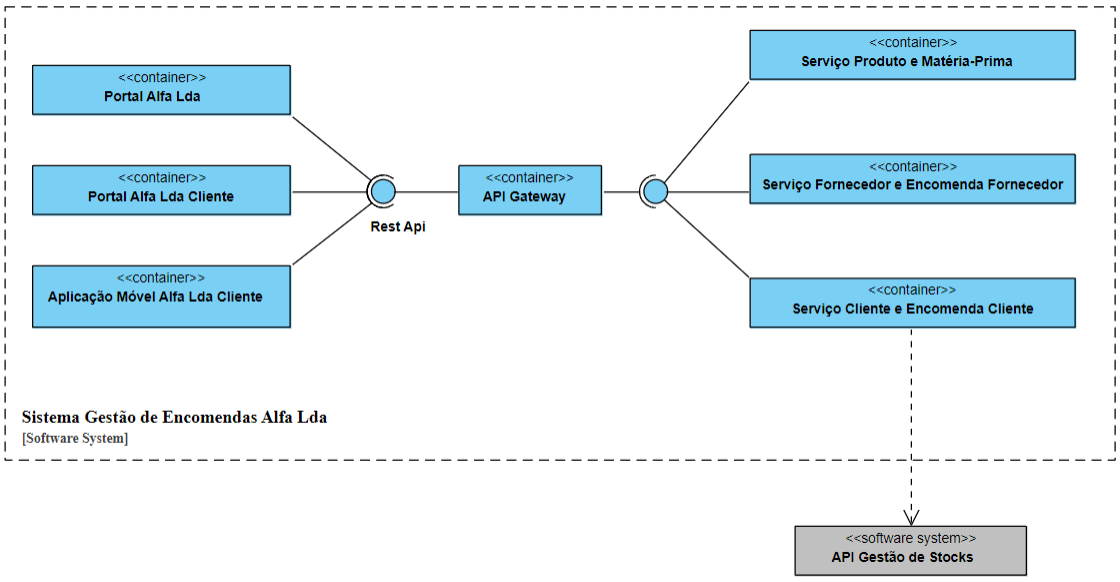


Figura 36 Vista lógica de nível 2 do caso de estudo 2

As aplicações Portal Alfa Lda e Portal Alfa Lda Cliente já foram consideradas no caso de estudo anterior, pelo que, neste contexto, pretende-se que estas sejam adaptadas, de forma a responderem às necessidades desta arquitetura. Esta adaptação permitirá analisar a forma como as plataformas em estudo respondem à necessidade de mudança e readaptação ao nível da arquitetura de um sistema. Para além disso, poderá também ter-se em conta a alternativa

considerada em 4.1.4 Arquitetura, para se perceber se a sua adoção apresentaria vantagens ao nível destas adaptações nas plataformas.

Adicionalmente, pretende-se que o sistema apresente uma aplicação móvel destinada aos clientes. A este nível pretende-se verificar quais as principais diferenças (caso existam) no desenvolvimento de uma aplicação web e aplicação mobile. Para além disso, pretende-se analisar a oferta das plataformas ao nível do desenvolvimento de aplicações nativas e ao nível de uma abordagem emergente como PWA (*Progressive Web App*).

Para uma melhor compreensão da composição dos *containers* do sistema, apresenta-se na Figura 37 um diagrama de componente que representa uma vista lógica de nível 3.

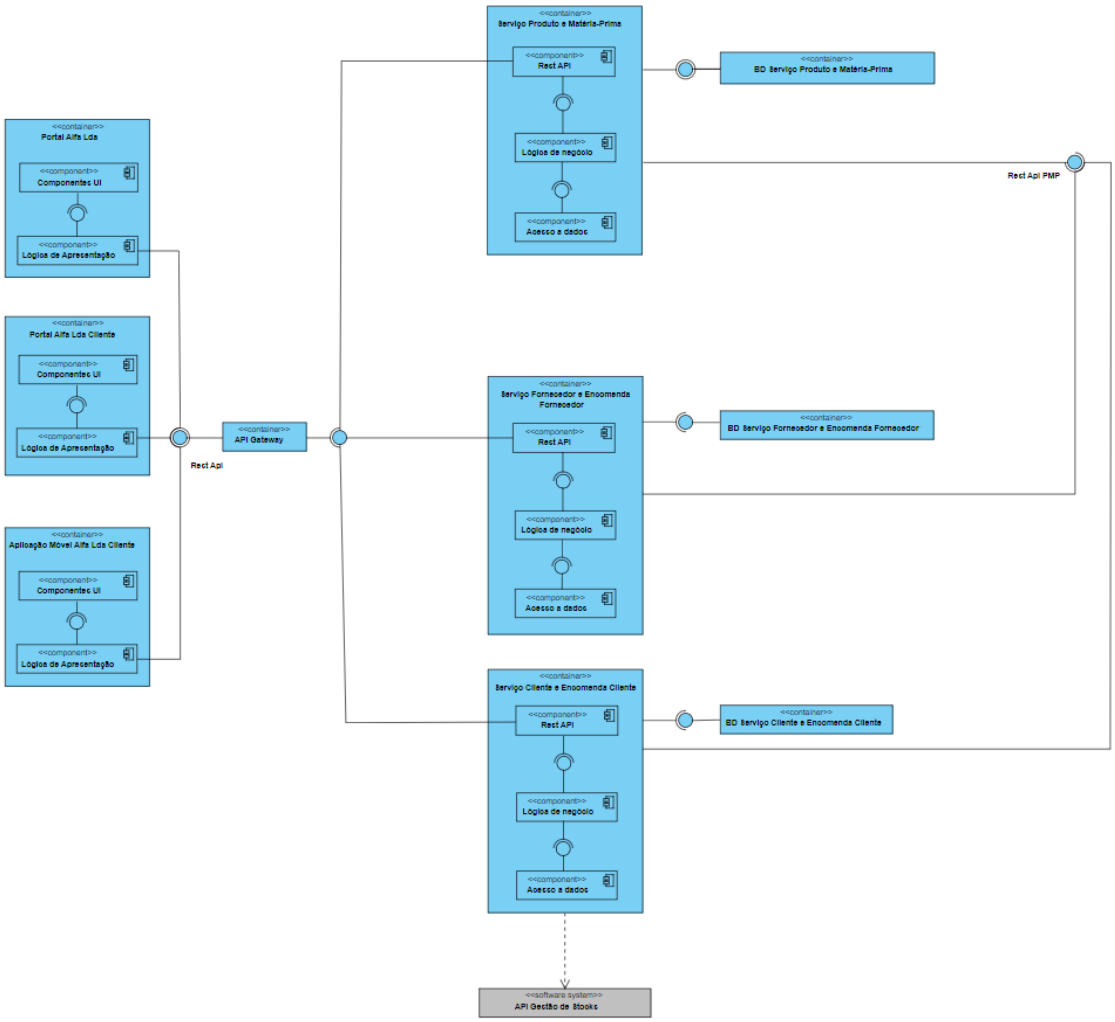


Figura 37 Vista lógica de nível 3 do caso de estudo 2

Ao nível dos serviços pretende-se que estes apresentem, à semelhança do servidor considerado em 4.1.4 Arquitetura, componentes com responsabilidades específicas – lógica de negócio e acesso a dados - sendo que, adicionalmente, é necessário considerar que cada microserviço deve disponibilizar uma *Rest Api*. Cada API é mapeada na *API gateway*, para estabelecer a comunicação com as aplicações cliente. No diagrama apresentado considera-se a comunicação dos serviços a considerar no desenvolvimento dos requisitos do sistema. Estas comunicações são reduzidas, uma vez que se consideraram microserviços maiores, tendo isto resultado na redução de dependências e chamadas entre eles. Desta forma, optou-se por considerar o padrão de comunicação síncrona, sendo que os microserviços comunicam entre si através de chamadas à API que cada um deles fornece. Por fim, pretende-se que cada serviço tenha uma base de dados própria, seguindo, assim o padrão *database per service*, que garante a persistência de dados de cada serviço, sendo estes acessíveis apenas através da sua API (Richardson, 2021). Alternativamente, poder-se-ia considerar o padrão *shared-database-per-service* que considera que a mesma base de dados é partilhada por vários microserviços. Contudo, muito frequentemente este é considerado um não-padrão uma vez que apresenta desvantagens como o acoplamento de tempo de execução ou a capacidade de não reduzir as dependências entre as equipas de desenvolvimento.

Ao nível das aplicações cliente pretende-se que sejam compostas pelos componentes: (i) Componentes UI responsáveis pela interface gráfica e (ii) Lógica de apresentação responsável pela lógica necessária para comunicar com a API Gateway. Ao nível do Portal Alfa Lda e Portal Alfa Lda Cliente pretende-se que mantenham os componentes implementados no caso de estudo 1, sendo apenas necessário considerar a implementação da lógica de apresentação.

Alternativamente à arquitetura descrita apresenta-se na Figura 38 um diagrama de componente que representa a vista lógica de nível 3 alternativa para o sistema.

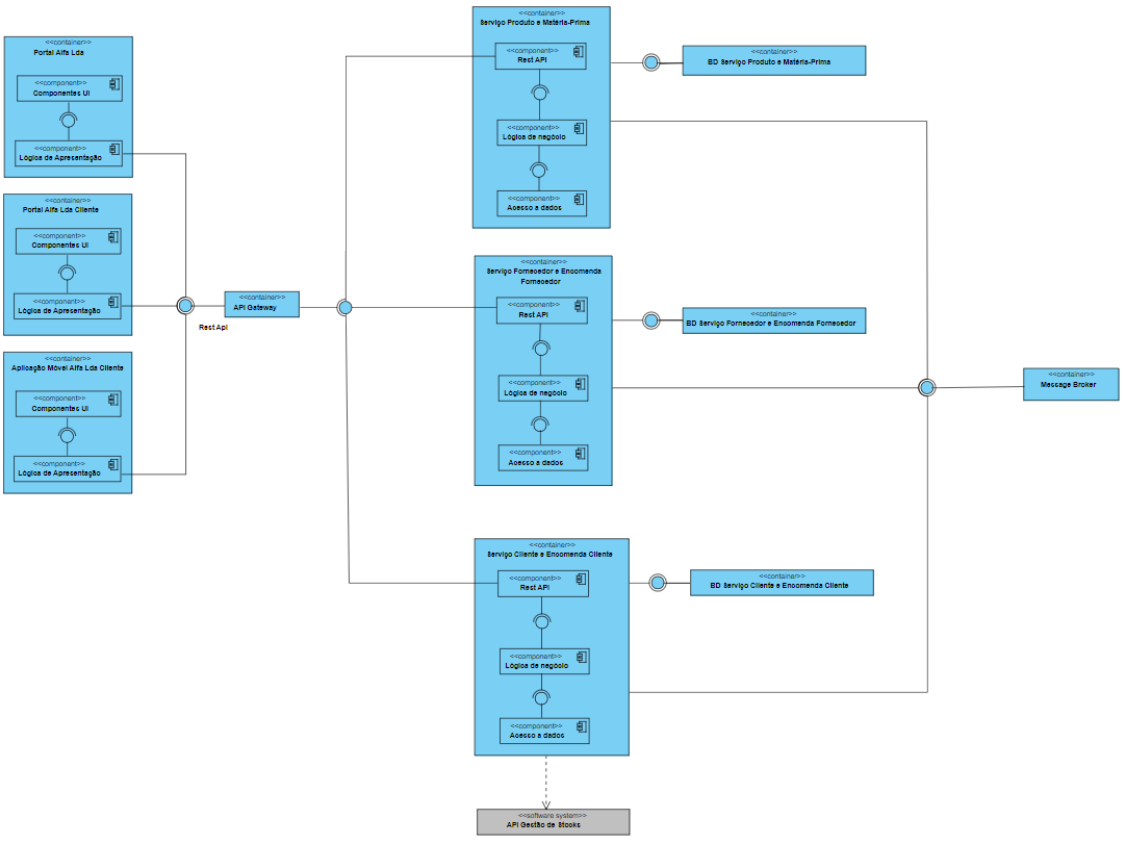


Figura 38 Vista lógica de nível 3 alternativa do caso de estudo 2

Esta alternativa considera um padrão assíncrono de mensagens entre os microserviços, sendo que neste caso um serviço envia uma mensagem sem esperar por uma resposta e um ou mais serviços processam a mensagem de forma assíncrona, considerando protocolos como AMQP. Nesta alternativa apresenta-se um *message broker* que corresponde a uma tecnologia (p.e. *RabbitMQ*, *Kafka*, etc) e que permite que os serviços comuniquem e troquem informação entre si (IBM, 2020). Tipicamente este é composto por um componente designado fila de mensagens, que armazena e ordena as mensagens até que os serviços as possam processar (IBM, 2020). Para além disso, este apresenta dois padrões de distribuição de mensagem, sendo que, para este caso de estudo, seria considerado o padrão de publicação e subscrição de mensagens (IBM, 2020). Neste padrão o produtor de cada mensagem publica-a como um tópico e os consumidores subscrevem os tópicos dos quais querem receber mensagens (IBM, 2020). Esta abordagem poderia permitir uma resposta mais rápida, sobretudo quando se apresentam muitas dependências entre serviços. Ou seja, esta abordagem poderia ser considerada se as dependências entre os serviços do sistema a desenvolver aumentassem. Pretende-se, com esta alternativa, analisar se, ao nível das plataformas, é possível considerar a integração com um

message broker e, em caso afirmativo, quais as tecnologias/conectores que disponibilizam para o efeito.

Para uma melhor compreensão do fluxo entre os principais *containers* do sistema apresenta-se, na Figura 39, o diagrama de sequência que representa a vista de processo desta arquitetura. O diagrama representa o fluxo para o requisito funcional UC1: Criar Encomenda (cf. 4.1.2 Requisitos Funcionais).

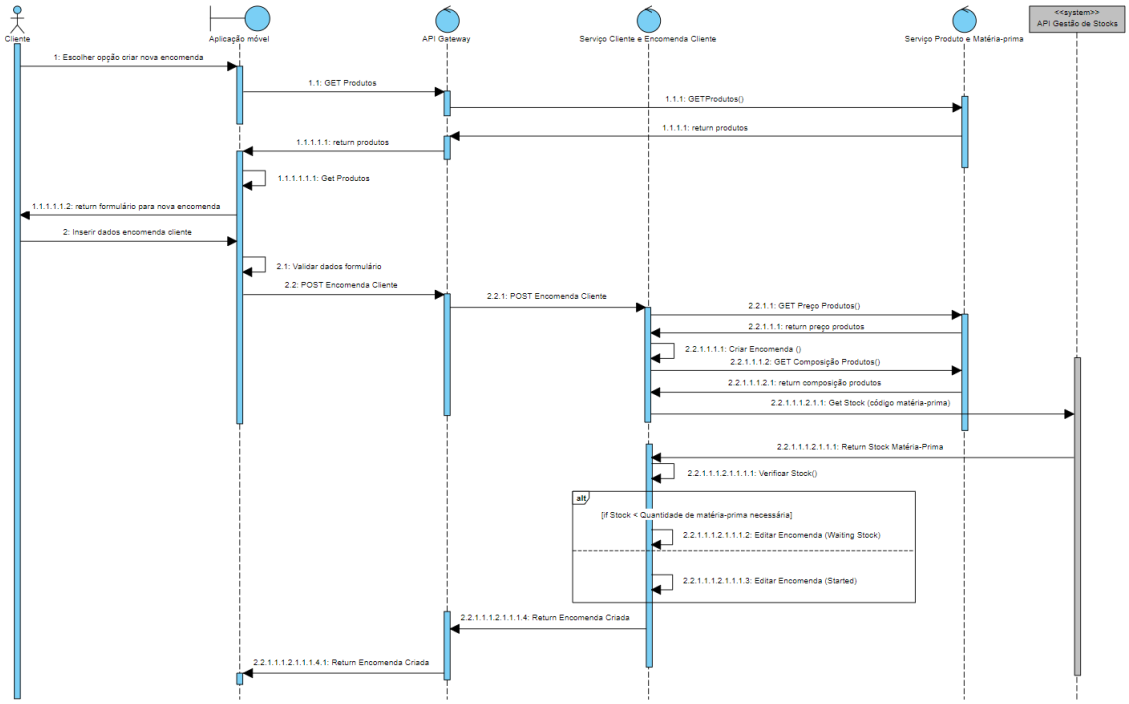


Figura 39 Vista de processo para UC1: Criar Encomenda

O processo de criação de encomenda inicia-se quando o cliente seleciona a opção para criar uma nova encomenda. Neste momento, obtêm-se os produtos através de um pedido à API *Gateway* que direciona o pedido para o Serviço PMP. Este serviço retorna os produtos e a API *gateway* redireciona esta resposta para a aplicação móvel que apresenta ao utilizador o formulário com a lista de todos os produtos. O cliente insere os dados da encomenda, estes são validados e um pedido com esta informação é enviado à API *gateway*, que o direciona para o Serviço CEC. Este serviço faz um pedido ao Serviço PMP para obter o(s) preço(s) do(s) produto(s) para, posteriormente, calcular o preço da encomenda e criar a encomenda. Depois, realiza-se um novo pedido ao Serviço PMP para obter a composição dos produtos encomendados. Com esta informação é possível fazer o pedido para obter o stock de matéria-prima. O Serviço CEC verifica a quantidade de stock e atualiza o estado da encomenda conforme a disponibilidade ou

não de stock. Por fim, é retornada a resposta para a API *Gateway* que, por sua vez, retorna esta resposta para a aplicação móvel, que apresenta a mensagem ao cliente.

A definição dos containers do sistema numa perspetiva de vista lógica de nível 2, permitiu representar, numa vista de implantação, os principais elementos do modelo físico do sistema e a comunicação entre eles. A Figura 40 apresenta o diagrama de implantação do sistema.

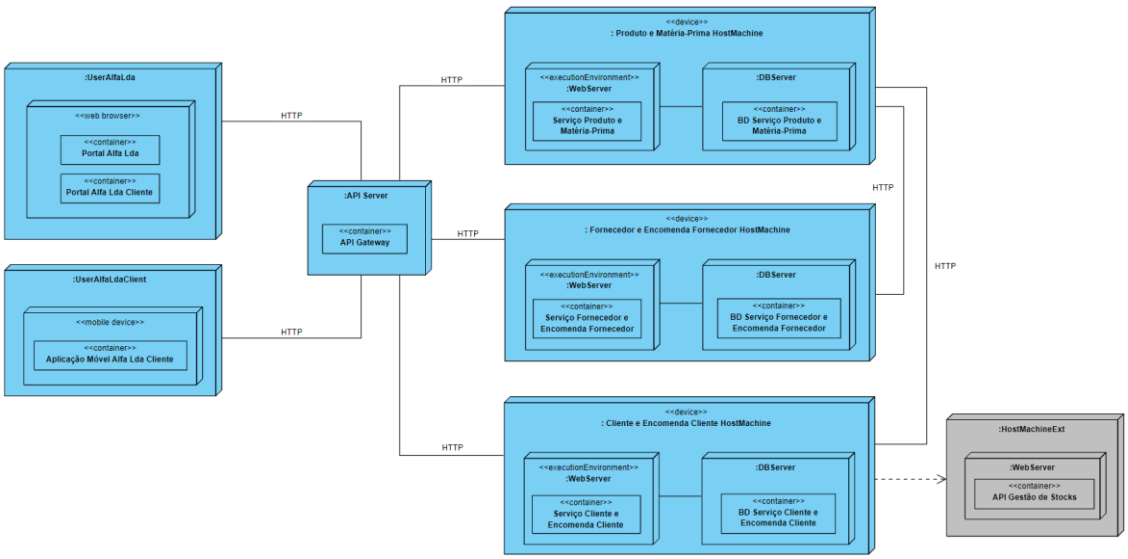


Figura 40 Vista de implantação do caso de estudo 2

Tal como se pode verificar no diagrama apresentado este modelo físico está dividido em sete nós. Os nós *UserAlfaLda* e *UserAlfaLdaClient* representam a instalação das aplicações web acedidas via browser e a instalação da aplicação móvel acedida através de um dispositivo móvel, respetivamente. Estes comunicam, via HTTP, com o nó API Server que representa a instalação da API Gateway. Por sua vez, este nó comunica via HTTP com os nós *Produto e Matéria-Prima HostMachine*, *Fornecedor e Encomenda Fornecedor HostMachine* e *Cliente e Encomenda Cliente HostMachine* que representam a instalação dos serviços do sistema. Externamente ao sistema, considera-se nó *HostMachineExt* que representa a instalação da API de gestão de stocks que comunica com o nó *Cliente e Encomenda Cliente HostMachine*.

Esta vista permite que se analise e compare o processo de implantação do sistema, seguindo esta abordagem, nas plataformas em estudo. Espera-se que, a este nível, as plataformas possam apresentar algumas limitações ao nível das licenças utilizadas, uma vez que podem considerar subscrições/contratos no quais se determinam, por exemplo, o número de servidores por aplicação.

5 Implementação

Este capítulo apresenta a descrição da implementação dos casos de estudo descritos no capítulo 4 Casos de estudo nas plataformas em estudo.

Numa fase inicial, uma vez que a autora deste projeto de dissertação não tinha conhecimento e experiência prévia com as plataformas, foi necessário recorrer à documentação e outros recursos disponíveis, como cursos breves por temas/tópicos da plataforma. Para além disso, ao longo deste desenvolvimento consultaram-se, também, os fóruns das plataformas. Importa realçar, também, que a licença gratuita foi utilizada em ambas as plataformas, pelo que o seu uso pode apresentar algumas limitações ao nível das funcionalidades fornecidas.

Nos subcapítulos apresentados, de seguida, estão organizados por plataforma e descrevem os principais conceitos da plataforma e os casos de estudo 1 e 2. A descrição dos principais conceitos das plataformas foi considerada para facilitar o entendimento do desenvolvimento das soluções. Para a descrição dos casos de estudo considera-se a análise da construção do requisito funcional UC1: Criar Encomenda (cf. 4.1.2 Requisitos Funcionais), uma vez que se trata de um requisito mais significativo ao nível da sua complexidade de lógica de negócio e por este ser desenvolvido em aplicação web e aplicação móvel.

Para o caso de estudo 2 recorreu-se à *Amazon API Gateway* (Amazon, 2021) durante o seu período gratuito. Durante esse período foi possível verificar que as APIs expostas em ambas as plataformas permitem a integração com esta API *gateway*, desde que estejam publicadas na *cloud* (no caso da plataforma Mendix). Depois deste período considerou-se a comunicação direta dos serviços com as aplicações finais.

5.1 Outsystems

Este subcapítulo apresenta a descrição do desenvolvimento dos casos de estudo no *Service Studio*.

5.1.1 Visão geral dos principais conceitos da plataforma

A (Outsystems, 2021b) possibilita o desenvolvimento de diferentes tipos de aplicação – *Reactive Web App*, *Table App*, *Phone App*, *Tradicional Web* e *Service*. Os primeiros quatro tipos fornecem

experiências de interfaces responsivas e nativas para *tablet* ou *smartphones*. Por outro lado, aplicações do tipo *Service* permitem centralizar e isolar lógica e dados de negócio que podem ser partilhadas por diferentes aplicações. As aplicações apresentam uma estrutura pré-definida, composta pelos componentes *Data*, *Logic*, *Processes* e *Interface*. Esta estrutura é transversal a todos os tipos de aplicação, à exceção das aplicações do tipo *Service*, que não apresentam Interface.

A modelação de dados na base de dados realiza-se no componente *Data*. A (Outsystems, 2021b) considera que entidades correspondem aos elementos que permitem persistir informação e implementar um modelo de base de dados. Por outro lado, é possível definir *Structures* que, tal como as entidades apresentam atributos, no entanto, estes dados não são persistidos na base de dados. Por omissão todas as entidades são criadas com o atributo Id, correspondente à chave primária (em Outsystems corresponde a *Entity Identifier*) e com as ações CRUD para a base de dados. Os atributos de uma entidade são representados por tipos de dados considerados pela plataforma, tais como: *Binary data*, *Boolean*, *Date*, *Email*, *<Entity> Identifier*, entre outros. A plataforma considera ainda a definição de entidades estáticas, que, para além de atributos apresentam *Records* que correspondem a um conjunto de valores fixos caracterizados por um nome e uma ordem. As relações entre as entidades podem ser *One-To-One*, *One-To-Many* e *Many-To-Many*. Para definir estas relações recorre-se a um *reference atribute*, que corresponde ao conceito de chave estrangeira, tradicionalmente conhecido (Outsystems, 2021b). A definição destas entidades resulta na criação das suas tabelas na base de dados, podendo estas ser consultadas diretamente na plataforma. Para além disso, é possível visualizar graficamente as entidades definidas e as relações entre elas, através da definição de um diagrama de entidades, obtido pelo *drag-and-drop* de entidades.

A lógica de negócio é implementada no componente *Logic*, através de *server actions* que correspondem a ações que executam a lógica no lado do servidor. Estas apresentam propriedades, tais como (Outsystems, 2021b): (i) *Public* que determina se a ação está disponível para ser adicionada como dependência num módulo e (ii) *Function* que permite definir a ação como uma função, sendo que esta deve retornar um valor. Estas ações correspondem a fluxos compostos por várias atividades e podem apresentar variáveis locais, *input* ou *output*, utilizadas para tratar, receber e enviar informação. As atividades destas ações podem ser: chamadas a outras ações do servidor, agregados para obter informação da base de dados, condições de fluxo, atribuições, (des)serialização de objetos e exceções. O componente de lógica

disponibiliza também a funcionalidade de integrar aplicações com outros sistemas, através de integrações com serviços SOAP, REST, SAP ou outros sistemas externos, sendo estes últimos integrados através do *Integration Builder* que suporta, por exemplo, integrações com *Salesforce* e *Microsoft Dynamics 365* (Outsystems, 2021b).

O componente *processes* considerado nas aplicações desta ferramenta permite a gestão de processos empresarias (BPT) e a sua integração com as aplicações (Outsystems, 2021b). Para a implementação destes processos, a ferramenta disponibiliza um conjunto de ações que correspondem a atividades realizadas pelo utilizador final, atividades automáticas, chamada a outros processos, decisões ou ações de espera.

A implementação de interfaces gráficas é definida no componente Interface. As aplicações são compostas por conjuntos de *Screens* que apresentam a interface UI, resultante da composição de vários elementos gráficos. As aplicações apresentam, também, temas, imagens e scripts que auxiliam o desenvolvimento destas interfaces. Para a sua implementação, considera-se a utilização de variáveis locais ou variáveis *input*, para tratar ou receber informação e de *client actions*, que permitem a implementação de lógica no lado do cliente. À semelhança das ações do servidor, estas podem apresentar variáveis locais, *input* e *output* e o seu fluxo pode apresentar chamadas a outras ações de cliente ou servidor, decisões, atualização e consulta de informação, atribuição, exceções, ações para retornar para outro ecrã, mensagens para o utilizador e implementação de código em *javascript*. Em termos de UI a plataforma disponibiliza componentes como tabelas, listas, botões, gráficos, ícones, menus, entre outros, que podem ser adicionados aos ecrãs.

5.1.2 Caso de Estudo 1

Tal como descrito em 4.1 Caso de estudo 1 para este caso de estudo pretendia-se seguir uma abordagem de arquitetura monolítica, combinada com arquitetura em camadas. De forma a orientar a solução para uma abordagem de arquitetura em camadas, teve-se em conta a composição das aplicações, por defeito, de Outsystems. Esta composição corresponde a uma divisão em componentes que, por sua vez correspondem às camadas consideradas na arquitetura definida.

Para esta solução criaram-se duas aplicações, uma composta pelo módulo para implementação de interfaces para os colaboradores e pelo módulo do tipo *service* para implementação de

modelação de dados e lógica do sistema. A outra é composta pelo módulo para implementação de interfaces para os clientes. Esta implementação poderia considerar uma única aplicação, no entanto optou-se por separar as aplicações, uma vez que se verificou que a implementação de um segundo módulo para implementação de interfaces iria herdar os componentes do primeiro. Para além disso, a implementação do módulo do tipo *service* foi considerada para que, através da gestão de dependências entre módulos as aplicações finais utilizassem os recursos implementados.

A implementação do requisito em análise iniciou-se com a modelação das entidades que compõe o problema. Inicialmente, foi necessário considerar a possibilidade de associar a entidade cliente com uma entidade, já existente, para autenticação, uma vez que se pretendia que o cliente se pudesse autenticar para realizar uma encomenda. Por defeito, a plataforma apresenta a entidade *User*, que representa os utilizadores da aplicação. Desta forma, definiu-se a relação da entidade *Customer* com a entidade *User*.

A Figura 41 apresenta as entidades *CustomerOrder* e *CustomerOrderItem* implementadas, bem como os seus atributos e as ações da base de dados.

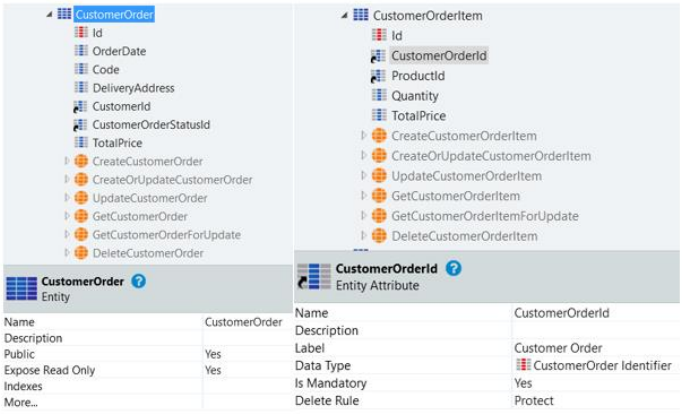


Figura 41 Entidades *CustomerOrder* e *CustomerOrderItem*

As ações CRUD sobre a base de dados são geradas automaticamente quando se cria uma nova entidade. Os atributos foram adicionados manualmente, sendo que, a este nível, a plataforma permite que se selecione o tipo do atributo manualmente ou este é determinado através do seu nome, uma vez que a plataforma deteta o nome e define um tipo de dados. Por exemplo, o atributo *Price* é definido, automaticamente, com o tipo *Currency*. Esta ação da plataforma permitiu facilitar, ao longo do desenvolvimento, a definição de atributos e variáveis. As relações

entre as entidades foram definidas recorrendo ao uso de *reference entity* que, no exemplo ilustrado na Figura 41 corresponde ao *CustomerOrderId* na entidade *CustomerOrderItem*, uma vez que a entidade *CustomerOrder* apresenta uma relação de um para muitos com *CustomerOrderItem*. Uma vez que o estado das encomendas dos clientes apresenta valores fixos, implementou-se uma entidade estática. Esta entidade permite representar o que, tradicionalmente, se define como enumerável. Foram adicionados *Records* com a designação de cada estado, sendo a sua ordem de apresentação definida automaticamente, podendo depois ser editada.

Pode verificar-se que a base de dados desta plataforma apresenta um comportamento de base de dados relacional, seguindo a abordagem do modelo ER (*Entity-Relationship*), uma vez que considera tipos de entidades e relações entre elas que são mapeadas em tabelas no modelo relacional. Ao nível da camada de acesso a dados prevista na arquitetura do sistema verifica-se, tal como se esperava, que a plataforma apresenta um nível de abstração considerável. Isto porque, as ações da base de dados são geradas automaticamente e não podem ser manipuladas, sendo assim a plataforma abstrai os desenvolvedores dos detalhes e implementação de ações/comandos na base de dados.

Para a lógica desta solução implementaram-se ações de servidor que invocam as respetivas ações à base de dados de forma a obter, criar, editar ou eliminar os objetos. Apesar disso, foi necessário implementar lógica adicional, por exemplo, para gerar o código da encomenda ou para criar a encomenda do cliente. Para a criação de uma nova encomenda pelo cliente, implementou-se a lógica para associar a encomenda criada a todos os elementos da lista de itens da encomenda. Isto acontece pela relação de um para muitos que *CustomerOrder* e *CustomerOrderItem* apresentam. A Figura 42 apresenta a ação que implementa a lógica do cenário descrito e os respetivos parâmetros de entrada.

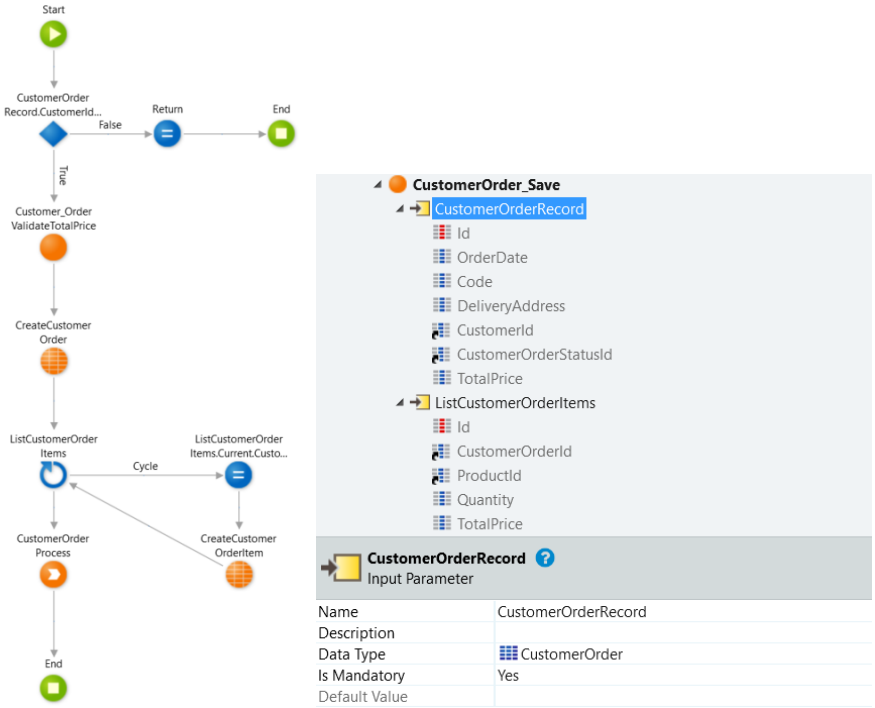


Figura 42 Ação do servidor para criar encomenda de cliente e parâmetros de entrada

Os parâmetros de entrada correspondem a um registo de encomenda e uma lista de itens da encomenda. Estes são preenchidos quando esta ação do servidor é chamada na ação de cliente, implementada na interface da aplicação. A ação começa por validar os parâmetros de entrada, garantindo que estes não são nulos. Caso isso não se verifique é enviada o retorno da ação e o fluxo termina. Caso contrário, o fluxo segue com a validação do preço total da encomenda. Segue-se a ação para criar a nova encomenda na base de dados. De seguida, com recurso à ação *ForEach* percorre-se a lista de itens da encomenda sendo que, em cada iteração é atribuído o Id da encomenda criada ao *CustomerOrderId* apresentado em *CustomerOrderItem* que, posteriormente é criado na ação da base de dados.

Por fim, é invocado o processo *CustomerOrderProcess*, cuja implementação é apresentada na Figura 43. Inicialmente, considerou-se a implementação apenas através de ações de servidor. No entanto, pelas ações e validações a considerar e, tendo em conta a funcionalidade disponibilizada pela plataforma, que permite executar processos em eventos de criação de objetos na base de dados, considerou-se oportuno entender e desenvolver o pretendido num contexto diferente das restantes ações implementadas.

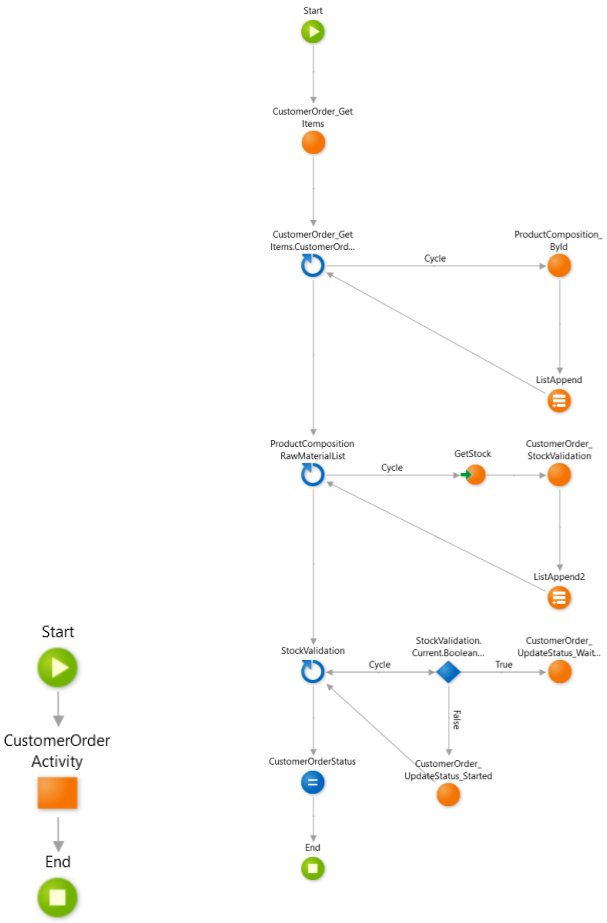


Figura 43 Fluxo do processo *CustomerOrderProcess*

De facto, os processos podem ser executados a partir de um evento da base de dados. Contudo, neste contexto, a criação da encomenda de cliente foi seguida de outras ações, que permitiram associar a encomenda aos itens. Sendo assim, não foi possível habilitar o processo para ser inicializado no evento *CreateCustomerOrder*, uma vez que nesta fase apenas é criada a encomenda. A atividade deste processo inicia-se com a ação para obter a informação da encomenda pelo seu Id, seguindo-se a ação para obter os respetivos itens da encomenda. Depois, percorre-se um *Foreach* que itera sobre a lista dos itens da encomenda, para se obter a composição dos produtos encomendados. Através da ação de sistema *ListAppend* retorna-se uma lista de composição do(s) produto(s), que é utilizada na ação seguinte de forma a verificar se existe pelo menos um produto cujo stock de matéria-prima seja insuficiente. Após esta validação, o estado da encomenda é atualizado conforme a disponibilidade de stock.

Para a validação de stock implementou-se uma *Rest Api* para consumo. Para tal, configurou-se o método da API a consumir, através da definição do URL do pedido, bem como o seu tipo e

Body (neste caso, um exemplo da resposta do pedido). Quando se cria um pedido é criada, automaticamente, uma estrutura de dados que recebe a sua resposta, fazendo o respetivo mapeamento de *json* para a estrutura, através da criação de atributos. A Figura 44 apresenta os detalhes da resposta, representada num parâmetro de saída, do pedido e a respetiva estrutura criada.

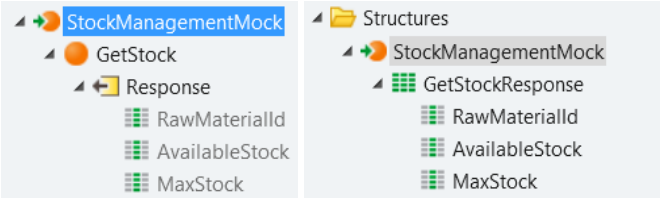


Figura 44 Detalhes da resposta do pedido API e respetiva estrutura

Na implementação de interfaces gráficas, optou-se por recorrer à criação *From Scratch*, partindo de uma tela branca, mas que já apresenta, por defeito, página de login e esquemas de menus definidos. A este nível é possível, através de *roles*, definir as permissões dos utilizadores aos ecrãs. Para o requisito em análise assegurou-se que os ecrãs estão autorizados para utilizadores com o *role* de cliente. Adicionalmente, é possível validar o papel de utilizador nas ações de servidor, através de ações fornecidas pela plataforma.

Para a implementação do formulário de criar encomenda de cliente, recorreu-se à opção da ferramenta que permite o *drag-and-drop* da entidade para o *MainFlow*. A partir desta ação foram gerados, automaticamente, dois ecrãs, um com o formulário para criar encomenda e outro com a listagem das encomendas. Esta ação permitiu simplificar esta implementação, uma vez que, ao associar-se uma entidade ao formulário este passa a considerar os seus atributos, mapeamento os seus tipos para elementos do formulário. Apesar disso, foi necessário implementar a lógica das ações de cliente invocadas nos eventos dos botões do formulário. No formulário para criar uma nova encomenda considerou-se a composição de encomenda, permitindo que na mesma página o utilizador adicione os itens à encomenda. Desta forma, adicionaram-se dois *Forms* na mesma página, um para inserir detalhes da encomenda e outro para inserir a composição da encomenda e uma *List* para, visualmente, ser possível verificar os produtos já adicionados. A Figura 45 apresenta o formulário para a criação de uma encomenda.

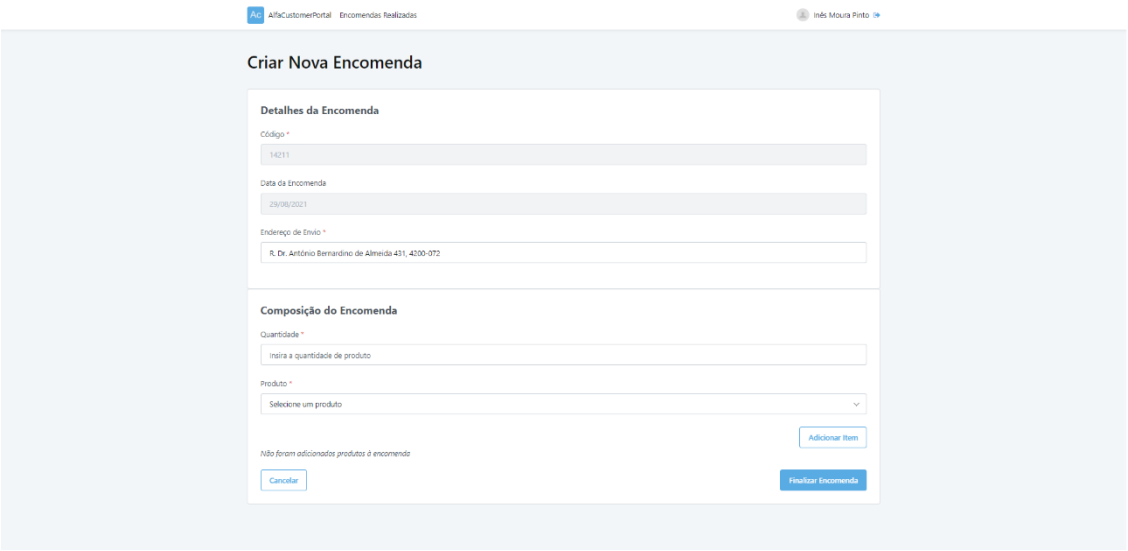


Figura 45 Formulário para criar nova encomenda

A este nível, implementaram-se ações de cliente que recebem informação do formulário e comunicam com as ações de servidor. Estas ações têm início no botão para criação de uma nova encomenda, apresentado na página das encomendas realizadas. Nesse momento atribuem-se os valores (e.g. código) a passar por parâmetro para o formulário. Ao nível do formulário recorreu-se à ação *aggregate* para, através de *data fetching*, ser possível obter informação, diretamente do servidor. Esta ação permitiu obter e listar todos os produtos na *dropdown* do formulário. Na página apresentada associaram ações de cliente aos botões Adicionar Produto e Finalizar Encomenda. Na primeira ação validam-se os dados do formulário, para garantir que o utilizador insere uma quantidade e seleciona um produto.

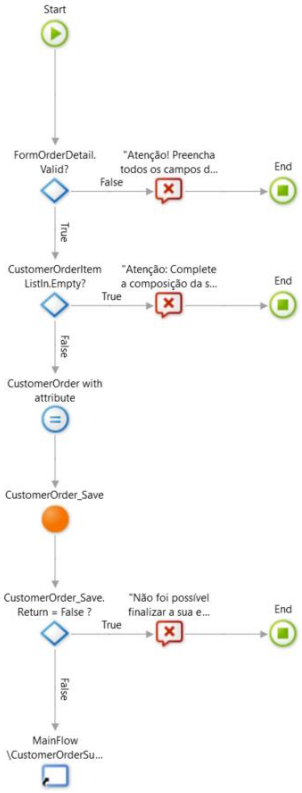


Figura 46 Ação do cliente para criar encomenda

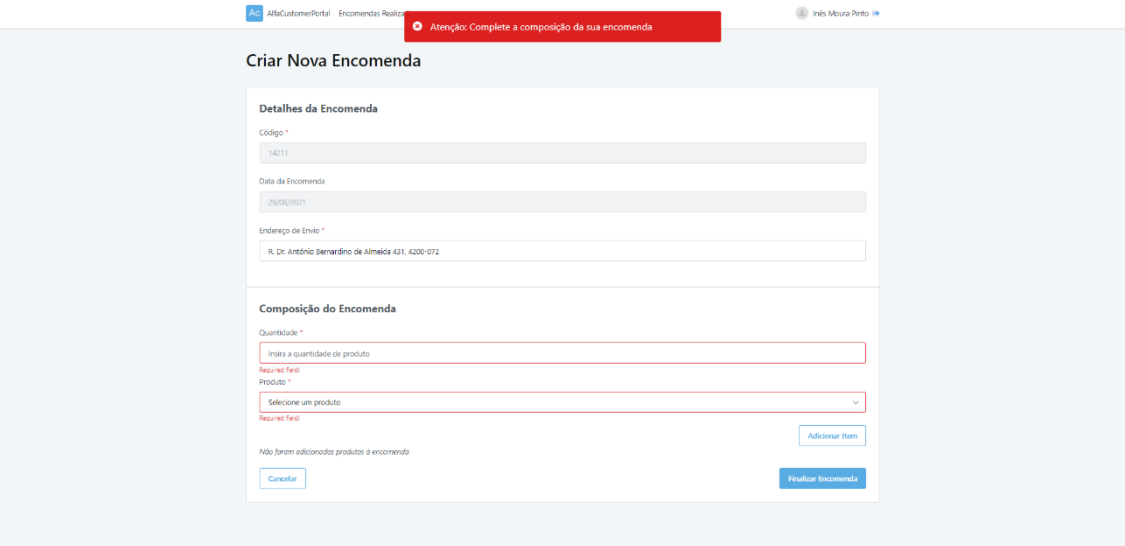


Figura 47 Mensagem de erro apresentada quando a lista de itens está vazia

A ação para finalizar a encomenda (ver Figura 46) recebe como parâmetro a lista de todos os itens adicionados à encomenda e inicia o seu fluxo com a validação dos dados do formulário. No caso destes não serem válidos apresenta-se uma mensagem ao utilizador. Caso contrário verifica-se se a lista de itens está vazia e, em caso afirmativo é apresentada uma mensagem de

erro (ver Figura 47). Caso contrário, mapeiam-se os valores das variáveis do formulário para os atributos de *CustomerOrder* (ver Figura 42). Esta entidade é enviada para a ação do servidor descrita previamente. Por fim, se o retorno da ação do servidor for *false*, retorna-se uma mensagem de insucesso ao utilizador, senão apresenta-se uma nova página que apresenta uma mensagem de sucesso.

Nesta solução implementaram-se testes de qualidade recorrendo ao componente *BDDFramework* disponibilizado pela plataforma. Este permite a automação de testes através da especificação da sintaxe *Gherkin*, que se direciona para profissionais de qualidade e desenvolvedores. Desta forma, foi necessário instalar o componente no ambiente de desenvolvimento e definir cenários através da sintaxe considerada. Para além disso, criou-se uma nova aplicação destinada a testes, tendo sido este aspeto considerado por orientação da Outsystems. Verificou-se que, para ser possível utilizar todo o material do componente de testes foi necessário criar uma aplicação do tipo *Tradicional Web* que, através de gestão de dependências, acede ao componente de testes e aos componentes desenvolvidos para a solução do caso de estudo. Para o requisito em estudo definiu-se o cenário que prevê que quando um cliente adiciona um item à encomenda a lista de itens desta é atualizada corretamente. Para este cenário adicionaram-se as ações *BDDScenario* e *BDDStep* na página da nova aplicação (ver Figura 48) e, consequentemente, definiram-se as ações para cada *BDDStep*. Neste caso pretende-se testar a ação implementada que adiciona um item à lista de itens da encomenda e, para tal, foi necessário recorrer às ações já implementadas, bem como às ações como *AssertTrue*, disponibilizadas pelo componente de testes, para validar a condição válida para o teste. O fluxo desta ação implementada para o passo *Then* apresenta-se na Figura 49. O resultado do teste é apresentado na página da aplicação e apresenta, visualmente, o resultado do teste.

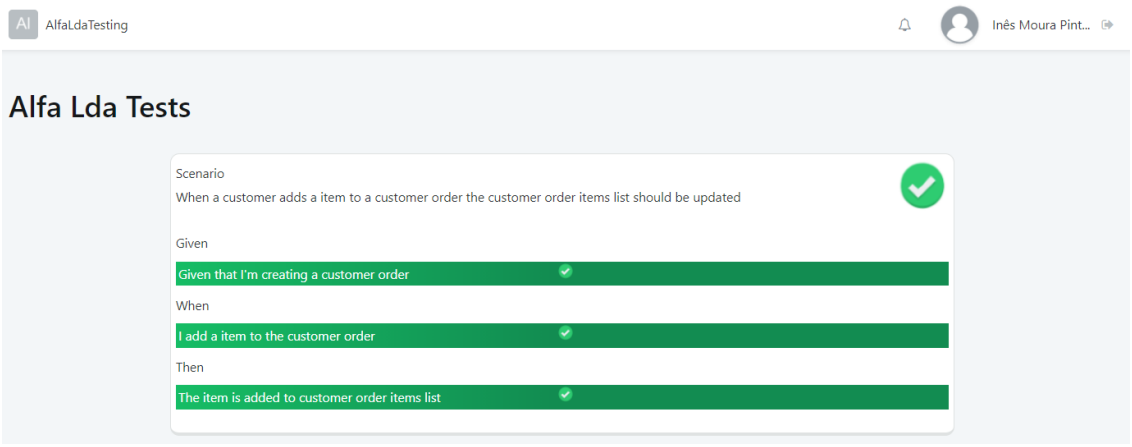


Figura 48 Interface do teste para verificar lista de itens recorrendo a BDDFramework

Figura 49 Fluxo da ação para o passo *Then*

Nesta plataforma foi possível seguir a estrutura que esta disponibiliza nas suas aplicações o que permitiu, com relativa facilidade, que se identificassem as camadas da arquitetura. Verificou-se que cada componente desta solução apresenta uma responsabilidade específica. Para além disso, ao nível da comunicação entre os componentes verificou-se que todos eles comunicam com o componente imediatamente inferior. Contudo, foi possível identificar que ao nível do componente de apresentação a implementação de *data fetching* pode comprometer esta comunicação, uma vez que esta ação obtém informação diretamente da base de dados. Por

outro lado, esta ação permitiu reduzir a utilização de variáveis e chamadas às ações do servidor. Para além disso, ao longo desta implementação foram utilizadas, ao nível das variáveis das ações de servidor e cliente, as entidades de domínio. Com esta abordagem os componentes de lógica e apresentação acedem diretamente a essas entidades. Em alternativa podia-se considerar a definição de estruturas que, uma vez que não são persistidas na base de dados, auxiliavam a transferência de informação entre as camadas da aplicação, evitando a exposição das entidades de domínio. O mapeamento entre as estruturas e as entidades seria feito na ação de atribuição, que teria que ser repetida sempre que fosse necessário mapear os dados. Esta abordagem seria semelhante ao, tipicamente conhecido, padrão DTO (*Data Transfer Object*).

Ao nível da arquitetura alternativa descrita em 4.1.4 Arquitetura, a adoção dessa arquitetura passaria por considerar uma implementação semelhante à descrita, em termos de estrutura da solução. Contudo, seria necessário considerar a publicação de uma *Rest Api* na aplicação do tipo *Service* e, por outro lado, o consumo desta ao nível das aplicações finais. A definição inicial da estrutura da solução implementada favorecia a adaptação para a arquitetura alternativa. De facto, a implementação de um *service* permitiu que se isolasse a lógica de negócio e base de dados, sendo que a adaptação deste módulo para disponibilizar uma API seria mais rápida e direta. Caso contrário, se a solução tivesse sido desenvolvida num único módulo, a sua decomposição para a arquitetura alternativa obrigava à implementação de um módulo de serviço e à migração de toda a lógica e base de dados da solução.

Em 4.1.4 Arquitetura apresentou-se, também, as abordagens a serem consideradas e analisadas ao nível da implantação do sistema. Nesta plataforma, a implementação de aplicações do tipo *Reactive Web App* acabou por determinar a abordagem seguida. Isto porque, este tipo de aplicação disponibiliza uma abordagem mais moderna, sendo esta implantada como SPA. Com esta abordagem a plataforma pretende que, contrariamente a aplicações renderizadas no lado do servidor, a reação a qualquer interação do utilizador seja mais rápida (Outsystems, 2021c). A escolha deste tipo de aplicação foi baseada na orientação da plataforma, que disponibiliza este tipo mais moderno e com mais capacidades em relação a outros tipos de aplicação. Isto porque, a plataforma ainda permite a implementação de aplicações renderizadas no lado do servidor (SSR) – *Tradicional Web* – sendo este o tipo de aplicação mais antigo da plataforma.

5.1.3 Caso de Estudo 2

Tal como descrito em 4.2 Caso de estudo 2 para este caso de estudo pretendia-se seguir uma abordagem de arquitetura orientada a microserviços, tendo como ponto de partida a solução implementada para o caso de estudo 1. Através de *copy-paste*, reaproveitaram-se as ações e componentes definidos na solução anterior. Para tal, criaram-se aplicações compostas pelo módulo do tipo *Service* destinadas à implementação de cada microserviço. Em cada módulo definiram-se as entidades, a lógica de negócio e publicou-se a API do serviço. Criou-se, também, uma aplicação móvel para implementação de interfaces e lógica de apresentação, que permite a comunicação com os serviços definidos. Ao nível das aplicações web existentes, estas foram mantidas, mas, no contexto deste caso de estudo, foi necessário adicionar interfaces (duplicadas do caso de estudo anterior) e lógica para consumir os pedidos.

A implementação do requisito em análise iniciou-se com a definição das entidades necessárias para o Serviço CEC. Em relação ao caso de estudo anterior, estas entidades mantiveram os seus atributos, tendo apenas sido necessário alterar o tipo de *ProductID* para *Long Integer*, uma vez que deixa de se representar a relação entre *CustomerOrderItem* entidade com *Product*.

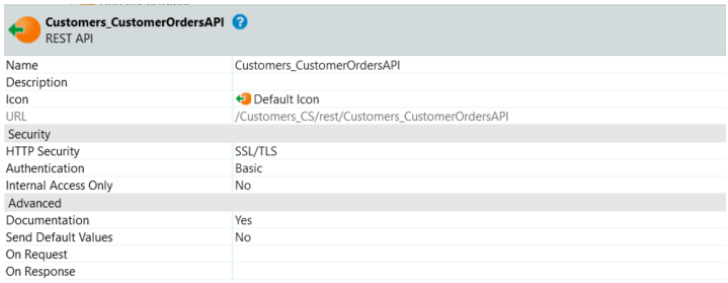


Figura 50 Propriedades de *Customers_CustomerOrdersAPI*

A principal particularidade deste serviço está ao nível da exposição de uma *Rest* API. Neste sentido, implementou-se a *Customers_CustomerOrdersAPI* (ver Figura 50), configurando a sua autenticação como *Basic*. Quando a autenticação foi alterada, automaticamente, foi adicionada a ação *OnAuthentication*, correspondente a uma *Rest Api Callback*, sendo executada antes de qualquer método da API. Para além disso, adicionou-se uma ação disponibilizada pelo sistema para garantir apenas a autenticação de utilizadores com o *role* de cliente. Adicionalmente, consideram-se outras propriedades da API como *HTTP Security* (SSL/TLS ou None) ou *Documentation*. Esta propriedade, por defeito definida como *Yes*, permite que seja

disponibilizada documentação a partir das propriedades, dos métodos e dos parâmetros da API, tal como se pode verificar na Figura 51 e Figura 52.

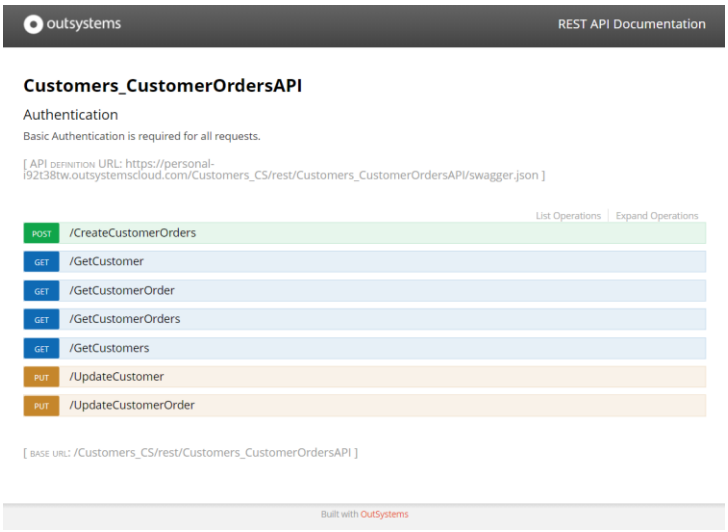


Figura 51 Documentação de *Customers_CustomerOrdersAPI*

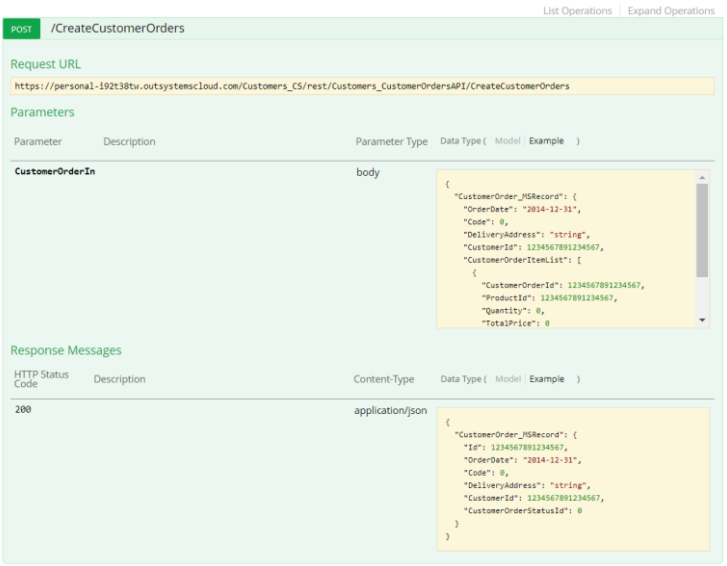


Figura 52 Detalhes da documentação do método POST

A exposição de uma API requer a definição de métodos *Rest* e, para tal, no âmbito do requisito em análise, adicionou-se o método POST para criar uma encomenda de cliente. O tipo do método é definido nas propriedades, assim como o URL *path,* que pode ser editado, considerando que este é utilizado a seguir ao URL do *endpoint*, sendo este definido pela plataforma. Este método apresenta um parâmetro de entrada que deve ser recebido no *body* do pedido. Este parâmetro é composto por uma estrutura de *CustomerOrder* que apresenta

uma lista de estrutura de *CustomerOrderItem*, sendo que estas apresentam apenas os atributos necessários para o pedido. Para além disso, este método apresenta uma variável local composta pela entidade *CustomerOrder* e uma lista da entidade *CustomerOrderItem* utilizada para o mapeamento de estrutura para entidade, e um parâmetro de saída que é composto pelas mesmas estruturas definidas anteriormente. O fluxo do método implementado (ver Figura 53) é semelhante ao fluxo de criar encomenda implementado em 5.1.2 Caso de Estudo 1, tendo sido necessário alterar alguns aspetos, tais como:

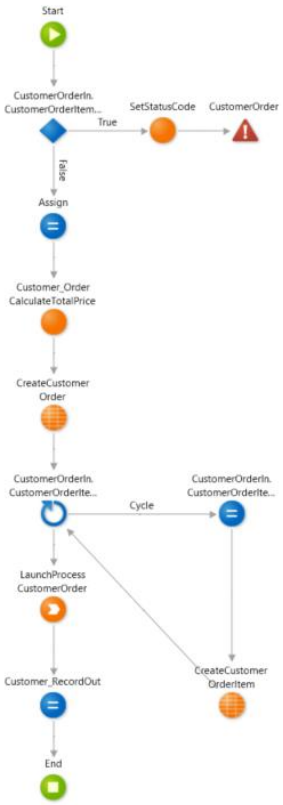


Figura 53 Método POST para criar encomenda de cliente

- A validação inicial para garantir que os dados recebidos não são nulos é, agora, seguida da ação *SetStatusCode*, obtida através da extensão *HTTPRequestHandler*, que altera o estado do pedido para 400, seguindo-se uma exceção que exibe uma mensagem de erro. Considerou-se a alteração do estado do pedido, uma vez que, por defeito, é definido o *Status* 500 quando é invocada uma exceção. Adicionalmente, a plataforma possibilita que sejam apresentados mais detalhes na resposta do pedido, através da propriedade *OnResponse*;

- Mapeamento dos valores das estruturas para entidades, através da ação de atribuição (ver Figura 54). Estas estruturas recebem informação que é mapeada para as entidades da base de dados. Neste caso, esta abordagem foi considerada para que, ao nível da interface, fossem utilizados apenas os atributos necessários;

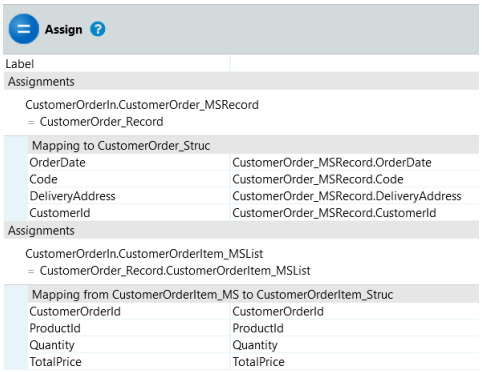


Figura 54 Mapeamento de estrutura para entidade

- A ação para calcular o preço da encomenda e o processo para validar o stock e atualizar o estado da encomenda mantiveram-se no fluxo, mas foram alterados, de forma a ser possível obter os preços dos produtos e a sua composição a partir de um pedido ao Serviço PMP.

Tal como referido neste último aspeto, foi necessário consumir o serviço de produtos para se obter a informação necessária para a criação da encomenda. Ao nível do Serviço de Produto e Matéria-Prima, apesar de se implementar o método para obter produto por id, considerou-se oportuno implementar um método cuja resposta apresenta-se apenas a quantidade e o código de matéria-prima. A Figura 55 apresenta o fluxo deste método., que se inicia com a ação *aggregate* que permite obter dados da base de dados e filtrá-los, sendo que, neste caso, estes foram filtrados pelo id do produto. Depois, é invocada a ação da base de dados para se obter a matéria-prima a partir do seu id, para ser possível mapear o código de matéria-prima e a respetiva quantidade para a resposta.

Figura 55 Método Get para obter composição de produto

Assim, ao nível do Serviço CEC foi necessário consumir estes métodos. Neste caso, a plataforma permite que se consumam todos ou apenas alguns métodos de um serviço, sendo que, em qualquer um dos casos são criadas, automaticamente, estruturas para as respostas dos métodos. Estas estruturas e o formato das respostas permitiram simplificar a adaptação dos fluxos da ação e do processo, uma vez que foi necessário apenas incluir a chamada ao método e atualizar as ações que usavam a sua resposta. Assim, o processo para validar o stock e atualizar o estado da encomenda apresentado na Figura 56, foi alterado, de forma a considerar o pedido da composição de produto descrito anteriormente.

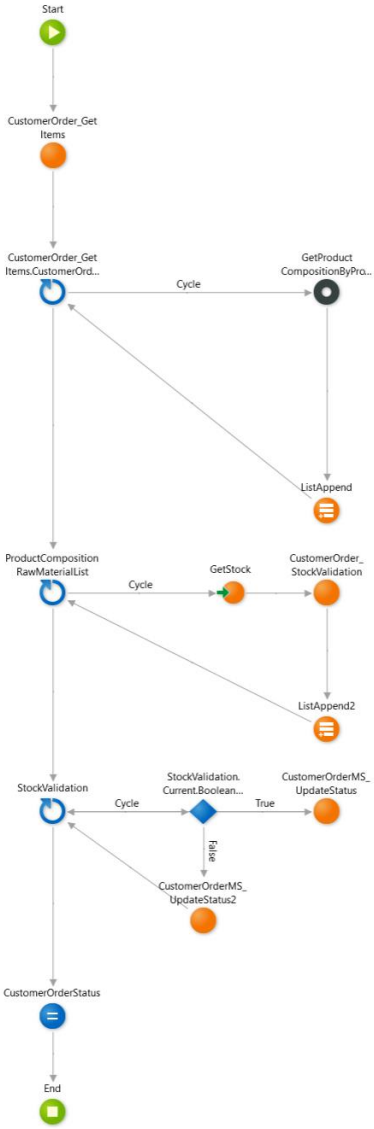


Figura 56 Fluxo do processo *CustomerOrderProcess*

Ao nível da aplicação móvel implementou-se uma aplicação do tipo *Phone App* sendo que esta apresenta uma estrutura semelhante à estrutura da aplicação considerada em 5.1.2 Caso de Estudo 1, tanto ao nível dos componentes gráficos como das ações de cliente. No âmbito do requisito em análise, implementou-se o ecrã apresentado na Figura 57.

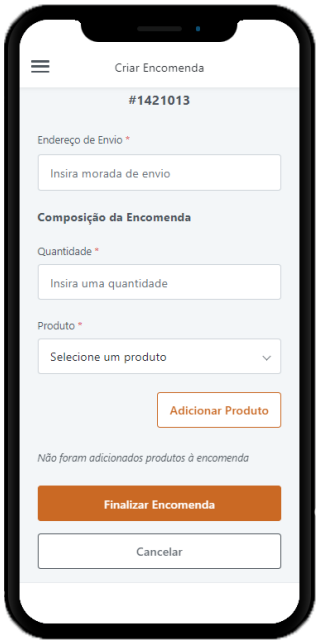


Figura 57 Formulário para criar encomenda na aplicação móvel

Uma vez que se pretendia que esta aplicação consumisse, via API *gateway*, os pedidos, adicionaram-se os métodos para consumo na aplicação. Esta ação foi semelhante à ação descrita em 5.1.2 Caso de Estudo 1 para consumir um método *Rest*. Durante este processo foi necessário validar alguns tipos de dados das estruturas criadas, uma vez que, como estas são geradas automaticamente nem sempre são mapeadas para o tipo que se espera. Para o formulário de criar encomenda, uma vez que a composição da encomenda pressupõe a apresentação dos produtos, adicionou-se a ação *Fetch Data From Other Sources* para chamar o método *Rest* e obter os produtos, de forma a retornar uma lista que será passada nas propriedades da *dropdown list* (ver Figura 58).

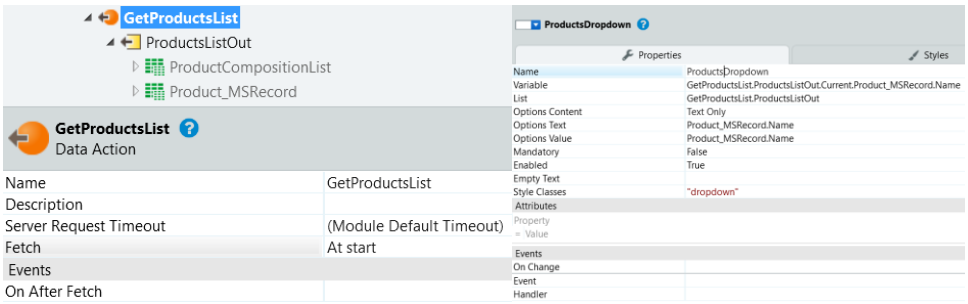


Figura 58 *Data fetch* do pedido *GetProducts* e propriedades da *dropdown list* de produtos

Para adicionar um produto à encomenda implementou-se uma ação de cliente semelhante à implementada no caso de estudo anterior. Esta ação é realizada localmente, apenas para preencher uma lista que, mais tarde, é enviada para o pedido responsável por criar encomenda. Para finalizar a encomenda implementou-se a ação cujo fluxo está apresentado na Figura 59, que se inicia com a ação de atribuição que mapeia os atributos do formulário para a variável local, definida através de uma estrutura. Por sua vez, esta é enviada, por parâmetro, para o método de criar encomenda, descrito anteriormente (ver Figura 53). Por fim, uma vez que não se verificou uma ação que permitisse verificar o estado do pedido, optou-se por verificar o conteúdo da resposta e, dependendo desta, retornar uma mensagem de sucesso ou insucesso.

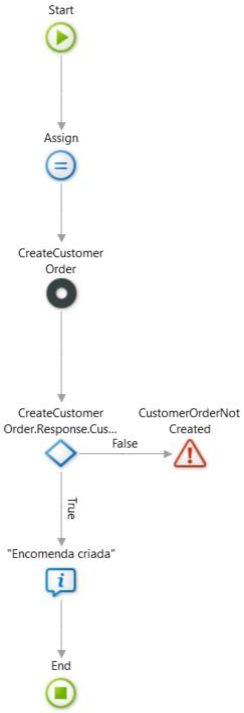


Figura 59 Ação de cliente para criar encomenda

Na fase final desta implementação implementaram-se testes de qualidade, tendo-se recorrido, mais uma vez, ao componente BDDFramework e à criação de uma nova aplicação para o efeito. Neste caso, este componente foi utilizado para testar as APIs publicadas tendo em conta um determinado cenário. Desta forma, seguiu-se uma abordagem de testes de integração. Para o requisito em estudo definiu-se um cenário para obter composição de um produto. Para tal, foi necessário consumir a API que realiza o pedido para obter composição de produto, tendo este sido solicitado na ação *When* e a sua resposta guardada numa variável local. Os valores desta

resposta são utilizados na ação *Then*, através das ações do tipo *AssertValue* que comparam o valor esperado com o valor obtido.

Durante a implementação da aplicação móvel não se detetaram diferenças significativas em relação à implementação das aplicações consideradas no caso de estudo anterior. Contudo, ao nível deste tipo de aplicação são disponibilizadas configurações de distribuição destas aplicações. Assim, as aplicações podem geradas tendo em conta as características nativas – Android ou IOS – ou podem ser distribuídas como PWA. Em ambos os casos, é gerado um QR *Code* que pode ser lido através de um dispositivo móvel. Em alternativa, é disponibilizado um *url* para download do package da aplicação ou para abrir a aplicação num browser, em configurações nativas ou de distribuição como PWA, respetivamente.

Ao nível da arquitetura de aplicações, esta plataforma sugere a conceção de soluções baseada em serviços SOA (*Service-Oriented Architecture*) e, para tal, disponibiliza módulos e ferramentas (p.e. *The Architecture Canvas*) que permitem isolar os principais conceitos de um problema. Apesar desta abordagem já reforçar uma abordagem modular, ela não representa uma implementação orientada a microserviços, uma vez que todos os serviços estão no mesmo ambiente/instalação que as restantes aplicações. Na implementação desta solução foi possível implementar módulos que disponibilizassem uma API e, por outro lado módulos que consumissem os pedidos da API *gateway*. Apesar disso, verificou-se que estes módulos foram instalados no mesmo ambiente (*personal environment* disponibilizado para licenças gratuitas), sendo que este corresponde a uma única peça de uma infraestrutura em Outsystems. Neste sentido, na perspetiva de vista de implantação não foi possível definir e validar a instalação dos principais *containers* da solução em nós isolados. Apesar disso, esta plataforma aborda as principais considerações para a adoção de uma arquitetura orientada a microserviços, apresentando e comparando cenários de microserviços semi-isolados e isolados totalmente num único ambiente ou em ambientes separados, respetivamente. Neste caso, os ambientes e infraestruturas são disponibilizados tendo em conta o plano subscrito na plataforma. Ao nível da alternativa descrita em 4.2.4 Arquitetura verificou-se que esta plataforma disponibiliza conetores (e.g. *Connector Kafka*) que permitem a integração dos serviços com *message broker*. Contudo, verificou-se que a este nível, como os conectores podem ser desenvolvidos por desenvolvedores da plataforma, na sua maioria, não apresentam documentação que oriente sua integração com os serviços. Por esta razão, a este nível, não foram realizadas experiências para validar a adequabilidade destes conectores.

Na adaptação da arquitetura já existente para a arquitetura orientada a microserviços, apesar das diferenças significativas ao nível da estrutura da solução, foi possível adaptar e reaproveitar todos os componentes que já estavam implementados. Apesar disto ter facilitado o desenvolvimento, ao nível das interfaces deixou de se beneficiar de algumas capacidades que a plataforma disponibiliza para acelerar a implementação, como, por exemplo, a utilização de *aggregates* para obter os dados diretamente da base de dados. Para além disso, implementaram-se, adicionalmente, estruturas que requererem um mapeamento para entidades, aumentando, assim, a necessidade de implementar mais ações. Se esta solução tivesse sido implementada seguindo a alternativa descrita em 4.1.4 Arquitetura, a sua adaptação para esta arquitetura poderia ser mais intuitiva e rápida, uma vez que as aplicações finais já estariam aptas para receber e enviar pedidos, sendo apenas necessário adaptar esses pedidos e decompor o *back-end* em microserviços. Contudo e, tal como referido anteriormente, nesta abordagem não se beneficiaria das principais capacidades da plataforma, sobretudo ao nível das interfaces.

5.2 Mendix

5.2.1 Visão geral dos principais conceitos da plataforma

A plataforma (Mendix, 2021b) fornece o *Mendix Studio Pro* que corresponde a um IDE que permite o desenvolvimento de aplicações, com recurso a ações completas e que podem requerer um maior entendimento de programação, em relação ao *Mendix Studio* (cf. 3.6.4 Mendix).

Ao nível da implementação, no *Mendix Studio Pro* todas as aplicações apresentam a mesma estrutura, concentrada em *App Explorer,* composta por uma pasta destinada a definições de segurança, navegação e outras configurações relevantes e pelo módulo da aplicação. Cada módulo da aplicação apresenta o seu próprio *domain model*, que consiste na representação de entidades e nas suas relações, sendo estas representadas por associações (Mendix, 2021b). Cada entidade apresenta uma propriedade que permite determinar a persistência ou não na base de dados. Em caso afirmativo, é criada uma tabela de base de dados para a entidade, caso contrário a entidade é armazenada na memória em tempo de execução.

As entidades são representadas por atributos, caracterizados por um tipo de dado (e.g. *Decimal*, *Enumeration*, *Binary*, *AutoNumber*, entre outro) e que podem ter uma ou mais *validation rules* ou podem apresentar um *microflow* que calcula o valor para o atributo. As relações entre as entidades são definidas através de associações, que apresentam uma multiplicidade indicada pelo número 1 ou por *. Estas associações podem ser do tipo *One-to-Many*, *Many-To-Many* ou *One-To-One*. Para além disso, as entidades podem apresentar *generalization* (frequentemente designada herança), que permite que se relacionem com entidades do sistema (Mendix, 2021b).

Em Mendix a lógica de negócio de uma aplicação pode ser implementada a partir de *microflows*, e *nanoflows*. Os *microflows*, cuja notação gráfica é baseada no BPMN (Modelo e Notação de Processos de Negócios), permitem realizar ações como, por exemplo, criar e atualizar objetos ou mostrar páginas. À semelhança destes fluxos, os *nanoflows* permitem realizar ações para expressar a lógica de uma aplicação. No entanto, apresentam vantagens como o funcionamento diretamente no browser/dispositivo e podem ser utilizados numa aplicação *offline* (Mendix, 2021b). Apesar disso, não é recomendado realizar ações de base de dados neste tipo de fluxo. Os fluxos descritos podem apresentar parâmetros de entrada, sendo estes preenchidos no local a partir do qual o fluxo é desencadeado (Mendix, 2021b). Para além disso, estes fluxos apresentam um conjunto de atividades sequenciais que correspondem a ações a serem executadas, podendo ser consideradas condições, bem como ciclos para iterar sobre uma lista ou uma condição.

Esta plataforma permite a integração com outras aplicações, podendo esta ser feita utilizando OData, REST ou SOAP/Web Services. A plataforma pode importar ou exportar dados de XML e JSON. Para tal, são utilizados mapeamentos – *import mapping* e *export mapping* - que permitem que os objetos de Mendix possam ser convertidos de/ou para XML ou JSON, respetivamente, de acordo com um esquema XML ou uma estrutura JSON.

Relativamente às interfaces gráficas de aplicações, em Mendix o resultado é sempre uma SPA o que significa que toda a interação tem lugar numa única janela do navegador (Mendix, 2021b). Cada página é baseada num layout que determina a estrutura da página e num *template* que se baseia em elementos pré-definidos. As páginas podem estar ligadas entre si, sendo possível uma página abrir outra página, considerando, também, a possibilidade de implementar ações através de *click events*. Estes eventos podem realizar ações como mostrar uma página, chamar um *microflow* ou *nanoflow*, criar um objeto, fechar uma página, entre outros. A implementação

de novas páginas realiza-se a partir de um editor de páginas que pode apresentar o modo *design* ou *structure*, sendo possível, através de *drag-and-drop*, adicionar elementos, tais como *dropdown*, *text box*, *list view*, *data view*, imagens, gráficos, botões, menus, entre outros.

5.2.2 Caso de Estudo 1

Tal como descrito em 4.1 Caso de estudo 1 para este caso de estudo pretendia-se seguir uma abordagem de arquitetura monolítica, combinada com arquitetura em camadas. Tipicamente, a estrutura que a plataforma Mendix disponibiliza para o desenvolvimento de aplicações é adaptável para que seja possível implementar uma solução baseada numa arquitetura em camadas. Para tal, criou-se uma aplicação, sendo esta responsável por toda a implementação de modelação de entidades, lógica de negócio e interfaces gráficas. Toda a estrutura do projeto foi organizada por pastas, de forma a simplificar o desenvolvimento. Neste caso, não foi possível implementar as aplicações finais separadas, para cada papel de utilizador. Para tal, foi necessário considerar as permissões dos utilizadores ao nível das páginas da aplicação. Esta consideração deve-se ao facto da criação de novos módulos exigir a definição de um modelo de domínio. Neste caso, não se considerou oportuno duplicar as entidades nos modelos de domínio e desenvolver as duas aplicações separadas.

Tendo em conta o requisito em análise, esta implementação iniciou-se com a definição das entidades que constituem o domínio do problema. Para tal, recorreu-se à ação de *drag-and-drop* e posterior configuração de atributos e associações. Como se pretende que um cliente autenticado realize as encomendas, foi necessário considerar a possibilidade de relacionar esta entidade com uma entidade definida para autenticação. A plataforma disponibiliza a entidade *User* que representa um utilizador da aplicação. Para tal, criou-se uma nova entidade *Customer* com referência à entidade *User*, através da propriedade *Generalization*. Para além disso, adicionaram-se os atributos necessários na entidade *Customer*.

A Figura 60 apresenta as entidades *Customer*, *CustomerOrder* e *CustomerOrderItem*, os seus atributos e a respetiva associação entre elas.

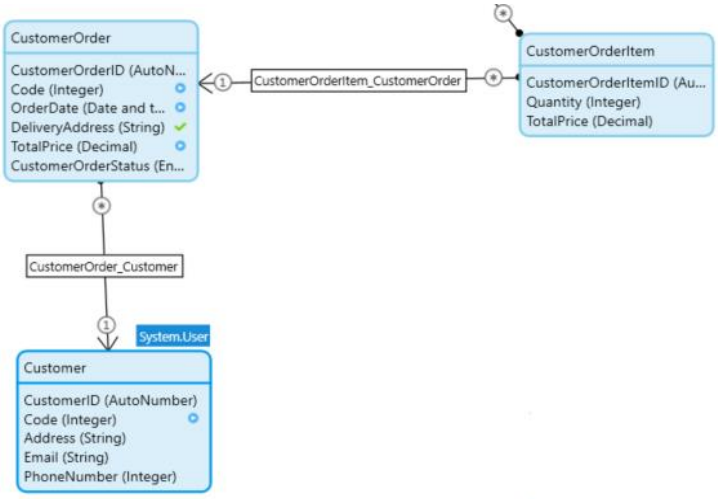


Figura 60 Entidades *Customer*, *CustomerOrder* e *CustomerOrderItem*

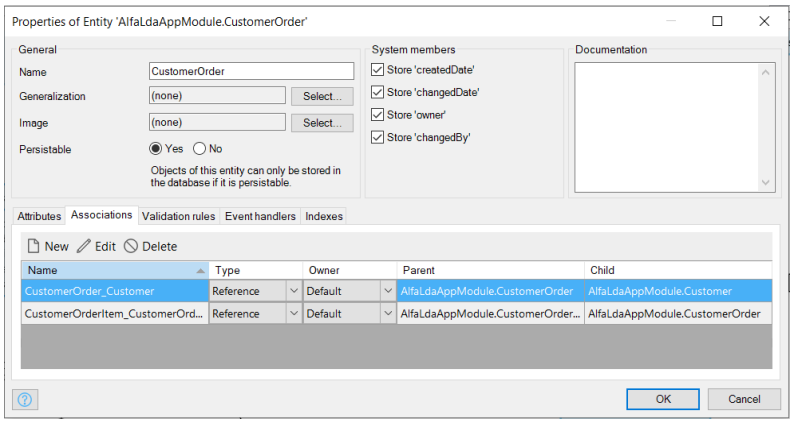


Figura 61 Propriedades da entidade *CustomerOrder*

Na definição dos atributos da encomenda, em casos particulares como o da data ou código, implementaram-se *microflows* para permitem calcular os valores desses atributos. Assim, invés dos atributos serem recebidos e armazenados, são definidos a partir de um fluxo, que é considerado na criação da entidade. Quando este fluxo é definido não é necessário realizar qualquer atribuição ou chamada ao longo da implementação da lógica. No entanto, quando estes atributos são calculados devem estar *read-only* no caso de serem apresentados nos formulários. Para além disso, definiram-se *validation rules* para garantir a obrigatoriedade de alguns atributos. As associações entre as entidades são definidas e representadas visualmente, sendo que estas são identificadas por um nome e por uma cardinalidade. Neste caso, a entidade *CustomerOrder* é *owner* e apresenta uma relação de muitos para um com *Customer*, por outro lado a entidade *CustomerOrderItem* é *owner* e apresenta uma relação de muitos para um com

CustomerOrder (ver Figura 61). Para a definição do estado de encomenda de cliente implementou-se um enumerável, uma vez que se pretende que este apresente valores fixos. Este enumerável foi preenchido com os valores pretendidos, sendo depois invocado na definição do atributo na entidade encomenda. Neste caso, o atributo deve apresentar um valor por defeito que, neste caso, corresponde ao valor *open*.

É possível verificar que a base de dados desta plataforma segue um comportamento de base de dados relacional, acedida através de uma abordagem assente em ferramentas ORM (*Object-Relational Mapping*). Com esta abordagem as estruturas da base de dados são criadas a partir das entidades definidas no modelo de domínio. Assim, a plataforma apresenta um nível de abstração considerável, sendo que as ações à base de dados realizam-se na implementação de *microflows*, através de ações como, por exemplo, *Commit*, *Change*, *Create*, *Delete Object* ou *Retrieve*. Considera-se, assim, que ao nível da camada de acesso a dados pretendida nesta implementação esta plataforma já se responsabiliza por tratar a abstração dos dados e das ações da base de dados.

A plataforma possibilita a chamada de eventos ao nível das páginas da aplicação (p.e. *Save Changes*, *Create Object* ou *Delete*) permitindo, por exemplo, criar diretamente os objetos que estão associados a um formulário. Esta abordagem pode ser mais rápida e interessante no caso de não ser necessário considerar lógica de negócio auxiliar. Caso contrário, é necessário criar fluxos que descrevam a lógica, tal como aconteceu no contexto do requisito em análise, tendo-se implementado um *microflow* para criar uma encomenda de cliente (ver Figura 62).

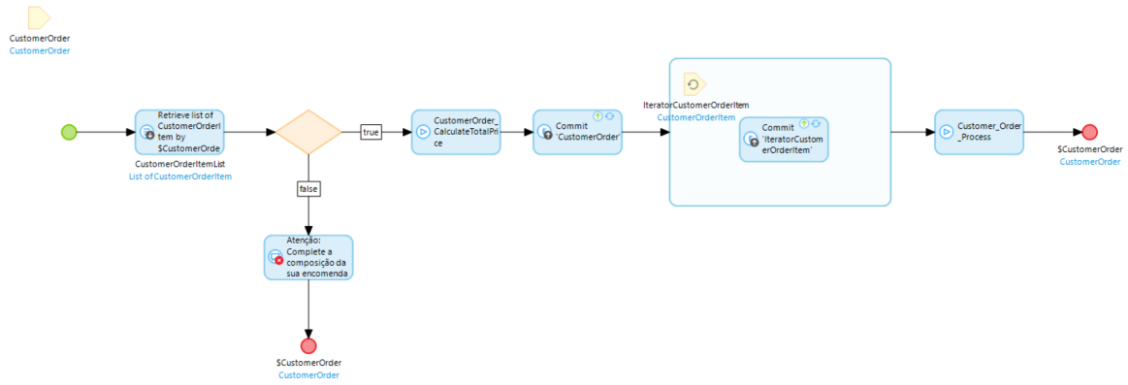


Figura 62 *Microflow* para criar encomenda de cliente

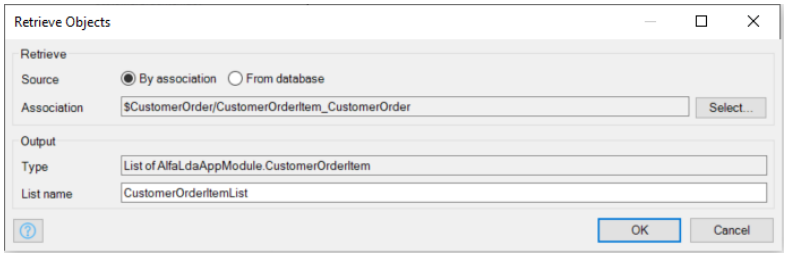


Figura 63 Configuração para obter lista de itens da encomenda por associações dos objetos

O fluxo para criar uma nova encomenda recebe como parâmetro o objeto *CustomerOrder* e começa por obter, por associação (Ver Figura 63), o conteúdo da lista dos itens da encomenda. Depois, o fluxo verifica se esta está vazia e, em caso afirmativo é enviada uma mensagem ao utilizador para completar a composição da encomenda. Caso contrário, é invocado o *microflow* que calcula o preço total da encomenda. De seguida, o objeto é criado na base de dados a partir da atividade *Commit CustomerOrder*, sendo depois necessário iterar sobre a lista de itens da encomenda de forma a criar todos eles na base de dados através de *Commit IteratorCustomerOrderItem*. Por fim, é invocado o fluxo *Customer_Order_Process*, cuja implementação está apresentada Figura 64.

Este fluxo recebe como parâmetro o objeto da encomenda e inicia-se com as ações para obter a lista dos itens encomendas e a composição dos produtos. Segue-se a criação de uma lista e de um objeto auxiliares, que permitem guardar as respostas da verificação de stock, numa fase posterior deste fluxo. Com recurso ao *ForEach* percorre-se a lista dos itens da encomenda, sendo que o ciclo é iniciado com a pesquisa pela composição do produto encomendado, seguindo-se outro *ForEach* para percorrer a lista da composição do produto. Por sua vez, este ciclo chama o fluxo do pedido para obter o stock de matéria-prima e verifica a disponibilidade do mesmo.

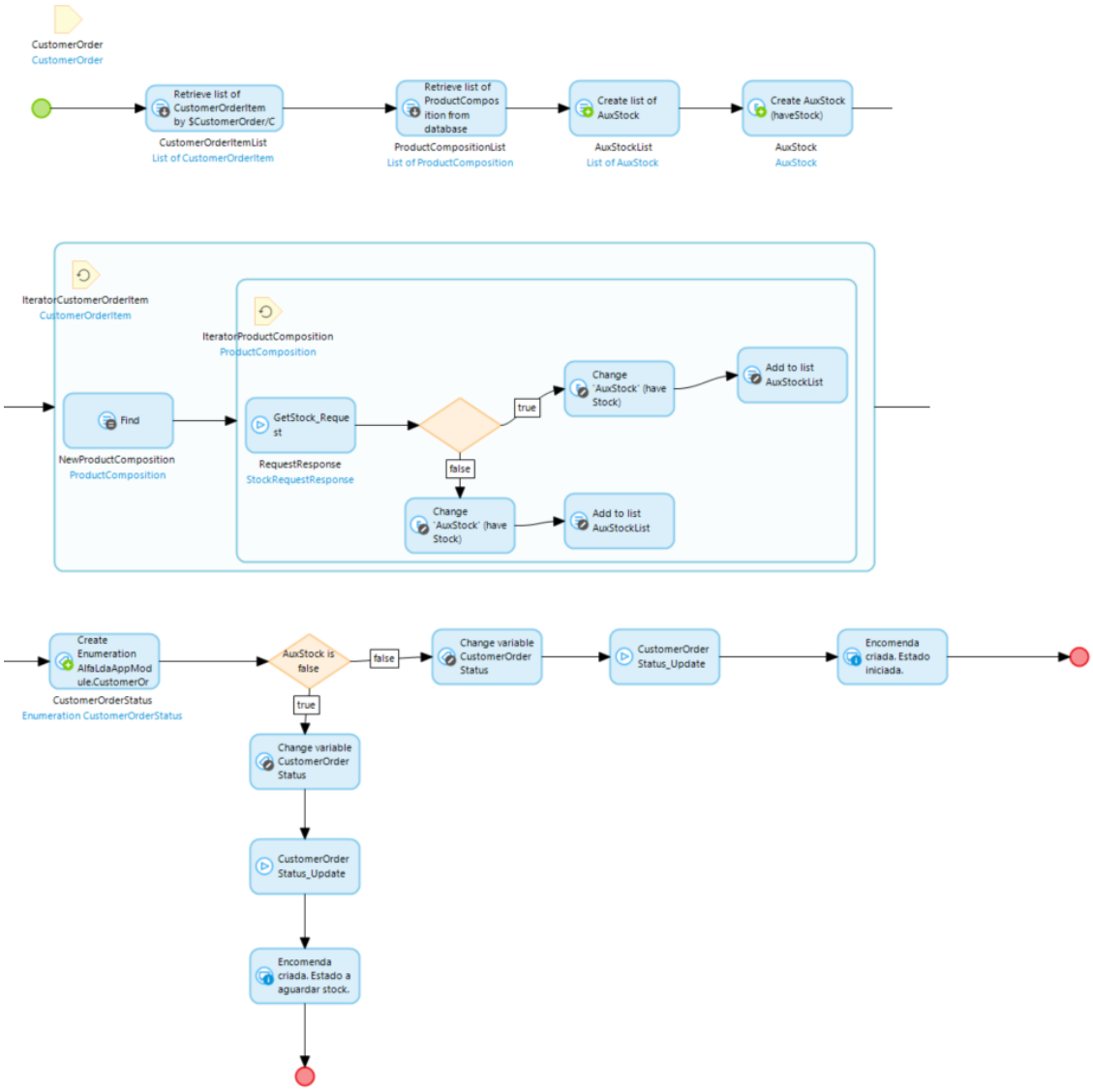


Figura 64 *Microflow* para processamento de encomenda

Para a implementação deste pedido foi necessário definir a lógica de consumo do método de *Rest API*. Assim, definiu-se a estrutura de JSON (ver Figura 65) que recebe um exemplo de resposta em JSON e, automaticamente, cria uma estrutura, genericamente designado objeto, com a definição dos atributos considerados na resposta. Esta estrutura é utilizada na definição do *import mapping* (ver Figura 66), que mapeia o objeto criado na estrutura com um objeto de uma entidade existente, tal como se pode verificar na Figura 67. Neste contexto, implementou-se uma entidade que não persiste na base de dados, sendo apenas utilizada para receber a informação a ser tratada. Desta forma, implementou-se um *microflow*, que recebe como parâmetro o código da matéria-prima a verificar. Este fluxo invoca a atividade *Call Rest*, responsável pela configuração do pedido, nomeadamente do seu URL e da resposta. Na

configuração da resposta, aplicou-se o *import mapping* definido anteriormente e definiu-se a variável para retornar o resultado.

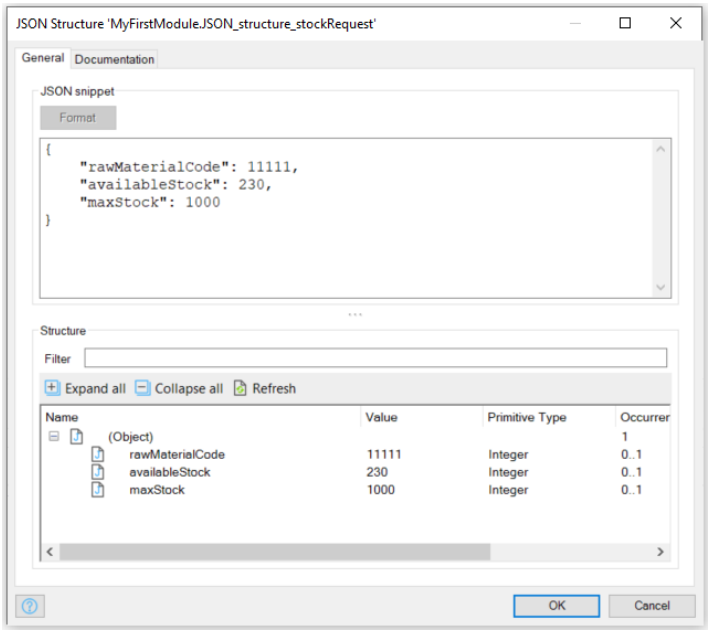


Figura 65 Estrutura de Json para resposta do pedido

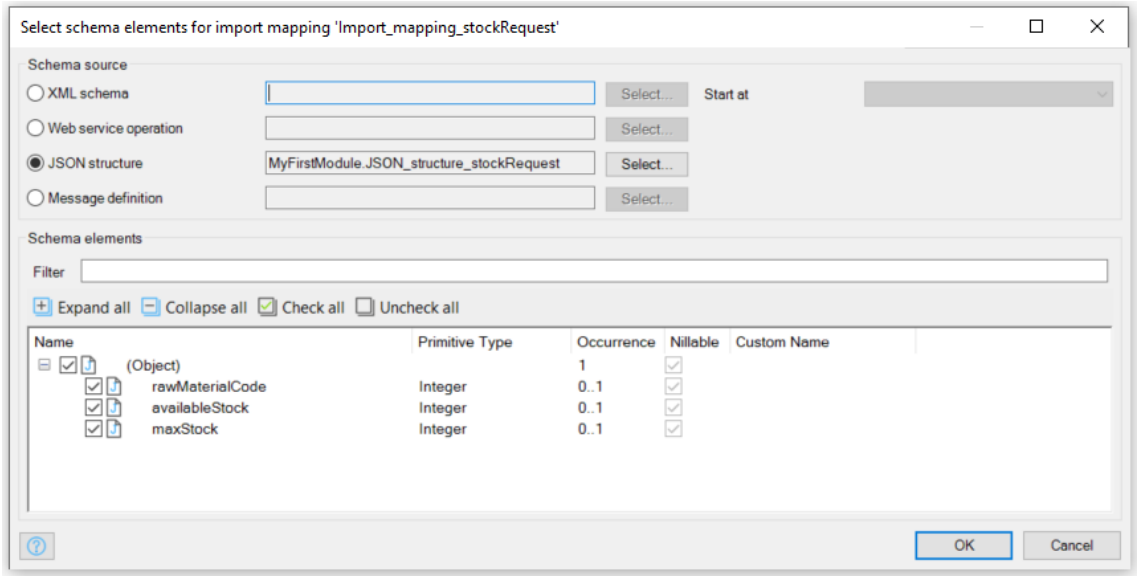


Figura 66 Configuração de *Import Mapping* a partir da estrutura JSON

Figura 67 *Import Mapping* com mapeamento *Root* para *StockRequestResponse*

A resposta do pedido permite, através de uma decisão, verificar se existe stock de matéria-prima necessária para um produto. Dependendo deste resultado o objeto auxiliar é preenchido (com *true* ou *false*) e adicionado à lista auxiliar. Por fim, verifica-se se esta lista apresenta pelo menos um valor *false* e, em caso afirmativo o estado da encomenda é atualizado para *Waiting Stock*. Caso contrário, atualiza-se o estado da encomenda para *Started*.

Na implementação da interface gráfica para criação de encomenda, foi utilizado um modelo de página disponibilizado pela plataforma – *Form Vertical* - e que se destina à implementação de um formulário. O formulário implementado, apresentado na Figura 68, é composto por um *data view*, tendo sido necessário configurar a sua *data source* para mapear o formulário de acordo a entidade *CustomerOrder*. Quando a *data source* foi alterada, automaticamente, o formulário foi mapeado de acordo com os atributos da entidade. Neste mapeamento todos os campos são apresentados de acordo com o tipo do atributo.

Figura 68 Formulário para criar encomenda

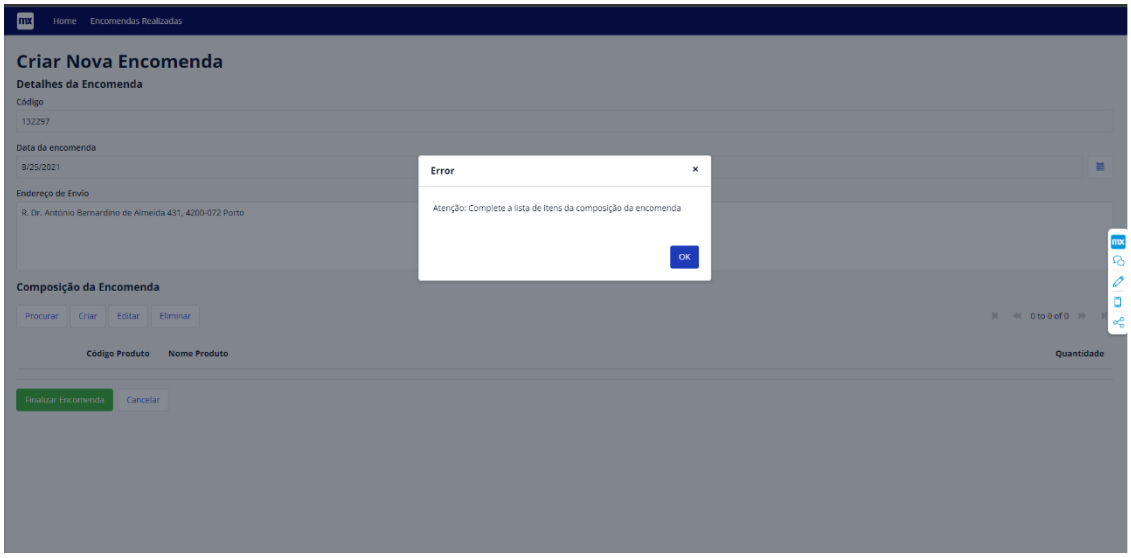


Figura 69 Mensagem de erro de lista de itens vazia

A *data view* apresentada no formulário não considera, automaticamente, os itens que compõe uma encomenda. Contudo, através da associação entre as entidades, adicionou-se uma *data grid*, definindo como *data source* a entidade *CustomerOrderItem* considerada na associação *CustomerOrderItem_CustomerOrder*. Desta *data grid* resulta uma tabela e os respetivos botões para ação de criar, editar ou eliminar um item da encomenda, tendo sido necessário criar e associar uma nova página para as duas primeiras ações. A nova página foi criada da mesma forma que a anterior, tendo sido necessário configurar a *data view* para considerar como *data source* a entidade *CustomerOrderItem*. Para além disso, no formulário de criar encomenda definiu-se no botão Finalizar Encomenda o evento *OnClick* que chama o *microflow* descrito anteriormente (ver Figura 62). Este fluxo inicia-se com a validação da lista de itens de encomenda que, no caso de estar vazia, apresenta uma mensagem ao utilizador, tal como se apresenta na Figura 69. Neste caso, por se tratar de informação da *data grid* a validação foi feita ao nível do *microflow*. No entanto para atributos como endereço de envio foi possível configurar nas propriedades uma validação para que este campo seja obrigatório, bem como a mensagem a apresentar ao utilizador. Adicionalmente, nesta interface configuraram-se as propriedades de visibilidade de navegação, definindo-se permissão aos utilizadores com o *role* de cliente, sendo os *roles* são apenas considerados em níveis de segurança *Prototype/demo* ou *Production*. A este nível, também é possível atribuir permissões para acesso a *microflows* e *nanoflows*.

Nesta solução foram implementados testes de qualidade, recorrendo aos módulos *Unit testing*, *Community Commons Function Library* e *Object Handling* disponibilizados pela plataforma. O módulo para testes unitários apresenta, por omissão, uma interface onde os testes podem ser executados, de forma individual ou em conjunto, e onde se podem consultar os detalhes da sua execução, tal como se pode verificar na Figura 70. Na solução desenvolvida adicionaram-se estes módulos e implementaram-se os *microflows* responsáveis pela lógica do teste. Para o requisito em análise implementou-se um teste para verificar que a lista de itens de uma encomenda não está vazia. Para tal, implementou-se o fluxo apresentado na Figura 71. Neste fluxo são criados os objetos necessários para o teste e é invocado o *microflow* para criação de encomenda, no final com recurso aos *microflows* do módulo de testes definiram-se as ações *ReportStep* e *AssertTrue*. A primeira é utilizada como auxílio para que seja possível verificar, na execução do teste, qual foi o último passo deste. A ação *AssertTrue* apresenta a condição a ser validada no teste.

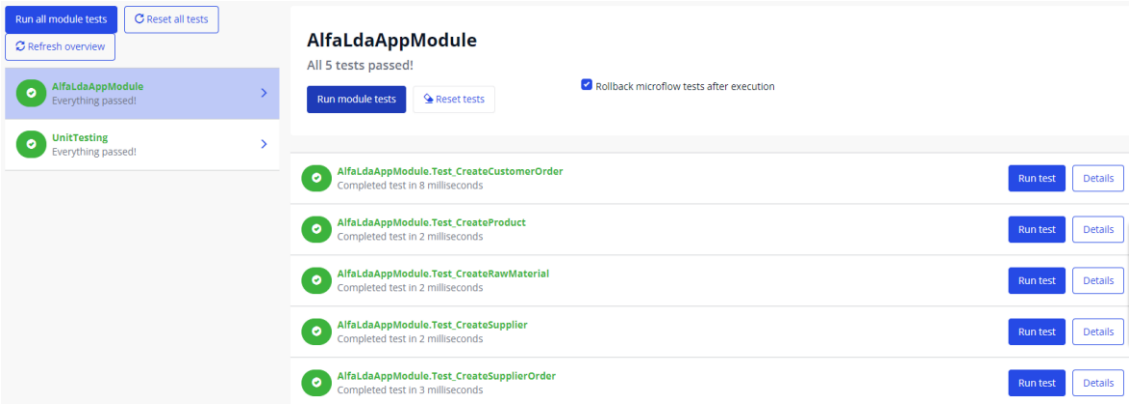


Figura 70 Interface dos testes unitários do módulo da solução recorrendo a *Unit testing*

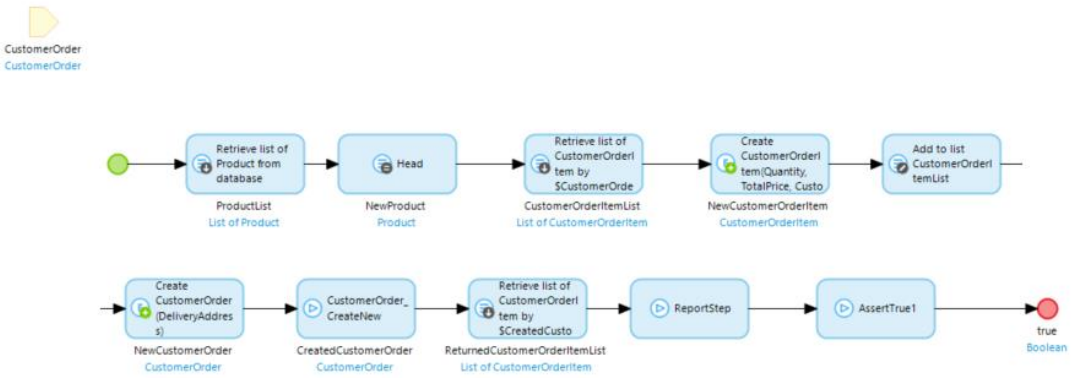


Figura 71 Fluxo para implementação do teste unitário

No contexto desta plataforma foi possível implementar a solução seguindo a arquitetura pretendida. Apesar da aplicação desenvolvida só apresentar, por defeito, o modelo de domínio, a definição da estrutura para seguir as camadas da arquitetura pretendida foi intuitiva. Isto porque, a plataforma permite a organização da solução por pastas e, para além disso, apresenta conceitos como modelo de domínio, *microflows* e interfaces que foram, facilmente, associados às camadas da aplicação. Neste desenvolvimento foi possível garantir que cada camada apresentasse uma única responsabilidade e comunicasse com a camada imediatamente inferior. Contudo, verificou-se que ao longo do desenvolvimento foram utilizadas entidades do domínio na definição de lógica e interfaces. Apesar de não se ter considerado outra abordagem, seria possível definir entidades que não persistem na base de dados, podendo estas ser utilizadas apenas para transferir informação. Neste caso, o mapeamento entre as entidades teria que ser definido através da definição de uma *message definition*, que define a estrutura da entidade persistida e de um *import/export mapping* para definir o mapeamento dos atributos das entidades. Esta abordagem seria semelhante ao, tradicionalmente conhecido, padrão DTO.

Tal como referido em 4.1.4 Arquitetura, para este caso de estudo foi considerada uma arquitetura alternativa. Para a adoção desta alternativa seria necessário considerar a implementação de uma aplicação com dois módulos. Neste caso, num destes módulos mantinha-se a lógica e definição de entidades e adicionava-se a publicação de uma *Rest Api*, no outro implementavam-se as interfaces e consumiam-se os pedidos definidos no módulo anterior. Para além disso, este módulo teria que considerar a definição de entidades que não persistem na base de dados como auxiliares ao desenvolvimento. Neste caso, as alterações para esta abordagem alternativa seriam possíveis, no entanto, a implementação exigiria mais ações e a duplicação das entidades de domínio em entidades não persistidas. Ao nível das abordagens consideradas na implantação do sistema em 4.1.4 Arquitetura, verificou-se que o *runtime* desta plataforma é composto por duas partes (Mendix, 2021b): (i) *Runtime Server*, responsável pela execução da lógica do servidor (p.e. *microflow* e base de dados) e (ii) *Mendix Client*, responsável pela execução de aplicações cliente web e mobile, sendo iniciado pelo utilizador final. Ao nível de aplicações web, o *Mendix Client* atua como uma SPA. A este nível, não é possível ter controlo sobre esta abordagem na plataforma. Tal como se esperava, a este nível, a plataforma segue uma abordagem considerada mais moderna.

5.2.3 Caso de Estudo 2

Tal como descrito em 4.2 Caso de estudo 2 para este caso de estudo pretendia-se seguir uma abordagem de arquitetura orientada a microserviços, tendo como ponto de partida a solução implementada para o caso de estudo 1. Através de *copy-paste*, reaproveitaram-se microflows e outras ações definidos na solução anterior. Para tal, criou-se uma aplicação por cada microserviço, sendo cada uma delas responsável por definir as entidades, implementar a lógica de negócio e expor uma API. Criou-se, também, uma aplicação móvel destinada à implementação de interfaces e lógica de apresentação. Para além disso, manteve-se a aplicação implementada no caso de estudo anterior, tendo-se implementado as alterações necessárias para este caso de estudo.

No âmbito do requisito em análise, esta implementação iniciou-se com a definição das entidades para o Serviço CEC. Para tal, consideraram-se as entidades definidas anteriormente, tendo sido apenas necessário adicionar o atributo *ProductID* do tipo *Integer* na entidade *CustomerOrderItem*. Esta alteração foi considerada, uma vez que deixa de se considerar a associação de *Product* com *CustomerOrderItem*.

Para a publicação da API do serviço do requisito em análise, adicionou-se um *Published Rest Service* que, para além da versão e do *endpoint* do serviço, é composto por recursos que correspondem a um conjunto de métodos *Rest*. A plataforma disponibiliza a documentação da API (ver Figura 72 e Figura 73) que apresenta os seus métodos, atributos e modelos e que permite testar os métodos definidos. Para além disso, assegurou-se a autenticação basic, que é a configuração padrão e que é automaticamente aplicada quando o nível de segurança é *Prototype/demo* ou *Production*. Para a criação de uma encomenda pelo cliente, adicionou-se a operação que recorre ao método POST. A Figura 74 apresenta a configuração para esta operação e tal como é possível verificar foi necessário chamar um *microflow* e adicionar um parâmetro do tipo *body* que recebe a informação para o pedido.

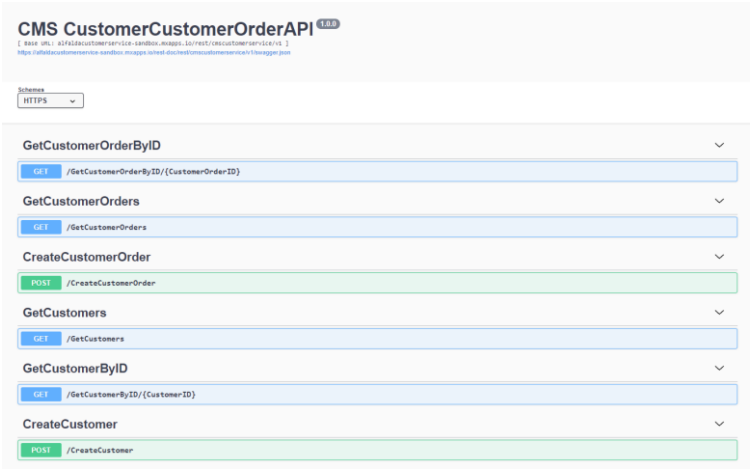


Figura 72 Documentação da *CustomerCustomerOrderServiceAPI*

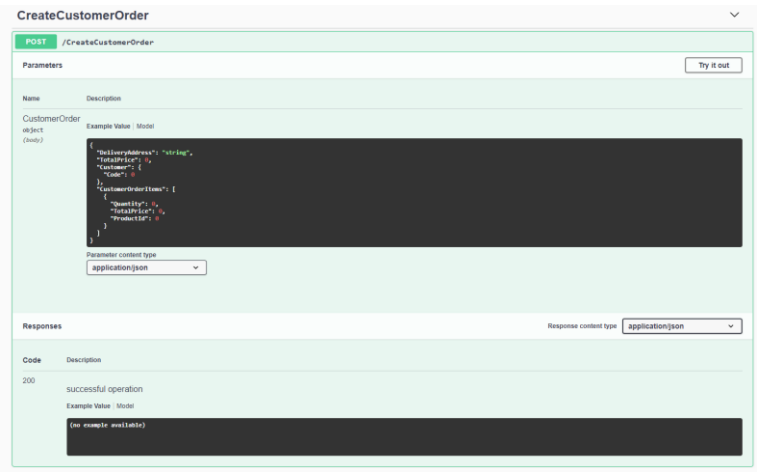


Figura 73 Detalhes da documentação do método POST

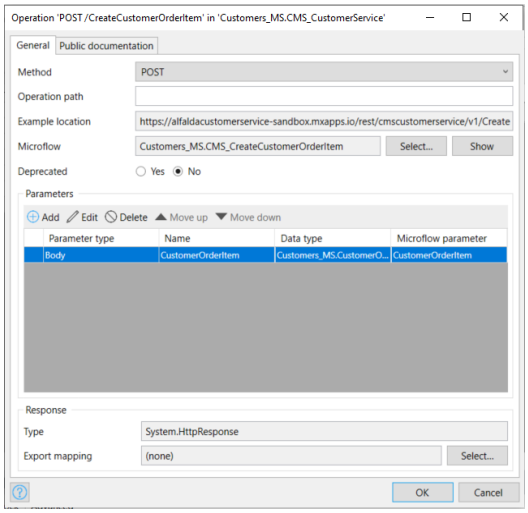


Figura 74 Configuração da operação POST da API

Para a implementação do *microflow* foi necessário definir, previamente, uma *message definition* para criar uma estrutura da entidade e que é utilizada no *export mapping*. Neste selecionaram-se os atributos necessários e realizou-se o mapeamento. Uma vez que estas ações permitem selecionar os atributos a serem utilizados, verificou-se que não seria necessário implementar entidades que não persistem na base de dados. O fluxo para criar encomenda, apresentado na Figura 75, é semelhante ao fluxo implementado no caso de estudo anterior, no entanto consideraram-se as seguintes alterações:

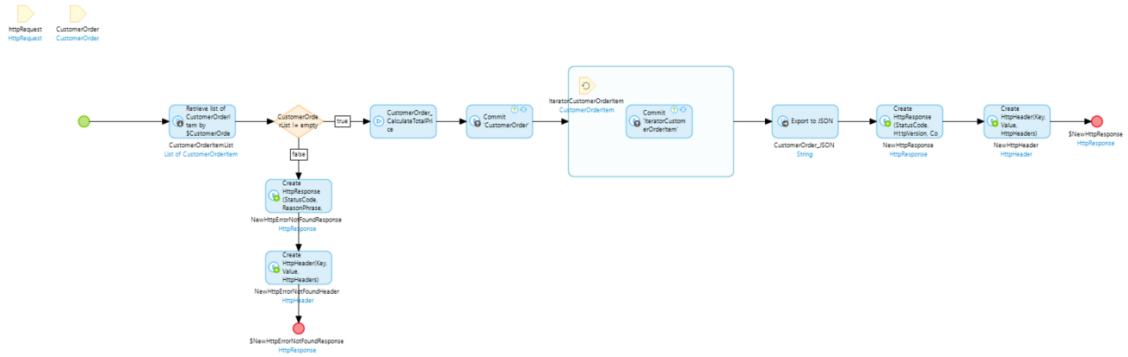


Figura 75 Microflow para operação POST de criar encomenda

- Após a validação da lista de itens da encomenda invés da apresentação de mensagens de sucesso/insucesso passam a considerar-se as respostas do pedido HTTP. Desta forma, quando a lista está vazia retorna-se o *Status Code* 400 e uma mensagem de erro. Por outro lado, se a lista estiver preenchida utiliza-se a ação *export with mapping* que mapeia a resposta do pedido para JSON, através da utilização do *export mapping* definido previamente. Esta ação realiza-se para que o pedido retorne o *Status Code* 201 e o conteúdo da variável resultante do mapeamento anterior. Para estas respostas criaram-se objetos do tipo *System.HttpResponse* e alteraram-se os seus atributos – *StatusCode*, *ReasonPhrase*, *HttpVersion* e *Content* A Figura 76 apresenta este objeto com a alteração dos atributos para o cenário de resposta de *Status Code* 400. Para além disso, foi necessário criar um objeto do tipo *System.HttpHeader* e alterar os atributos *Key*, *Value* e *System.HttpHeaders*, sendo atribuída a variável da resposta anterior;

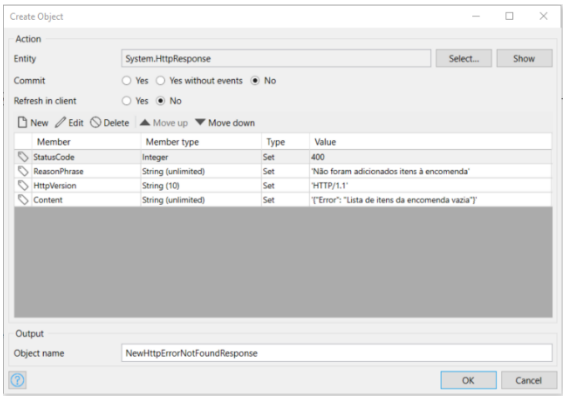


Figura 76 Objeto *System.HttpResponse* para resposta de erro

- De forma a obter o preço dos produtos e a sua composição foi necessário consumir os pedidos publicados no Serviço PMP. A implementação do consumo destes pedidos foi semelhante ao descrito em 5.2.2 Caso de Estudo 1, tendo sido necessário considerar entidades que não persistem na base de dados, uma estrutura *Json* e um *import mapping*. Para as entidades estabeleceram-se as relações entre elas, de forma a facilitar a implementação no fluxo, possibilitando, por exemplo, a ação para obter uma lista por associação;

  Ao nível fluxo do processo adicionou-se a chamada ao fluxo do pedido para a composição dos produtos e adicionou-se a ação para obter a lista desta composição, de forma a, através do *ForEach*, se iterar sobre cada composição. Por fim, este fluxo atualiza o estado da encomenda, no entanto não é retornada nenhuma mensagem, uma vez que, neste contexto, pretende-se que esta mensagem seja retornada na resposta do pedido para criar encomenda.

Tal como referido, ao nível do Serviço PMP implementou-se um método que, a partir do id de um produto, retorna a sua composição. Apesar de já se considerar o método para obter um produto pelo id, este método permitiu implementar uma resposta apenas com a informação necessária que, neste caso corresponde à lista da composição do produto (ver Figura 77). A Figura 78 apresenta o fluxo deste método. Este inicia-se com a ação para obter o produto da base de dados, seguindo-se a ação para obter a lista da sua composição. Depois, é implementada uma condição, para garantir que o produto e a lista existem. O retorno desta condição é igual ao descrito no fluxo para criar encomenda, sendo criados os objetos *System.HttpResponse* e *System.HttpHeader*.

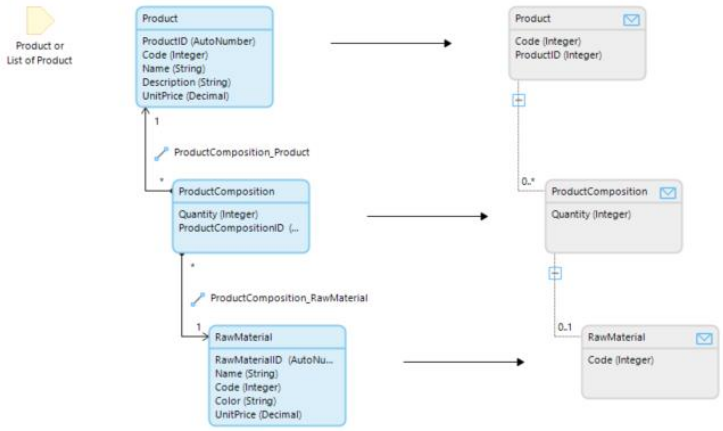


Figura 77 *Export Mapping* para resposta do pedido Get para obter composição de produto

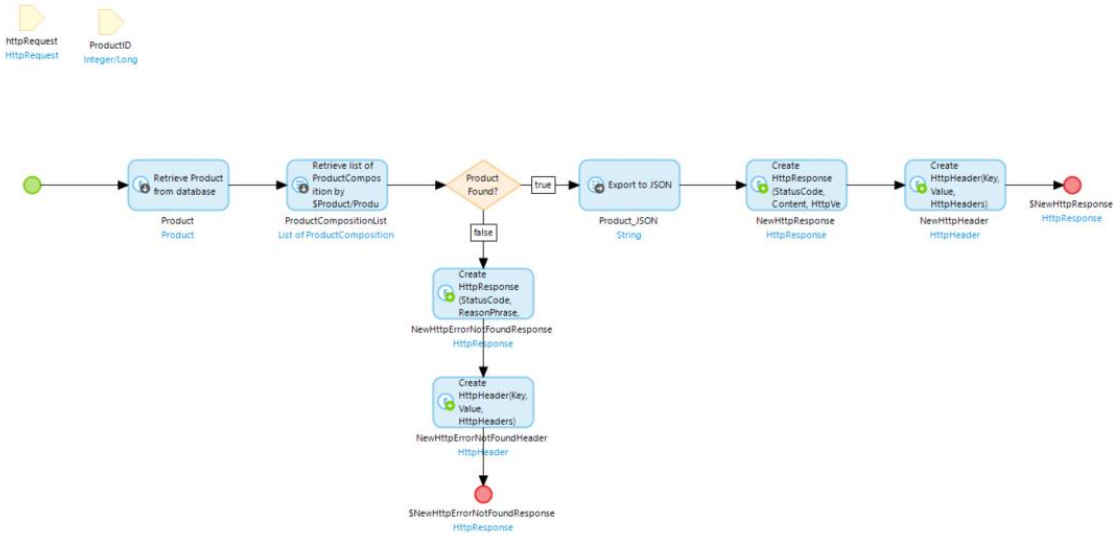


Figura 78 *Microflow* da operação Get para obter lista de composição de produto

Ao nível da aplicação móvel esta plataforma permite o desenvolvimento de interfaces para aplicação nativa ou para aplicação responsiva. Neste sentido, optou-se por implementar interfaces para aplicação responsiva, recorrendo aos *templates* fornecidos pela plataforma, à semelhança de 5.2.2 Caso de Estudo 1. Adicionalmente, foi necessário adicionar o perfil de *phone web* nas configurações da aplicação, de forma a ser possível executar a aplicação num dispositivo móvel. Apesar disso, a utilização dos mesmos *templates* e estrutura de páginas facilitou a implementação das interfaces, uma vez que já existia um conhecimento e experiência

prévia. Assim, ao nível do requisito em análise, implementou-se a página para criar uma nova encomenda que se apresenta na Figura 79.

Figura 79 Formulário de criar encomenda na aplicação móvel

Como se pretendia que esta aplicação consumisse, via API *gateway*, os pedidos desenvolvidos nos serviços, foi necessário definir entidades que não persistem na base de dados. Esta entidades foram utilizadas ao nível do mapeamento de dados (*import/export mapping*) e ao nível da *data source* das páginas. Esta *data source* foi considerada, mais uma vez, na *Data View* do formulário, tendo sido atribuída a entidade não persistida defina previamente para encomenda de cliente. Ao nível da composição da encomenda adicionou-se uma *data grid* que apresenta como *data source* a entidade não persistida para item da encomenda. Por esta razão, contrariamente ao caso de estudo 1, a *data grid* não apresenta, automaticamente, os botões para adicionar informação. Este foram adicionados manualmente, assim como a página para se inserir a composição da encomenda. Nesta página, foi necessário invocar um *microflow* para obter a lista de todos os produtos, através de um pedido Rest que retorna uma lista de produtos como resposta. Para além disso, para finalizar a encomenda implementou-se o fluxo do consumo do pedido POST para criar encomenda (ver Figura 80), sendo este chamado no evento *OnClick* do botão do formulário. Este fluxo recebe como parâmetro o objeto da encomenda e inicia-se com a ação para obter a lista de todos os itens da encomenda. Segue-se a validação dos dados e a chamada ao pedido para criar encomenda. O fluxo termina com a apresentação de uma mensagem, conforme o *Status Code* retornado.

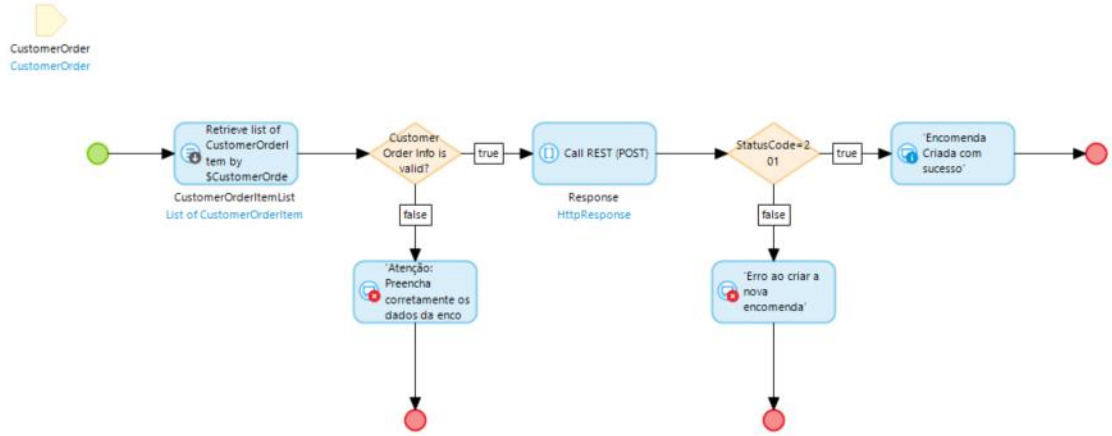


Figura 80 *Microflow* para consumo do pedido POST para criar encomenda

Nesta solução implementaram-se testes de qualidade, recorrendo, mais uma vez, aos módulos disponibilizados. Nesta solução os testes focaram-se nos *microflows* que definem os pedidos das APIs publicadas, uma vez que os restantes fluxos que auxiliam esta implementação já foram testados anteriormente. A principal diferença nos testes neste caso de estudo diz respeito ao tratamento da resposta do *microflow* do pedido. No fluxo implementado aplicou-se um *export mapping*, a ser utilizado para comparação na validação da ação *AssertTrue*. Alternativamente, poder-se-ia ter considerado a validação do *StatusCode* do pedido.

A principal diferença na implementação de uma aplicação móvel, em relação a uma aplicação *web* foi ao nível da configuração do perfil para este tipo de aplicação. Esta configuração permite que a aplicação móvel seja publicada como uma PWA com possibilidade de selecionar a opção que permite que a aplicação pré-carregue recursos estáticos, como páginas ou imagens em segundo plano (Mendix, 2021c). Para publicar uma aplicação como PWA esta deve estar publicada na *cloud*. Após esta configuração é possível conectar-se à aplicação, a partir de um dispositivo móvel através da leitura do QR *code* ou a partir de um *url* que pode ser acedido a partir de um *browser*. Para além disso, esta plataforma disponibiliza a construção de aplicações móveis nativas, através da funcionalidade *Build Native App* que prevê serviços adicionais de *GitHub* e *Microsoft Visual Studio AppCenter*. A plataforma disponibiliza, também, a configuração de perfil de aplicações nativas, fornecendo a aplicação *Make It Native* que permite a pré-visualizar, testar e fazer *debug* de aplicações móveis nativas. Esta aplicação está disponível para Android e iOS.

Ao nível da implementação desta solução seguindo a arquitetura pretendida verificou-se que a plataforma permitiu a definição de serviços isolados que disponibilizam uma API e a definição de aplicações finais que comunicam com a *api gateway* através de pedidos *Rest*. Ao nível do isolamento de aplicações, esta plataforma considera ambientes dentro da *Mendix Cloud*, sendo cada aplicação executada num ambiente totalmente separado de outras aplicações (Mendix, 2021d). Para além disso, cada aplicação é executada em uma ou mais instância do *Mendix Runtime* dentro do ambiente e apresenta uma base de dados própria. Ao longo deste desenvolvimento a principal limitação desta comunicação verificou-se quando se pretendiam testar as ligações dos pedidos. Como a plataforma permite que as aplicações sejam executadas localmente ou em produção (através da sua publicação), quando se estabeleceu a comunicação do Serviço CEC com o Serviço PMP, deixou de ser possível testar localmente, tendo sido necessário publicar sempre os dois serviços para que fosse possível a sua comunicação. Isto porque, apenas assim se assegurava a autenticação, uma vez que quando as aplicações são publicadas, automaticamente passam a considerar a autenticação definida por defeito. Ao nível da alternativa considerada em 4.2.4 Arquitetura verificou-se que esta plataforma disponibiliza conetores (e.g. *Kafka*) que permitem a integração dos serviços com *message broker*. Considerando a dificuldade em tentar testar esta integração, não foram realizadas experiências para validar a adequabilidade destes conectores.

Na adaptação da arquitetura já existente em 5.2.2 Caso de Estudo 1 para a nova abordagem, verificou-se que, apesar das diferenças na estrutura da solução, foi possível adaptar e reaproveitar os componentes que já estavam implementados. Isto permitiu facilitar o processo de implementação da nova arquitetura. Ao longo do desenvolvimento, ao nível das aplicações finais, foi necessário duplicar as entidades de domínio, passando estas a não serem persistidas na base de dados. Nesta alteração, apesar de se continuar a considerar as associações entre entidades, o que facilitou, por exemplo, a integração de *data grid* no formulário, verificou-se que deixou de se usufruir das capacidades da plataforma a este nível. Isto porque foi necessário, por exemplo, implementar lógica para as *dropdown list*, que como invocam entidades que não persistem na base de dados devem ter na sua *data source* um *microflow* associado. Caso se tivesse implementado a alternativa descrita em 4.2.4 Arquitetura, a adaptação para esta arquitetura poderia ser mais intuitiva, uma vez que as aplicações finais já estariam preparadas para receber e enviar pedidos. No entanto, se esta abordagem fosse seguida deixaria também de se usufruir das principais capacidades desta plataforma, pelo menos ao nível das interfaces.

6 Comparação das plataformas

Este capítulo apresenta a descrição a comparação das plataformas em análise, tendo em conta a construção dos casos de estudo, a experiência pessoal e a investigação dos principais aspetos das plataformas em estudo. Esta comparação tem em conta os seguintes aspetos, considerados tendo em conta as questões definidas em 1.3 Objetivos: (i) utilização das plataformas; (ii) metodologia de desenvolvimento; (iii) arquitetura de soluções; (iv) testabilidade; (v) disponibilidade e escalabilidade e (vi) custos e subscrições. Os subcapítulos seguintes apresentam a comparação de cada um destes aspetos.

6.1 Utilização das Plataformas

Este subcapítulo apresenta a comparação das plataformas considerando o método adotado, usabilidade e curva de aprendizagem.

6.1.1 Método Adotado

A falta de experiência e conhecimento nas plataformas determinou que fosse necessário, numa fase inicial, estudar e analisar a documentação e/ou cursos de cada uma das plataformas. Este estudo foi crucial para o entendimento de conceitos e funcionalidades que, mais tarde foram aplicados no desenvolvimento das soluções. Durante este estudo, verificou-se que a experiência prévia de programação permitiu que a aprendizagem nestas plataformas fosse mais rápida e fácil de compreender. Apesar do estudo inicial, ao longo do desenvolvimento foi necessário consultar a documentação e/ou fóruns das plataformas à medida que surgiam dúvidas.

A construção das soluções iniciou-se com o desenvolvimento do caso de estudo 1 em Outsystems, tendo-se, depois implementado em Mendix. No desenvolvimento do caso de estudo 2 seguiu-se a mesma ordem. Importa realçar que esta ordem foi aleatório e não seguiu nenhum critério. Este processo permitiu que, de forma mais direta, fossem identificadas as semelhanças e diferenças nas plataformas para cada caso de estudo. Por outro lado, identificou-se uma limitação neste processo, pois a primeira implementação em Outsystems tornou o desenvolvimento em Mendix mais fácil, no sentido em que já se conheciam as principais características que são comuns a estas plataformas.

6.1.2 Usabilidade

As plataformas em análise diferem na sua estrutura e apresentação para os desenvolvedores. A experiência de construção nestas plataformas permitiu avaliar a sua usabilidade. Assim, considerando os critérios de usabilidade definidos em (Quesenbery, 2004)– eficácia, eficiência, *engaging*, tolerância ao erro e fácil de aprender - avaliaram-se as plataformas segundo a Escala de (Likert, 1932). Em Anexo 3 – Tabelas de avaliação de usabilidade apresenta-se os detalhes desta escala, bem como dos critérios avaliados. A Figura 81 apresenta o gráfico com os resultados desta avaliação e respetiva comparação entre as plataformas.

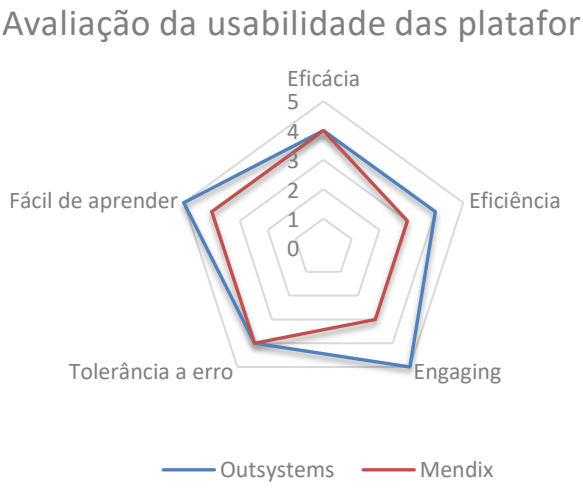


Figura 81 Resultado da avaliação pessoal da usabilidade das plataformas Outsystems e Mendix

O *Service Studio* da Outsystems apresenta interfaces *user-friendly* e fáceis de memorizar. Para além das suas interfaces apelativas, esta plataforma disponibiliza uma estrutura bem definida nas suas aplicações o que contribui para um desenvolvimento mais simples e intuitivo. As propriedades dos componentes desenvolvidos são apresentadas quando se clica sobre os mesmos, o que permite a sua alteração em poucos segundos. Adicionalmente, inclui ajuda contextual apresentando nos componentes uma *tooltip* de ajuda ao passar o cursor em cima e apresentando informação sobre o objetivo das propriedades quando se passa o cursor em cima destas. Nesta plataforma destaca-se, também, a capacidade de esta detetar o tipo de dados através do nome do atributo (cf. 5.1.2 Caso de Estudo 1) permitindo, durante o desenvolvimento, que a definição deste tipo seja ainda mais imediata.

O *Mendix Studio Pro* apresenta, também, interfaces *user-friendly*. No entanto, por esta plataforma não apresentar uma estrutura definida nas suas aplicações a adaptação às suas funcionalidades foi um pouco mais demorada, confusa e difícil de memorizar. Ao nível da configuração de propriedades das atividades num fluxo é necessário fazer duplo clique e abrir a janela para se editarem os detalhes dessas atividades. Para além disso, essas propriedades, a par das propriedades apresentadas nas interfaces gráficas não apresentam ajuda contextual, sendo que em caso de dúvidas foi necessário realizar uma pesquisa sobre o significado e/ou objetivos dessas propriedades. Esta ações exigem mais tempo de adaptação.

No tratamento de erros ao longo do desenvolvimento estas plataformas apresentam comportamentos semelhantes, fornecendo informação sobre o erro e a forma como este pode ser resolvido. Considera-se que a plataforma Outsystems apresenta, ao nível da usabilidade, ações e funcionalidades mais intuitivas e fáceis de memorizar, o que permitiu reduzir o tempo de adaptação à plataforma. Por outro lado, a plataforma *Mendix* apresenta ações e funcionalidades que requerem mais cliques e que não disponibilizam ajuda contextual, o que contribuiu para que o tempo de adaptação às interfaces desta plataforma fosse superior relativamente à outra plataforma. É importante considerar que esta plataforma apresenta também o *Mendix Studio* destinada a *no-code*, o que favorece perfis menos experientes ao nível da programação. No entanto, neste caso este ambiente não foi utilizado e, assim, não foi considerado nesta avaliação.

6.1.3 Curva de aprendizagem

Nos casos de estudo definiram-se requisitos com um nível de complexidade idêntico (cf. 4.1.2 Requisitos Funcionais), sendo que isto permitiu analisar o esforço necessário para o seu desenvolvimento, ao longo do tempo. Uma vez que nesta implementação não se consideraram os tempos de implementação de cada um dos requisitos, optou-se por analisar a curva de aprendizagem numa abordagem qualitativa. A conceito de curva de aprendizagem, introduzido por (Ebbinghaus, 1885), prevê que a eficiência em realizar uma tarefa aumenta à medida que um individuo pratica aquela tarefa. Assim, o esforço para terminar uma tarefa diminuirá ao longo do tempo. Para esta análise teve-se em conta as estimativas de esforço definidas em Anexo 4 – Tabela de Estimativa de Esforço que permitiram ter uma ideia do tempo e do esforço previsível para o desenvolvimento dos requisitos, ao nível do caso de estudo 1. Os pares de requisitos UC4-UC5 e UC3-UC6 apresentavam o mesmo valor estimado, uma vez que

correspondiam a requisitos semelhantes em termos de necessidades e complexidade. No processo de desenvolvimento deste caso de estudo estes implementaram-se na ordem seguinte: UC4, UC5, UC3 e UC6. Desta forma, seria expectável que se verificassem os seguintes cenários:

- Depois do desenvolvimento de UC4, a estimativa de UC5 seria igual ou menor à estimativa definida inicialmente, ou seja, igual ou menor a 2;
- Depois do desenvolvimento de UC3, a estimativa de UC6 seria igual ou menor à estimativa definida inicialmente, ou seja, igual ou menor a 3.

Estes cenários são esperados, tendo em conta que, numa fase inicial, a realização de uma tarefa exigiu mais tempo e esforço, uma vez que não existia tanto conhecimento da plataforma e das suas funcionalidades. Por outro lado, esperava-se que a realização de uma tarefa semelhante, numa fase seguinte, não exigisse o mesmo tempo e esforço estimado inicialmente. Ao nível do desenvolvimento, em Outsystems o requisito UC3 exigiu um maior esforço e tempo inicial, em relação à implementação do mesmo requisito em Mendix. Apesar disso, verificaram-se, em ambas as plataformas, os cenários descritos, tendo os requisitos UC5 e UC6 exigido menor esforço e tempo em relação ao esperado. Esta análise pode indicar que nestas plataformas a curva de aprendizagem pode apresentar uma tendência crescente em termos de eficiência, ao longo do tempo de utilização das mesmas. Adicionalmente, o desenvolvimento do caso de estudo 2 permitiu reforçar esta análise, uma vez que este foi desenvolvido tendo em conta o conhecimento adquirido no primeiro caso de estudo. Neste caso, apesar de se terem realizado algumas alterações, verificou-se que o esforço e tempo necessários foram diminuindo ao longo do tempo de desenvolvimento.

6.2 Metodologia de desenvolvimento

Apesar de não se ter considerado (diretamente) a adoção de uma metodologia de desenvolvimento neste projeto, foi possível analisar quais são as principais orientações que as plataformas em estudo recomendam e/ou seguem. Estas recomendam, essencialmente, a adoção de metodologias ágeis de desenvolvimento.

6.2.1 Outsystems

A (Outsystems, 2021d) orienta as equipas a seguirem uma abordagem ágil, pois acreditam que esta é a melhor decisão, uma vez que com a exigência dos utilizadores finais, as soluções podem ser desenvolvidas de forma iterativa, acompanhadas com o feedback contínuo dos utilizadores finais. Nesta abordagem a Outsystems considera que uma equipa deve focar-se na colaboração e deve incluir diferentes perfis, tais como (Outsystems, 2021d): *Product Owner*, *Tech Lead*, *Developers*, *Project Manager*, *Business Analyst*, *Tester/QA Coordinator*, *Program Manager*, *Experts*, *Entreprise Architect* e *IT Manager*. A colaboração da equipa de desenvolvimento é suportada por um controlo de versões incorporado na ferramenta que permite que todos os desenvolvimentos sejam armazenados num repositório central. Durante o desenvolvimento das soluções este processo foi realizado automaticamente, através de um clique no botão que, simultaneamente publica a aplicação. Apesar disso, a plataforma apresentava mensagens quando detetava a necessidade de realizar um *merge*. Com esta abordagem os projetos passam a considerar uma *timebox*, que soma o esforço necessário para todas as características originais do *backlog* e deve ser definido no início, para permitir uma melhor previsão de custos e tempo de comercialização. Por sua vez, consideram-se critérios de priorização que devem ser atribuídos às iterações, tendo em conta as características que apresentam um maior valor comercial ou a dependência entre elas. Por fim, o feedback permite considerar alterações ou novas características nas próximas iterações. Desta forma, há um foco contínuo nas características prioritárias, acompanhado de feedback para garantir que as soluções são desenvolvidas tendo em conta os objetivos do negócio. Por outro lado, a Outsystems acredita que o tradicional modelo em cascata não consegue responder às exigências atuais e, dessa forma, não apresenta capacidade de resposta em tempo útil para impulsionar valor para o utilizador final (Outsystems, 2021d).

6.2.2 Mendix

A (Mendix, 2021e) também orienta as equipas para uma abordagem ágil, considerando que esta permite um aumento da colaboração, capacidade das equipas de desenvolvimento e transparência no processo de desenvolvimento, sempre acompanhados de feedback contínuo dos utilizadores finais. Nesta abordagem a Mendix foca-se na adoção de *Scrum*. Neste sentido, a plataforma descreve os principais papéis num projeto seguindo esta abordagem (Mendix, 2021e) – *Scrum Master*, *Product Owner*, *Subject Matter Experts*, *Business Owner* e *Development*

Team. Para além disso, são também consideradas as cerimónias de *Scrum* – *Sprint Planning*, *Daily Stand-Up*, *Sprint Review Meeting* e *Retrospective*. De forma a não limitar a orientação para *Scrum*, a plataforma apresenta dois exemplos de possíveis metodologias a utilizar, em alternativa a *Scrum*: *Kanban* e *Extreme Programming* (Mendix, 2021e). Apesar disso, durante o processo de estudo da plataforma e no desenvolvimento, foi possível verificar que esta disponibiliza no *Developer Portal* várias funcionalidades que suportam a colaboração entre equipas, sendo estas baseadas nas principais características e conceitos de *Scrum*. Por exemplo, nesse portal podem ser definidas *stories* associadas a uma *sprint*. Esta plataforma apresenta, ainda, o Team Server que permite o controlo de versões de uma aplicação desenvolvida em Mendix Studio Pro e Mendix Studio. Este permite, por exemplo o *commit* ou *update* do projeto, sendo que, no caso do *commit* este pode ser associado a uma *story* definida para o projeto.

6.2.3 Sumário

Tal como se pode verificar ambas as plataformas privilegiam a adoção de metodologias ágeis. Apesar disso, verifica-se que enquanto Mendix tende a orientar e fornecer ferramentas para a adoção de *Scrum*, Outsystems orienta apenas para uma abordagem ágil, não direcionando esta escolha para uma metodologia em particular. Por um lado, a abordagem de Outsystems pode permitir uma maior liberdade de escolha desta metodologia, contudo, por outro lado, a abordagem de Mendix pode ajudar as equipas numa melhor adaptação à plataforma, uma vez que fornece ferramentas para tal. Ao nível de controlo de versões, como suporte à colaboração entre equipa, verifica-se que as plataformas apresentam diferentes abordagens. Mendix disponibiliza um controlo de versões que segue um formato tradicional e já conhecido dos desenvolvedores, o que pode permite, por um lado, uma melhor adaptação por parte destes, mas, por outro lado, poderá dificultar a utilização de *citizen-developers*. A Outsystems disponibiliza um controlo de versões com um elevado nível de abstração e, apesar de adicionar alterações e realizar *merge* das mesmas pode limitar equipas que pretendem ter uma maior flexibilidade no controlo de versões dos seus projetos. Por outro lado, a abstração apresentada pode ser vantajosa para *citizen-developers*. Ao nível de solicitar feedback de utilizadores, objetivo em comum em ambas as abordagens das plataformas, estas disponibilizam ações que permitem que os utilizadores submetam o seu feedback quando interagem com as aplicações (web ou móvel).

6.3 Arquitetura de soluções

De um modo geral, ambas as plataformas conseguiram responder às necessidades previstas nos casos de estudo, apresentando funcionalidades que, embora diferentes, permitiram um desenvolvimento semelhante.

6.3.1 Adequabilidade da arquitetura

Ao nível da adoção de uma arquitetura monolítica combinada com arquitetura em camadas verificou-se que ambas as plataformas permitiram, com relativa facilidade, a implementação de soluções baseadas nesta abordagem. Apesar disso, as plataformas apresentam uma estrutura de aplicação própria e que diferem entre si. Enquanto em Outsystems as aplicações apresentam uma estrutura em componentes, tendo cada um deles uma responsabilidade específica, em Mendix as aplicações apresentam um conjunto de definições e um modelo de domínio. Nesta perspetiva, apesar da estrutura em Outsystems poder indicar uma menor capacidade de adaptar a solução a outras abordagens, esta apresenta-se intuitiva e fácil de utilizar, permitindo identificar claramente as camadas de uma solução. Por outro lado, em Mendix há uma maior flexibilidade para estruturar a solução. No entanto, esta abordagem não se torna tão direta e intuitiva, à semelhança do que em Outsystems.

O caso de estudo 1 considerava uma arquitetura alternativa onde se esperava que o componente responsável pela apresentação comunicasse com o servidor através de pedidos HTTP. Neste sentido, verificou-se que apesar de ambas as plataformas permitirem considerar essa abordagem, a estrutura de aplicações e módulos disponibilizada pela Outsystems poderia favorecer o seu desenvolvimento. Isto porque apresenta a possibilidade de isolar a lógica de negócio e base de dados num módulo do tipo *Service*. Relativamente à abordagem SSR ou SPA ambas as plataformas seguem a segunda abordagem, considerada a mais moderna. Contudo, alternativamente, em Outsystems é, ainda, possível implementar aplicações SSR através de um dos tipos de aplicação mais antigos da plataforma.

O caso de estudo 2 permitiu analisar o comportamento das plataformas na alteração da arquitetura das soluções para uma arquitetura orientada a microserviços, bem como entender as diferenças, se existissem, para o desenvolvimento de uma aplicação móvel. Apesar das plataformas não disponibilizarem um mecanismo para migração de arquiteturas, a adaptação das soluções para a nova arquitetura foi simples e permitiu que fossem reaproveitados os

componentes desenvolvidos anteriormente, através do *copy-paste* destes. Ao nível do desenvolvimento da aplicação móvel não se detetaram diferenças significativas nas ações ou componentes desenvolvidos. A principal particularidade deste tipo de aplicação foi verificada ao nível das suas configurações, uma vez que estas podem ser publicadas como PWA ou configuradas para aplicações móveis nativas. Ambas as plataformas disponibilizam estes dois tipos de configuração sendo que, a principal diferença está ao nível das aplicações móveis nativas. Neste caso, a Outsystems gera um *package* da aplicação e a Mendix permite a construção destas aplicações a partir da aplicação *Make It Native*, disponibilizada pela plataforma.

As plataformas permitiram a adoção de uma arquitetura orientada a microserviços, contudo, cada uma delas apresenta algumas particularidades, sobretudo ao nível da publicação de APIs. Enquanto a Outsystems considera, adicionalmente, uma extensão para obter ações de pedidos HTTP, em relação à implementação do caso de estudo 1, Mendix considera a implementação de ações que não tinham sido consideradas na implementação anterior (e.g. *message definitions* e objetos *HTTPResponse* ou *HTTPHeader*). Apesar das diferenças na implementação, verificou-se que a documentação das APIs disponibilizada é semelhante nas duas plataformas, apresentando, para além das interfaces gráficas, o ficheiro *swagger.json*. Ao nível da decomposição do sistema em microserviços foi possível verificar que Outsystems sugere uma a adoção de uma arquitetura SOA. Esta sugestão complementa-se com a estrutura fornecida pela plataforma. Apesar de não se pretender uma arquitetura SOA esta abordagem da plataforma facilitou a adoção da arquitetura orientada a microserviços. Por outro lado, Mendix não sugere uma abordagem deste tipo, o que pode favorecer a flexibilidade para adotar outras estruturas, mas que neste caso, não facilita a compreensão eficaz e a implementação de microserviços.

A experiência da construção de soluções com arquiteturas diferentes permitiu verificar que, embora as plataformas suportem as duas arquiteturas, quando adotam uma arquitetura orientada a microserviços deixam de beneficiar das suas principais características de plataformas de desenvolvimento rápido. Estas características correspondem, por exemplo, à rápida implementação de interfaces gráficas com recurso a ações que aceleram o acesso aos dados. Com a decomposição de microserviços toda a informação passa a ser consumida através de pedidos à API. Contudo, note-se que esta limitação pode ser transversal a outras tecnologias ou ferramentas, tradicionalmente utilizadas, cuja arquitetura de uma solução determina vantagens ou limitações da sua utilização. A limitação identificada, juntamente com a

experiência da construção destas soluções levantaram algumas dúvidas ao nível da adoção de uma arquitetura diferente da arquitetura esperada e/ou estabelecida nas plataformas. Isto porque, por exemplo, a Outsystems disponibiliza orientações para uma arquitetura, que é reforçada com a própria estrutura pré-definida nas aplicações. Estas orientações podem associar-se ao facto de as plataformas pretenderem que *citizen-developers* sejam capazes de participar no desenvolvimento de aplicações, pelo que a definição de uma arquitetura, de forma direta ou indireta (no caso de Mendix), poderá facilitar o desenvolvimento. Por outro lado, a escolha destas plataformas por equipas experientes requer uma reflexão sobre os principais benefícios na adoção de uma arquitetura, sempre alinhada com os principais objetivos e necessidade de um projeto.

6.3.2 Vendor Lock-in

Um dos principais aspetos que leva à não adoção deste tipo de plataformas relaciona-se com o *lock-in* dos fornecedores (cf. 3.3 Adoção de plataformas *low-code*). Neste projeto a utilização de licenças gratuitas não permitiu experienciar as funcionalidades que as plataformas disponibilizam e que, na sua perspetiva, evitam o *lock-in*.

Neste sentido, a (Outsystems, 2021e) prevê que em caso de cancelamento de uma subscrição seja possível recuperar a última versão do código-fonte gerado para aplicações, pode este ser alterado e executado em ferramentas externas à plataforma (e.g. *Visual Studio*). Para além disso, podem ser recuperados os últimos dados da aplicação, na última versão do *schema* da base de dados. Esta plataforma prevê que estes dados possam ser migrados ou integrados para outras aplicações externas à plataforma.

Por outro lado, (Mendix, 2021f) prevê a possibilidade de exportação dos modelos da plataforma, sendo possível, através do Mendix SDK automatizar uma migração para plataformas *low-code* ou plataformas como Java. Esta ferramenta permite a geração de código Java a partir de modelos da aplicação. Neste sentido, a plataforma disponibiliza documentação dos modelos utilizados para definir uma aplicação (Mendix Model SDK, 2021). Ao nível dos dados de uma aplicação, a plataforma prevê que se possa descarregar uma cópia de segurança da *Mendix Cloud*, sendo possível armazenar estes dados ou criar uma nova base de dados com estes dados.

Ao nível da exportação dos dados da aplicação as duas plataformas apresentam opções semelhantes. Por outro lado, ao nível de exportação de modelos ou geração de código estas

apresentam opções diferentes. A Outsystems prevê a geração de código das aplicações, a partir da última versão destas, por outro lado, Mendix prevê a exportação dos seus modelos, sendo necessário considerar o esforço de conversão destes em código. Neste sentido, a Outsystems apostou numa solução mais rápida e direta, garantindo aos seus utilizadores que, caso cancelem o vínculo com eles, podem recuperar a última versão das suas aplicações. A questão do *lock-in* pode limitar a decisão de adoção destas plataformas e, desta forma, é necessário avaliar as vantagens e os riscos nesta adoção, tendo sempre em conta os objetivos e necessidades de negócio.

6.4 Testabilidade

No desenvolvimento das soluções deste projeto foi possível experimentar e analisar a oferta das plataformas ao nível de testes de qualidade.

6.4.1 Outsystems

Em Outsystems recorreu-se ao componente *BDD Framework* (Outsystems, 2020) para implementar testes de qualidade. Estes testes foram implementados em novas aplicações criadas para o efeito onde, a partir do componente de testes, foi possível definir cenários de testes e desenvolver a lógica necessária. Para o caso de estudo 1 recorreu-se a esse componente para realizar testes de componentes (designação da Outsystems para testes unitários) e testaram-se as principais ações implementadas. Neste caso, a gestão de dependências disponibilizada pela plataforma permite o isolamento dos testes, facilitando a chamada aos componentes desenvolvidos. Para o caso de estudo 2 recorreu-se ao mesmo componente, mas implementaram-se testes de integração e, para tal, consumiram-se as APIs e definiram-se as estruturas para auxiliar no desenvolvimento destes testes. Tal como se pode verificar, esta plataforma disponibiliza um componente que permite a implementação de testes de componentes ou testes unitários e testes de integração. Esta plataforma fornece também orientações para a implementação de testes funcionais. Ao nível de ferramentas para testes a plataforma fornece outros componentes destinados a testes e, para além disso, disponibiliza informação sobre ferramentas externas que podem ser consideradas para testes de qualidade tais como, por exemplo (Outsystems, 2019b): *Ghost Inspector*, *Katalon*, *Tricentis Tosca* ou *JMeter*.

6.4.2 Mendix

Em Mendix recorreu-se ao módulo *Unit Testing* (Mendix, 2021g) e restantes módulos dependentes para implementar testes de qualidade. Estes módulos foram adicionados às soluções implementadas, permitindo a utilização de ações para implementar testes unitários. Uma das principais particularidades do módulo disponibilizado é fornecer uma interface que permite executar os testes da aplicação e consultar os seus detalhes. Para além deste tipo de teste, a plataforma orienta os desenvolvedores para testes a partir de ferramentas externas, tais como *SoapUI*, *TestNG* e *Selenium IDE* (Mendix, 2021g). Fornece, adicionalmente, o *Mendix Application Test Suite* (ATS) que corresponde a um conjunto de ferramentas para a incorporação de testes automatizados no ciclo de vida de uma aplicação, sendo estas construídas pelo CLEVR em Mendix sobre o *Selenium* (Mendix, 2021g).

6.4.3 Sumário

A implementação destes testes permitiu reforçar a importância de se implementarem ações de servidor ou *microflows* com uma única responsabilidade, de forma a evitar um aumento de complexidade no desenvolvimento de testes. Ambas as plataformas orientam o desenvolvimento para este ser composto em vários fluxos com uma responsabilidade única. Esta abordagem, juntamente com a abordagem considerada desde a definição dos casos de estudo, permitiram facilitar o desenvolvimento de testes de qualidade, ao nível dos dois casos de estudo. De facto, os fluxos destas plataformas permitiram que não se verificassem restrições significativas nos testes dos dois casos de estudo. As plataformas apresentaram diferentes formas de testar, por um lado, Outsystems permite definir cenários, o que favorece o entendimento do objetivo do teste e permite um melhor entendimento das necessidades de negócio. Por outro lado, Mendix apresenta uma abordagem semelhante aos tradicionais testes unitários. A este nível, considera-se que a abordagem seguida por Outsystems pode favorecer os *citizen-developers* (Khorram et al., 2020).

Adicionalmente, esta implementação permite, também, analisar a capacidade destas plataformas adotarem um processo de desenvolvimento TDD. A Outsystems disponibiliza orientações para esta abordagem, considerando que esta pode ser seguida recorrendo aos componentes de testes disponibilizados. Tradicionalmente, este processo começa com a reflexão do comportamento de uma classe, através da definição de testes que permitem

identificar as necessidades para essa classe. As plataformas em estudo não apresentam classes, mas apresentam módulos, assim, os testes nesta abordagem devem permitir definir a composição e comportamento destes. Nas plataformas foi possível verificar que seria possível definir o comportamento de um módulo a partir dos testes, uma vez que os fluxos destes permitem adicionar ações à medida que é necessário. Neste sentido, invés de serem implementadas todas as ações das plataformas e só depois testadas, nesta abordagem definiam-se testes e neste iriam-se definindo as ações e entidades necessárias. Contudo, importa realçar que, no caso de Outsystems, a o componente de testes utilizado teve que ser utilizado numa aplicação *Tradicional Web,* uma vez que noutro tipo não era possível utilizar as interfaces de testes. Neste sentido, seria necessário validar se os restantes componentes disponíveis permitiam a implementação em abordagens mais atuais – *Reactive Web App*.

6.5 Disponibilidade e Escalabilidade

Disponibilidade e escalabilidade são dois requisitos não-funcionais importantes para um funcionamento e uma evolução adequados de aplicações. Nas soluções desenvolvidas, apesar de se terem considerado estes requisitos, verificou-se que as definições nas plataformas, a este nível, são consideradas em subscrições pagas (cf. 6.6 Custos e subscrições). Apesar disso, apresenta-se uma breve descrição que pretende descrever a oferta das plataformas neste âmbito.

6.5.1 Outsystems

A Outsystems suporta uma arquitetura distribuída para manter uma escalabilidade elevada, suportando o equilíbrio de carga e removendo pontos únicos de falha no ambiente de execução, através da configuração de mais servidores *front-end* e da sua adição ao cluster apropriado (Outsystems, 2021f). Para suportar a escalabilidade a plataforma recorre a otimização e gestão automatizada de recursos, garantindo que nenhum leitor, ligação ou transação é deixado em aberto, evitando o erro humano (Outsystems, 2021f). A Outsystems permite escalar verticalmente as suas instalações, aumentando o poder computacional dos *front-ends* e das bases de dados e, consequentemente, melhorar o apoio aos desenvolvedores quando a equipa cresce (Outsystems, 2021f). Por outro lado, podem ser adicionados mais servidores *front-end* em qualquer ambiente de produção, permitindo uma escalabilidade horizontal. Por sua vez,

esta permite aumentar a disponibilidade e aumentar a carga de utilizadores. Para além disso, a (Outsystems, 2021f) permite a conceção de soluções de alta disponibilidade, sendo estas divididas em dois designs: (i) alta disponibilidade localizada, que ao apresentar redundância, pode equilibrar a carga para responder aos eventos existentes e (ii) alta disponibilidade geográfica, que ao apresentar redundância, tanto no centro de dados, como dentro do centro de dados, permite a gestão de eventos do sistema geográfico. Na adoção destas soluções pode-se optar por ter redundância em todos os ambientes ou apenas em alguns, dependendo das necessidades (Outsystems, 2021f).

6.5.2 Mendix

Uma aplicação (Mendix, 2021h) é composta por três componentes: *Mendix Runtime*, base de dados e *Amazon S3 (file) storage*. Ao nível de escalabilidade, as instâncias de *Mendix Runtime* podem ser escaldas horizontalmente, adicionando mais instâncias e verticalmente, adicionando mais memória em cada instância. Não é necessário nenhum esforço adicional para executar uma aplicação em modo cluster, uma vez que o *runtime* não tem estado (Mendix, 2021h). A alta disponibilidade em *Mendix Cloud* assegura um tempo de paragem nulo no caso de uma interrupção do *Mendix Runtime*. A escalabilidade pode ser realizada nos ambientes das aplicações, sendo que no caso de se considerar mais do que uma instância, a aplicação continua a funcionar no caso de uma falhar (Mendix, 2021h). A escalabilidade dos ambientes ocorre no *developer portal*.

6.5.3 Sumário

Tal como se pode verificar, as plataformas asseguram uma oferta idêntica destes aspetos. Por se tratar de aspetos que apresentam um custo, a adoção destes requer uma análise, alinhada com os objetivos e necessidade de um projeto. Para além disso, prevê-se que as plataformas façam um acompanhamento adequado dos projetos dos clientes, de forma a orientá-los nas melhores decisões para as suas necessidades de negócio. No âmbito do domínio dos casos de estudo, a consideração destes aspetos poderia não ser relevante numa primeira fase do desenvolvimento do sistema. Contudo, a evolução deste desenvolvimento e posterior aumento de utilizadores aumentaria a necessidade de se considerarem estes aspetos.

6.6 Custos e subscrições

As plataformas em estudo oferecem diferentes tipos de subscrições, que variam na oferta de funcionalidades e, consequentemente, no custo associado. A (Outsystems, 2021g) disponibiliza subscrições que variam na oferta das seguintes funcionalidades: (i) Capacidade, que inclui limite de aplicações a desenvolver e *end-users* internos e externos; (ii) Deployment, que inclui infraestrutura na *Outsystems Cloud* ou *cloud on-premise* ou privada; (iii) Suporte, que inclui formação e suporte online, bem como acompanhamento do cliente na adoção da plataforma (*Customer Success*); (iv) Configurações da Plataforma, que inclui número de ambientes, pipelines e CI/CD.

A (Mendix, 2021i) disponibiliza subscrições para o desenvolvimento de uma aplicação ou de aplicações ilimitadas. Estas variam, por sua vez variam na oferta das seguintes funcionalidades: (i) *Build*, que inclui desenvolvimento ágil, *low-code* ou *no-code*, *dashboard* de projeto e *App Store* privada; (ii) Deploy, que inclui *Mendix Cloud*, *Private Cloud*, ou *Server-based Deployment*; (iii) Run, que inclui número de ambientes por aplicação, *database tenancy* e escalabilidade vertical e horizontal; (iv) Suporte, que inclui formação e suporte online, bem como acompanhamento do cliente na adoção da plataforma (*Customer Success Manager*).

De uma forma geral, verifica-se que a oferta de funcionalidades nestas subscrições é semelhante, sendo que estas permitem considerar as principais necessidades de um projeto. Neste sentido, é importante considerar que os custos destas subscrições abrangem todas essas necessidades. Uma das principais diferenças na oferta das subscrições relaciona-se com o número de aplicações a desenvolver. Enquanto Outsystems permite uma capacidade ilimitada de aplicações, Mendix direciona a sua oferta para apenas uma aplicação ou número ilimitado. Estas subscrições apresentam diferentes valores, contudo, a este nível não é possível estabelecer uma comparação adequada, uma vez que Mendix apresenta valores mensais, sendo que a estes acrescem valores extra destinados, por exemplo, a considerar um maior número de utilizadores da aplicação. Este cenário reforça a limitação identificada em 3.6.4 Mendix, relativamente à política de preços desta plataforma. Contudo, para além destes custos é importante considerar os custos associados às certificações das plataformas (Outsytems, 2021) (Mendix, 2021j), necessárias para os elementos de uma equipa adquirirem conhecimento e se especializarem. No entanto, considerando que estas plataformas contribuem para uma redução da complexidade técnica existe a possibilidade de citizen developers também desenvolverem

aplicações. Nesta perspetiva, o processo de contratação nas organizações pode tornar-se mais flexível, uma vez que deixará de ser necessário procurar profissionais com competências técnicas muito específicas no mercado. Neste sentido, também os custos associados a estas contratações sofrem impacto com a adoção destas plataformas, uma vez que, este tipo de perfil, contrariamente aos perfis mais técnicos, não exige salários muito elevados. Estes salários elevados surgem, maioritariamente, pela elevada concorrência e procura no mercado.

6.7 Sumário

Ao longo deste capítulo foi apresentada a comparação das plataformas em estudo, tendo em conta vários aspetos. Para além disso, foi possível verificar que estes aspetos nem sempre poderão ser entendidos e/ou implementados por desenvolvedores sem experiência de programação – *citizen developer* – sendo necessário contar com desenvolvedores e/ou engenheiro experientes – *software engineer*. Assim, apresenta-se a Tabela 3 que pretende resumir os aspetos comparados e entender a que tipo de desenvolvedor se destinam.

Tabela 3 Resumo da comparação das plataformas

Item Comparação	Outsystems	Mendix	Desenvolvedor-Alvo
Utilização das Plataformas			
Training e documentação	Sim	Sim	SW Engineer/Citizen developer
Interfaces das plataformas	*User-Friendly*	*User-Friendly*	SW Engineer/Citizen developer
Metodologia de Desenvolvimento			
Processo de desenvolvimento	Ágil	Ágil (Scrum)	SW Engineer/Citizen developer
Suporte de controlo de versões	Sim	Sim	SW Engineer/Citizen developer
Modo de controlo de versões	Automático	Manual	SW Engineer/Citizen developer
Arquitetura de soluções			
Noção/identificação de camada	Maior	Menor	SW Engineer/Citizen developer
Aplicação Web			
SSR	Suporta	Não Suporta	SW Engineer/Citizen developer

Item Comparação	Outsystems	Mendix	Desenvolvedor-Alvo
SPA	Suporta (*Default*)	Suporta (*Default*)	SW Engineer
Suporte à evolução da arquitetura	*Copy-Paste*	*Copy-Paste*	SW Engineer
Conceito de serviço/microserviço	Sim	Não	SW Engineer/Citizen developer
Consumo/Publicação de APIs	Sim	Sim	SW Engineer
Aplicação móvel			
PWA	Suporta	Suporta	SW Engineer/Citizen developer
Aplicações Nativas	Suporta	Suporta	SW Engineer
Lock-in			
Geração de Código	Suporta	Não Suporta	SW Engineer/Citizen developer
Exportação de modelos	Não Suporta	Suporta	SW Engineer
Testabilidade			
Ferramentas de testes	Sim	Sim	SW Engineer/Citizen developer
Possibilidade TDD	Sim	Sim	SW Engineer
Disponibilidade e Escalabilidade			
Escalabilidade Horizontal	Suporta	Suporta	SW Engineer
Escalabilidade Vertical	Suporta	Suporta	SW Engineer
Alta Disponibilidade	Suporta	Suporta	SW Engineer

Tal como se pode verificar na tabela apresentada os aspetos considerados em arquitetura de soluções e disponibilidade e escalabilidade foram onde se verificou mais a necessidade de desenvolvedores mais experientes. Essencialmente, consideram-se citizen-developers como desenvolvedores alvo em aspetos onde se verificou que as plataformas fornecem orientações adequadas, permitindo a sua compreensão por este tipo de perfil. Contudo, em aspetos como consumo ou publicação de APIs verifica-se que, independentemente da sua facilidade, é sempre necessário considerar conceitos básicos de APIs. Da mesma forma, se considerou que, ao nível de *lock-in*, a exportação de modelos, disponibilizada pela Mendix pode ser pouco adequada a *citizen-developers*, dada a complexidade da sua abordagem.

7 Conclusão

Neste trabalho foi realizada uma pesquisa no âmbito de metodologias de desenvolvimento e plataformas *low-code* que contribuiu para a sistematização dos aspetos mais relevantes destes temas, bem como para a especificação do contexto, problema e casos de estudo deste projeto. A especificação destes casos de estudo permitiu definir os requisitos e arquitetura dos sistemas pretendidos. Por sua vez, desenvolveram-se estes sistemas nas plataformas em estudo, sendo que, esta experiência contribuiu para se estabelecer a comparação entre elas. Pretende-se, agora, responder às questões de investigação (cf. 1.3 Objetivos), bem como analisar o cumprimento dos objetivos deste projeto, as limitações, trabalho futuro e considerações finais.

7.1 Resposta às questões de investigação

Com esta comparação foi possível refletir sobre as questões de investigação definidas para este trabalho, cujas respostas se apresentam nas subsecções seguintes.

7.1.1 Adequabilidade e limitações das ferramentas

Relativamente à questão de investigação:

> **"Q1:** Qual a adequabilidade das ferramentas *low-code* para suportar o desenvolvimento de aplicações com diferentes arquiteturas (e.g. monolítica e orientada a serviços)? E, por outro lado, quais são as principais limitações na adoção de cada uma dessas ferramentas?"

e de acordo com o descrito na secção 6.3.1 Adequabilidade da arquitetura verificou-se que, embora as plataformas suportem a adoção destas arquiteturas, quando se segue uma arquitetura orientada a microserviços deixa de se usufruir das principais características destas plataformas, que permitem um desenvolvimento rápido. Desta análise surgiram algumas dúvidas sobre os benefícios em adotar arquiteturas diferentes das arquiteturas base disponibilizadas pelas plataformas. Estas dúvidas, no contexto de adoção destas plataformas, devem estar alinhadas com os objetivos e necessidade de um projeto. Isto porque, para projetos de áreas de negócio diferentes de TI a adoção das orientações das plataformas pode tornar o projeto mais simples e fácil de compreender, suportado com a documentação da plataforma. Por outro lado, para projetos que exigem requisitos e arquitetura mais complexos devem considera-se todas as (des)vantagens e desafios que a adoção destas plataformas

trazem para o projeto. As principais limitações destas plataformas encontram-se descritas ao longo do capítulo 6 Comparação das plataformas.

7.1.2 Impacto da adoção de ferramentas *low-code*

Relativamente à questão de investigação:

> **"Q2:** Qual o impacto da adoção de uma ferramenta low-code ao nível de custo, tempo e qualidade em relação à adoção de uma ferramenta de desenvolvimento tradicional?"

pode-se concluir que a construção de soluções nas plataformas *low-code* em estudo permitiram descrever as principais características destas, ao nível de implementação, podendo estas serem comparadas às características e conceitos tradicionalmente conhecidos. Apesar de não se ter avaliado o tempo necessário para a implementação das soluções é possível verificar, ao nível da construção, qual a vantagem ao nível de tempo de desenvolvimento na adoção destas ferramentas. Esta vantagem relaciona-se com a rapidez e simplicidade de algumas tarefas desenvolvidas (e.g. definição de entidades de domínio ou implementação de interfaces gráficas), comparativamente a ferramentas tradicionais. Esta vantagem é reforçada com a análise descrita em 6.1.3 Curva de aprendizagem onde é possível verificar que ao longo da experiência nestas plataformas o esforço estimado em tarefas semelhantes decresce, diminuindo, consequentemente, o tempo de desenvolvimento.

Em termos de qualidade de soluções as plataformas suportam, à semelhança das ferramentas tradicionais, a implementação de testes de qualidade o que permite avaliar as soluções a este nível. Contudo, a este nível, considera-se que estas plataformas, como apresentam orientações das melhores práticas de desenvolvimento, funcionalidades que suportam estas práticas e alertam para principais problemas nas aplicações, podem garantir uma qualidade mais adequada das soluções, em relação a ferramentas tradicionais. De facto, apesar destas poderem apresentar uma maior flexibilidade de desenvolvimento, não garantem que os seus desenvolvimentos são os mais adequados. Os custos associados a estas plataformas diferem de acordo com as funcionalidades pretendidas para um projeto (cf. 6.6 Custos e subscrições). Para além disso, devem considerar-se os custos associados às certificações dos elementos das equipas. Apesar disso, é importante realçar que os custos de subscrição das plataformas consideram um conjunto transversal de funcionalidades necessários ao desenvolvimento de um projeto de software. Tradicionalmente, os custos de desenvolvimento de um projeto estão

separados em vários serviços distintos. Neste sentido, a adoção destas plataformas pode ser vantajosa no sentido em que se estabelece um único contrato com estes fornecedores.

7.1.3 Metodologia e Perfil de Competências

Relativamente à questão de investigação:

> **"Q3:** Tendo em conta os seguintes fatores ao nível organizacional: (i) metodologia de desenvolvimento e (ii) perfil de competências da equipa de desenvolvimento, qual o nível de adaptação requerido para a utilização de uma ferramenta *low-code* no processo de desenvolvimento de uma organização?"

e de acordo com o descrito na secção 6.2 Metodologia de desenvolvimento, ambas as plataformas orientam as equipas para a adoção de uma abordagem ágil, considerando que responde a três objetivos: (i) *time-to-market*; (ii) resposta à mudança e (iii) valor para o utilizador final. Neste sentido, internamente, se as organizações pretendem adotar estas plataformas devem refletir sobre o impacto desta abordagem no processo de desenvolvimento atual. A abordagem de desenvolvimento rápido apresentou-se como uma evolução do tradicional modelo em cascata e, por sua vez, contribui para o aparecimento de abordagens ágeis. Assim, importa refletir sobre a abordagem apresentada em 2.3 Rapid Application Development. Nesta, (McConnell, 1996) defende que seria vantajoso que as organizações definissem, primeiramente, um desenvolvimento eficiente - em termos de custos, esforço e tempo – para, mais tarde, divergirem para um desenvolvimento rápido. Esta abordagem foi apresentada no final dos anos 90, sendo que se considera que hoje as organizações estejam mais conscientes de um desenvolvimento eficiente. Contudo, note-se que organizações que adotem metodologias mais tradicionais poderão ter que considerar a abordagem descrita anteriormente. Por outro lado, espera-se que organizações que adotam metodologias mais atuais consigam adaptar melhor e mais rapidamente o seu processo de desenvolvimento.

O estudo das plataformas e a experiência na construção das soluções permitiu identificar a facilidade ou complexidade das tarefas nas plataformas. Neste caso, a experiência prévia em desenvolvimento permitiu, mais facilmente, compreender os principais conceitos da plataforma. Isto pode indicar que, apesar das plataformas disponibilizarem ações mais fáceis e imediatas, será sempre necessário ter em conta conceitos essenciais para a sua compreensão. Apesar disso, é possível considerar que, no caso de *citizen-developers*, a curva de aprendizagem

pode apresentar um crescimento mais lento, no entanto, isto não invalida que estes, posteriormente, se tornem especialistas nestas plataformas. A consideração deste tipo de perfil, permite tornar o processo de contratação de uma organização mais flexível, uma vez que deixará de ser necessário procurar profissionais com competências técnicas muitos específicas no mercado.

7.2 Limitações

Durante este trabalho surgiram as seguintes limitações, que impactaram o seu desenvolvimento:

- **Pesquisa de informação:** escassez de informação científica sobre plataformas low-code e de informação credível relativa a metodologias de desenvolvimento, mais concretamente de RAD;
- **Licenças gratuitas:** a utilização destas licenças não permitiu a análise e verificação de alguns aspetos das plataformas que seriam relevantes para o estudo (e.g. geração de código);
- **Aprendizagem de duas plataformas:** o estudo de duas plataformas em simultâneo foi desafiante, pois embora possam apresentar funcionalidade semelhantes, apresentam interfaces, conceitos e características diferentes entre si. Por vezes, estes detalhes acabavam por se confundir, ao longo do desenvolvimento;
- **Perfil *citizen-developer*:** este perfil é tipicamente considerado nestas plataformas, contudo, neste trabalho apenas foi possível analisar a adequação deste perfil, tendo em conta a experiência nas plataformas (por um perfil programador).

7.3 Trabalho futuro

Com o desenvolvimento deste trabalho pretende-se auxiliar as organizações e/ou pessoas que pretendem analisar o possível investimento numa plataforma *low-code*, por um lado, pretende-se desafiar a comunidade académica e científica a continuar os estudos neste tema que, embora popular atualmente, carece de estudos e análises que estimulem e suportem a sua adoção. Neste sentido, considera-se que estes estudos podem focar-se, essencialmente, nas razões para a não adoção deste tipo de plataforma (cf. 3.3 Adoção de plataformas *low-code*),

tais como falta de conhecimento de plataformas *low-code*, preocupações com *lock-in* com o fornecedor e preocupações de segurança das aplicações. Estes dois últimos aspetos são, de facto, dos mais críticos e são também os mais desafiantes de investigar. Por outro lado, relativamente à falta de conhecimento destas plataformas, este aspeto pode considerar um estudo que permita analisar a possibilidade de integração destas plataformas no ensino académico (na área de Informática ou outras áreas relevantes), avaliando os principais benefícios e desafios e verificando se esta integração poderia, a longo prazo, aumentar o conhecimento deste tipo de plataforma.

7.4 Considerações finais

Este trabalho permitiu o estudo dos principais conceitos e características de metodologias de desenvolvimento que, por sua vez, permitiu direcionar o trabalho para o estudo de plataformas *low-code*. Para além disso, este trabalho permitiu a definição de casos de estudo e a sua construção nas plataformas. Esta construção resultou numa experiência pessoal nessas plataformas. Esta experiência contribuiu para se estabelecer uma comparação das plataformas e para responder às questões de investigação definidas neste trabalho.

O desenvolvimento deste trabalho foi desafiante desde a pesquisa de informação relevante sobre o tema. A escassez de informação levou a que fosse necessário expandir a pesquisa para metodologias de desenvolvimento, no sentido de ser entender o contexto de RAD com as plataformas *low-code*. Esta pesquisa permitiu definir questões de investigação que conduziram este trabalho. A construção de soluções nestas plataformas também foi desafiante, uma vez que foi necessário ter bem presente os objetivos destas e o seu contributo para o trabalho. A experiência pessoal nestas plataformas aumentou a curiosidade de se entender ou pesquisar mais sobre determinadas características, no entanto, estas acabavam, muitas vezes, por não acrescentar valor ao trabalho.

Apesar dos desafios ao longo deste trabalho cumpriram-se os principais objetivos propostos neste trabalho. Numa perspetiva pessoal, este trabalho permitiu adquirir conhecimento sobre este tipo de plataformas, bem como a experiência de desenvolvimento nestas. Sendo que, a este nível, a experiência foi agradável e permitiu entender as principais razões que levam à adoção deste tipo de plataforma. Apesar da experiência interessante nas duas plataformas, a Outsystems destacou-se, desde logo, pela agradável ferramenta que disponibiliza, pela

simplicidade das ações e pela estrutura das suas aplicações que melhorou a experiência no desenvolvimento. Por outro lado, Mendix não apresentou o mesmo destaque, desde logo, pelo ambiente de desenvolvimento ser um pouco confuso e por, em várias ações ser muito pouco intuitivo o que acaba por dificultar o desenvolvimento.

Por fim, este permitiu verificar que a adoção de ferramentas low-code não representa apenas uma mudança tecnológica, mas uma mudança na forma como as organizações desenvolvem soluções impactando, por exemplo, o planeamento e organização de equipas, relação com o cliente, metodologias de desenvolvimento e custos. Apesar dos desafios associados, considera-se que, futuramente, existirão cada vez mais organizações a adotarem este tipo de plataforma, isto porque os desafios dos projetos desenvolvidos de forma tradicional têm imposto a necessidade de novas formas de desenvolver aplicações mais rapidamente. Por outro lado, este crescimento leva a um aumento de procura, tendo os fornecedores destas plataformas de estar a par das necessidades dos seus clientes e continuar a investigar possíveis melhorias e evoluções nas suas plataformas.

Referências

Agile Business, C., 2014. DSDM Agile Project Framework Handbook [WWW Document]. URL https://www.agilebusiness.org/page/ProjectFramework_00_welcome (accessed 2.15.21).

Ahimbisibwe, A., Cavana, R., Daellenbach, U., 2015. A contingency fit model of critical success factors for software development projects: A comparison of agile and traditional plan-based methodologies. J. Enterp. Inf. Manag. 28, 7–33. https://doi.org/10.1108/JEIM-08-2013-0060

Alexander, F., 2021. What Is Low-Code? [2021 Update] [WWW Document]. URL https://www.outsystems.com/blog/posts/what-is-low-code/ (accessed 2.20.21).

Amazon, 2021. Amazon API Gateway | Gerenciamento de APIs | Amazon Web Services [WWW Document]. Amaz. Web Serv. Inc. URL https://aws.amazon.com/pt/api-gateway/

Balaji, S., 2012. WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC. . Vol. 5.

Beck, K., 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley Professional.

Beynon-Davies, P., Carne, C., Mackay, H., Tudhope, D., 1999. Rapid application development (RAD): an empirical review. Eur. J. Inf. Syst. 8, 211–223. https://doi.org/10.1057/palgrave.ejis.3000325

Boehm, B.W., 1988. A Spiral Model of Software Development and Enhancement. Computer 21, 61–72. https://doi.org/10.1109/2.59

Boussaïd, I., Siarry, P., Ahmed-Nacer, M., 2017. A survey on search-based model-driven engineering. Autom. Softw. Eng. https://doi.org/10.1007/s10515-017-0215-4

Brambilla, M., Cabot, J., Wimmer, M., 2012. Model-Driven Software Engineering in Practice, Synthesis Lectures on Software Engineering. https://doi.org/10.2200/S00441ED1V01Y201208SWE001

C4 Model, 2021. The C4 model for visualising software architecture [WWW Document]. URL https://c4model.com/

Cuba Platform, 2018. Classification of Development Frameworks for Enterprise Applications – CUBA Platform [WWW Document]. URL https://www.cuba-platform.com/blog/classification-of-development-frameworks-for-enterprise-applications/ (accessed 2.19.21).

Despa, M.L., 2014. Comparative study on software development methodologies 20.

Ebbinghaus, H., 1885. Memory: A Contribution to Experimental Psychology. Ann. Neurosci. 20, 155–156. https://doi.org/10.5214/ans.0972.7531.200408

Fowler, M., 2015. MicroservicePremium [WWW Document]. martinfowler.com. URL https://martinfowler.com/bliki/MicroservicePremium.html

Gartner, 2021. Definition of DevOps - Gartner Information Technology Glossary [WWW Document]. Gartner. URL https://www.gartner.com/en/information-technology/glossary/devops (accessed 2.25.21).

Gilb, T., 1988. Principles of Software Engineering Management. Addison-Wesley.

Gilb, T., 1985. Evolutionary Delivery versus the “waterfall model.” ACM SIGSOFT Softw. Eng. Notes 10, 49–61. https://doi.org/10.1145/1012483.1012490

Google Trends, 2021. Google Trends [WWW Document]. Google Trends. URL https://trends.google.pt
Hohl, P., Klünder, J., Bennekum, A., Lockard, R., Gifford, J., Münch, J., Stupperich, M., Schneider, K., 2018. Back to the future: origins and directions of the “Agile Manifesto” – views of the originators. J. Softw. Eng. Res. Dev. 6. https://doi.org/10.1186/s40411-018-0059-z
IBM, 2021. What are microservices? [WWW Document]. URL https://www.ibm.com/cloud/learn/microservices (accessed 6.25.21).
IBM, 2020. What are Message Brokers? - United Kingdom | IBM [WWW Document]. URL https://www.ibm.com/uk-en/cloud/learn/message-brokers#toc-o-que--um--46XZOV4S
IBM, 2009. Rational Application Developer V7.5 Programming Guide 1412.
IBM, C., 2007. The IBM Rational Unified Process for System z.
Ibraigheeth, M., Fadzli, S., 2019. Core Factors for Software Projects Success. JOIV Int. J. Inform. Vis. 3. https://doi.org/10.30630/joiv.3.1.217
Jones, T., King, S.F., 1998. Flexible systems for changing organizations: implementing RAD. Eur. J. Inf. Syst. 7, 61–73. https://doi.org/10.1057/palgrave.ejis.3000289
Kerr, J., Hunter, R., 1994. Inside RAD: How to Build Fully Functional Computer Systems in 90 Days or Less. McGraw-Hill Inc.,US.
Khorram, F., Mottu, J.-M., Sunyé, G., 2020. Challenges & opportunities in low-code testing | Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings [WWW Document]. URL https://dl.acm.org/doi/10.1145/3417990.3420204
Koen, P., Ajamian, G., Boyce, S., Clamen, A., Fisher, E., Fountoulakis, S., Johnson, A., Puri, P., Seibert, R., 2002. Fuzzy Front End: Effective Methods, Tools,and Techniques.
Koen, P., Bertels, H., Kleinschmidt, E., 2014a. Managing the Front End of Innovation—Part I.
Koen, P., Bertels, H., Kleinschmidt, E., 2014b. Managing the Front End of Innovation—Part II.
Kruchten, P., 1995. Architectural Blueprints --- The “4+1” View Model of Software Architecture [WWW Document]. URL https://www.researchgate.net/publication/280113621_Architectural_Blueprints_---_The_41_View_Model_of_Software_Architecture
Lefort, B., Costa, V., 2020. Benefits of Low Code Development Environments on Large Scale Control Systems. Proc. 17th Int. Conf. Accel. Large Exp. Phys. Control Syst. ICALEPCS2019, 6 pages, 1.586 MB. https://doi.org/10.18429/JACOW-ICALEPCS2019-WEDPR02
Likert, R., 1932. A technique for the measurement of attitudes. Arch. Psychol. 22 140, 55–55.
Martin, J., 1991. Rapid Application Development. New York : Macmillan Pub. Co. ; Toronto : Collier Macmillan Canada ; New York : Maxwell Macmillan International.
Matharu, G.S., Mishra, A., Singh, H., Upadhyay, P., 2015. Empirical Study of Agile Software Development Methodologies: A Comparative Analysis. ACM SIGSOFT Softw. Eng. Notes 40, 1–6. https://doi.org/10.1145/2693208.2693233
McConnell, S., 1996. Rapid Development: Rapid Devment _p1. Microsoft Press.
McKinsey, 2020. COVID-19 digital transformation & technology | McKinsey [WWW Document]. URL https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/Strategy%20and%20Corporate%20Finance/Our%20Insights/How%20COVID%2019%20has%20pushed%20companies%20over%20the%20technology%20tipping%20point%20and%20transformed%20business%20forever/How-COVID-19-has-pushed-companies-over-the%20technology%20tipping-point-final.pdf?shouldIndex=false (accessed 2.11.21).

Mendix, 2021a. Low-code Application Development Platform - Build Apps Fast & Efficiently | Mendix [WWW Document]. URL https://www.mendix.com/
Mendix, 2021b. Mendix Runtime - Studio Pro 9 Guide [WWW Document]. Mendix Doc. URL https://docs.mendix.com/refguide/runtime
Mendix, 2021c. Progressive Web Apps - Studio Pro 9 Guide | Mendix Documentation [WWW Document]. URL https://docs.mendix.com/refguide/progressive-web-app (accessed 10.6.21).
Mendix, 2021d. Mendix Cloud Security - Controls, Backups, Encryption, Logging & Audit Trails | Mendix Evaluation Guide [WWW Document]. URL https://www.mendix.com/evaluation-guide/enterprise-capabilities/cloud-architecture/
Mendix, 2021e. Guide to Agile Methodology – Scrum Team, Agile Ceremonies, FAQ [WWW Document]. Mendix. URL https://www.mendix.com/agile-guide/
Mendix, 2021f. Avoid Being Locked In to the Mendix Platform | Mendix Evaluation Guide [WWW Document]. Mendix. URL https://www.mendix.com/evaluation-guide/enterprise-capabilities/no-vendor-lock-in/
Mendix, 2021g. Testing - Studio Pro 9 How-to's | Mendix Documentation [WWW Document]. URL https://docs.mendix.com/howto/testing/
Mendix, 2021h. Mendix Cloud Overview - Architecture, Regions & High Availability [WWW Document]. Mendix. URL https://www.mendix.com/evaluation-guide/app-lifecycle/mendix-cloud-overview/
Mendix, 2021i. Mendix Application Development Platform Pricing [WWW Document]. Mendix. URL https://www.mendix.com/pricing/
Mendix, 2021j. Mendix Academy - Certifications [WWW Document]. URL https://academy.mendix.com/link/certifications
Mendix Model SDK, 2021. Mendix Model SDK [WWW Document]. URL https://apidocs.rnd.mendix.com/modelsdk/latest/index.html
Microsoft, 2021. O que é o Power Apps? - Power Apps [WWW Document]. URL https://docs.microsoft.com/pt-pt/powerapps/powerapps-overview (accessed 2.21.21).
Natarajan, H., Somasundaram, R.K., Lakshmi, K., 2013. A Comparison Between Present and Future Models Of Software Engineering 10, 4.
Neis, P., Wehrmeister, M.A., Mendes, M.F., 2019. Model Driven Software Engineering of Power Systems Applications: Literature Review and Trends. IEEE Access 7, 177761–177773. https://doi.org/10.1109/ACCESS.2019.2958275
Nilsson, A., Wilson, T.L., 2012. Reflections on Barry W. Boehm's "A spiral model of software development and enhancement." Int. J. Manag. Proj. Bus. 5, 737–756. https://doi.org/10.1108/17538371211269031
Osterwalder, A., Smith, A., Pigneur, Y., Bernarda, G., 2015. Criar Propostas de Valor.
Oustystems, 2018. Getting started [WWW Document]. OutSystems. URL https://success.outsystems.com/Documentation/11/Getting_started (accessed 2.24.21).
Outsystems, 2021a. Build Applications Fast, Right and For the Future | OutSystems [WWW Document]. URL https://www.outsystems.com/
Outsystems, 2021b. OutSystems 11 - OutSystems [WWW Document]. Outsystems Doc. URL https://success.outsystems.com/Documentation/11
Outsystems, 2021c. What Is a Single Page Application (SPA)? [WWW Document]. URL https://www.outsystems.com/blog/posts/single-page-application/

Outsystems, 2021d. Agile Methodology Course - Training | OutSystems [WWW Document]. URL https://www.outsystems.com/training/courses/83/agile-methodology/ (accessed 10.14.21).
Outsystems, 2021e. Standard architecture with no lock-in | Evaluation Guide | OutSystems [WWW Document]. URL https://www.outsystems.com/evaluation-guide/standard-architecture-with-no-lock-in/
Outsystems, 2021f. High availability and scalability strategies - OutSystems [WWW Document]. URL https://success.outsystems.com/Support/Enterprise_Customers/High_availability_and_scalability_strategies
Outsystems, 2021g. OutSystems Pricing & Editions [WWW Document]. URL https://www.outsystems.com/pricing-and-editions/
Outsystems, 2020. Your Complete Guide To BDD Testing In OutSystems [WWW Document]. URL https://www.outsystems.com/blog/posts/bdd-testing/
Outsystems, 2019a. The State of Application Development, 2019 - Insurance Industry Report [WWW Document]. URL https://www.outsystems.com/1/state-app-development-insurance/
Outsystems, 2019b. OutSystems Testing Guidelines [WWW Document]. OutSystems. URL https://success.outsystems.com/Documentation/Best_Practices/OutSystems_Testing_Guidelines
Outsytems, 2021. OutSystems Certification program and exams [WWW Document]. URL https://www.outsystems.com/certifications/
Patel, U., Jain, N., 2013. New Idea In Waterfall Model For Real Time Software Development.
Petersen, K., Wohlin, C., Baca, D., 2009. The Waterfall Model in Large-Scale Development. https://doi.org/10.1007/978-3-642-02152-7_29
PMI, 2020. Pulse of the Profession (2020) | PMI [WWW Document]. URL https://www.pmi.org/learning/library/forging-future-focused-culture-11908 (accessed 2.12.21).
Quesenbery, W., 2004. Balancing the 5Es: Usability. Cut. IT J. 17, 4–11.
Research and Markets ltd, 2020. Low-Code Development Platform Market Research Report: By Offering, Deployment Type, Enterprise, Vertical - Global Industry Analysis and Growth Forecast to 2030 [WWW Document]. URL https://www.researchandmarkets.com/reports/5184624/low-code-development-platform-market-research
Richardson, C., 2021. Microservices [WWW Document]. microservices.io. URL http://microservices.io
Rovce, D.W.W., 1970. MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS 11.
Rubin, K.S., 2012. Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley.
Rymer, J.R., 2017. The Forrester Wave™: Low-Code Development Platforms For AD&D Pros, Q4 2017 21.
Rymer, J.R., Koplowitz, R., 2019. The Forrester Wave™: Low-Code Development Platforms For AD&D Professionals, Q1 2019 17.
Saaty, T., 1991. The Analytic Hierarchy Process.
Sahay, A., Indamutsa, A., Ruscio, D.D., Pierantonio, A., 2020. Supporting the understanding and comparison of low-code development platforms, in: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Presented at the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 171–178. https://doi.org/10.1109/SEAA51224.2020.00036

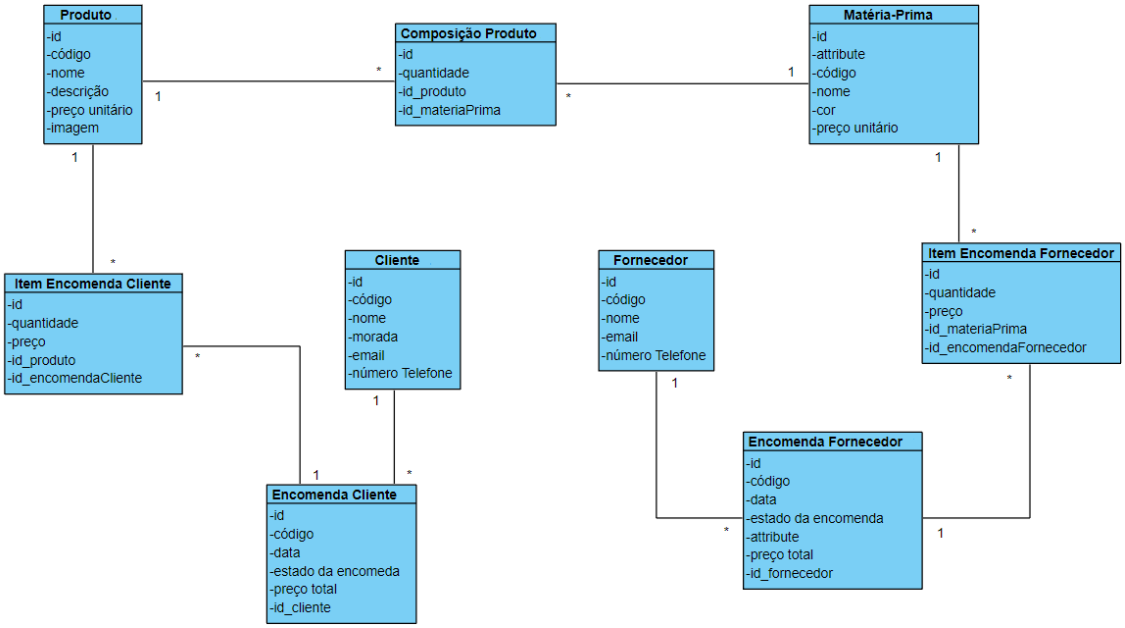
Salesforce, 2021. Low Code [WWW Document]. Salesforce.com. URL https://www.salesforce.com/products/platform/low-code/
Sanchis, R., García-Perales, Ó., Fraile, F., Poler, R., 2020. Low-Code as Enabler of Digital Transformation in Manufacturing Industry. Appl. Sci. 2076-3417 10, 12.
Schmidt, D.C., 2006. Guest Editor's Introduction: Model-Driven Engineering. Computer 39, 25–31. https://doi.org/10.1109/MC.2006.58
Schwaber, K., Sutherland, J., 2020. Scrum Guide | Scrum Guides [WWW Document]. Scrum Guide. URL https://www.scrumguides.org/scrum-guide.html (accessed 2.16.21).
Scrum.org, 2020. The Scrum Framework Poster [WWW Document]. Scrum.org. URL https://www.scrum.org/resources/scrum-framework-poster (accessed 2.16.21).
Selic, B., 2003. The pragmatics of model-driven development. IEEE Softw. 20, 19–25. https://doi.org/10.1109/MS.2003.1231146
Silver, A., 2021. Software development in 2021 and beyond [WWW Document]. Off. Microsoft Blog. URL https://blogs.microsoft.com/blog/2021/01/14/software-development-in-2021-and-beyond/ (accessed 2.12.21).
Simao, E.M., 2011. Comparison of Software Development Methodologies based on the SWEBOK 110.
Tisi, M., Mottu, J.-M., Kolovos, D.S., De Lara, J., Guerra, E.M., Di Ruscio, D., Pierantonio, A., Wimmer, M., 2019. Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms, in: STAF 2019 Co-Located Events Joint Proceedings: 1st Junior Researcher Community Event, 2nd International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems, and 1st Research Project Showcase Workshop Co-Located with Software Technologies: Applications and Foundations (STAF 2019), CEUR Workshop Proceedings (CEUR-WS.Org). Eindhoven, Netherlands.
van Vliet, H., 2007. Software Engineering: Principles and Practice 560.
Vijayasarathy, L.R., Butler, C.W., 2016. Choice of Software Development Methodologies. IEEE Softw. 9.
Vincent, P., 2019. What to Consider Prior to Low-Code Development. Comput. Wkly. 20–23.
Vincent, P., Natis, Y., Iijima, K., Wong, J., Ray, S., Jain, A., Leow, A., 2020. Magic Quadrant for Enterprise Low-Code Application Platforms [WWW Document]. URL https://www.gartner.com/doc/reprints?id=1-24TEFBCB&ct=201214&st=sb (accessed 2.20.21).
Waheed, M.A., S, M., I, S., A, M., K, A., 2018. A Review of Popular Agile Software Development Technologies. J. Inf. Technol. Softw. Eng. 08. https://doi.org/10.4172/2165-7866.1000245
Waszkowski, R., 2019. Low-code platform for automating business processes in manufacturing. IFAC-Pap., 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019 52, 376–381. https://doi.org/10.1016/j.ifacol.2019.10.060
Wood, W.A., Kleb, W.L., 2002. Extreme Programming in a Research Environment, in: Wells, D., Williams, L. (Eds.), Extreme Programming and Agile Methods — XP/Agile Universe 2002, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 89–99. https://doi.org/10.1007/3-540-45672-4_9
Woodall, T., 2003. Conceptualising "Value for the Customer": An Attributional, Structural and Dispositional Analysis.
Zaminkar, M., Reshadinezhad, M., 2013. A Comparison Between Two Software Engineering Processes, RUP And Waterfall Models.
Zeithmal, V., 1988. Consumer Perceptions of Price, Quality, and Value: A Means-End Model and Synthesis of Evidence.

Anexos

Anexo 1 – Taxonomia para plataformas *low-code*

Feature	Description
Graphical user interface	
Drag-and-drop designer	This feature enhances the user experience by permitting to drag all the items involved in making an app including actions, responses, connections, etc.
Point and click approach	This is similar to the drag-and-drop feature except it involves pointing on the item and clicking on the interface rather than dragging and dropping the item.
Pre-built forms/reports	This is off-the-shelf and most common reusable editable forms or reports that a user can use when developing an application.
Pre-built dashboards	This is off-the-shelf and most common dashboards that a user can use when developing an application.
Forms	This feature helps in creating a better user interface and user experience when developing applications. A form includes dashboards, custom forms, surveys, checklists, etc. which could be useful to enhance the usability of the application being developed.
Progress tracking	This features helps collaborators to combine their work and track the development progress of the application.
Advanced Reporting	This features enables the user to obtain a graphical reporting of the application usage. The graphical reporting includes graphs, tables, charts, etc.
Built-in workflows	This feature helps to concentrate the most common reusable workflows when creating applications.
Configurable workflows	Besides built-in workflows, the user should be able to customize workflows according to their needs.
Interoperability support	
Interoperability with external services	This feature is one of the most important features to incorporate different services and platforms including that of Microsoft, Google, etc. It also includes the interoperability possibilities among different low-code platforms.
Connection with data sources	This features connects the application with data sources such as Microsoft Excel, Access and other relational databases such as Microsoft SQL, Azure and other non-relational databases such as MongoDB.
Security Support	
Application security	This feature enables the security mechanism of an application which involves confidentiality, integrity and availability of an application, if and when required.
Platform security	The security and roles management is a key part in developing an application so that the confidentiality, integrity and authentication (CIA) can be ensured at the platform level.
Collaborative development support	
Off-line collaboration	Different developers can collaborate on the specification of the same application. They work off-line locally and then they commit to a remote server their changes, which need to be properly merged.
On-line collaboration	Different developers collaborate concurrently on the specification of the same application. Conflicts are managed at run-time.
Reusability support	
Built-in workflows	This feature helps to concentrate the most common reusable workflows in creating an application.
Pre-built forms/reports	This is off-the-shelf and most common reusable editable forms or reports that a user might want to employ when developing an application.
Pre-built dashboards	This is off-the-shelf and most common dashboards that a user might want to employ when developing an application.
Scalability	
Scalability on number of users	This features enables the application to scale-up with respect to the number of active users that are using that application at the same time.
Scalability on data traffic	This features enables the application to scale-up with respect to the volume of data traffic that are allowed by that application in a particular time.
Scalability on data storage	This features enables the application to scale-up with respect to the data storage capacity of that application.
Business logic specification mechanisms	
Business rules engine	This feature helps in executing one or more business rules that help in managing data according to user's requirements.
Graphical workflow editor	This feature helps to specify one or more business rules in a graphical manner.
AI enabled business logic	This is an important feature which uses Artificial Intelligence in learning the behaviour of an attributes and replicate those behaviours according to learning mechanisms.
Application build mechanisms	
Code generation	According to this feature, the source code of the modeled application is generated and subsequently deployed before its execution.
Models at run-time	The model of the specified application is interpreted and used at run-time during the execution of the modeled application without performing any code generation phase.
Deployment support	
Deployment on cloud	This features enables an application to be deployed online in a cloud infrastructure when the application is ready to deployed and used.
Deployment on local infrastructures	This features enables an application to be deployed locally on the user organization's infrastructure when the application is ready to be deployed and used.
Kinds of supported applications	
Event monitoring	This kind of applications involves the process of collecting data, analyzing the event that can be caused by the data, and signalling any events occurring on the data to the user.
Process automation	This kind of applications focuses on automating complex processes, such as workflows, which can takes place with minimal human intervention.
Approval process control	This kind of applications consists of processes of creating and managing work approvals depending on the authorization of the user. For example, payment tasks should be managed by the approval of authorized personnel only.
Escalation management	This kind of applications are in the domain of customer service and focuses on the management of user viewpoints that filter out aspects that are not under the user competences.
Inventory management	This kind of applications is for monitoring the inflow and outflow of goods and manages the right amount of goods to be stored.
Quality management	This kind of applications is for managing the quality of software projects, e.g., by focusing on planning, assurance, control and improvements of quality factors.
Workflow management	This kind of applications is defined as sequences of tasks to be performed and monitored during their execution, e.g., to check the performance and correctness of the overall workflow.

Anexo 2 – Diagrama de classes



Anexo 3 – Tabelas de avaliação de usabilidade

Critérios de usabilidade

Critérios	Definição
Eficácia	Verifica se o produto/software é útil e ajuda os utilizadores a atingir os objetivos de forma precisa
Eficiência	Verifica a velocidade com que o trabalho pode ser feito de forma individual
Engaging	Define o quão agradável, satisfatório ou interessante pode ser a utilização da interface
Tolerância ao erro	Define a forma como o produto/software previne os erros e ajuda os utilizadores a recuperarem de qualquer erro que ocorra
Fácil de entender	Corresponde à facilidade de aprender a utilizar o produto/software e à forma como este suporta a orientação inicial e uma aprendizagem mais profunda

Escala de Likert

Numeração	Grau de concordância
1	Discordo fortemente
2	Discordo
3	Não concordo nem discordo
4	Concordo
5	Concordo fortemente

Anexo 4 – Tabela de Estimativa de Esforço

Requisito Funcional	**Esforço (segundo sequência de *Fibonacci*)**
UC1	5
UC2	2
UC3	3
UC4	2
UC5	2
UC6	3
UC7	5

Anexo 5 - Análise de Valor

4 Análise de Valor

Este capítulo apresenta uma análise dos cinco elementos de NCD no contexto deste projeto, o valor do projeto para o cliente, bem como a proposta de valor.

4.1 New Concept Development Model

A Figura 82 apresenta o *New Concept Development Model* (NCD) que se divide em três áreas (Koen et al., 2014a):

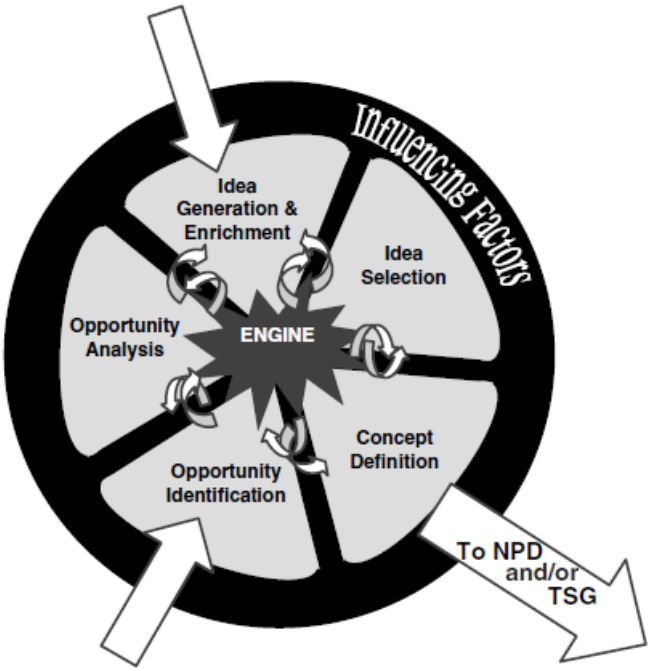


Figura 82 Modelo NCD (Koen et al., 2002)

- *Engine* que impulsiona os cincos elementos de atividade de *front-end* que são controláveis pela organização;
- Interior do modelo que define os cincos elementos de atividade e inclui cinco elementos de atividade (Koen et al., 2002) (Koen et al., 2014b):
    - **Identificação de Oportunidade:** neste elemento a organização identifica oportunidades, através da definição do mercado ou área tecnológica. A

essência deste elemento são as fontes e métodos utilizados para identificar a oportunidade;

- **Análise de Oportunidade:** neste elemento avalia-se a oportunidade, através do estudo de mercado, dos concorrentes e do cliente, de forma a entender o seu potencial;
- **Geração de ideias:** elemento que corresponde ao nascimento, desenvolvimento e amadurecimento de uma ideia concreta, através de processos formais, como *brainstorming* ou fora dos limites formais, como, por exemplo, um pedido de um utilizador;
- **Seleção de ideia:** neste elemento seleciona-se a ideia através de um processo formal ou não, sendo que não existe um único processo que garanta uma boa seleção. Nas organizações a maior dificuldade é selecionar as ideias que têm potencial para contribuir para alcançar um maior valor de negócio;
- **Definição de conceito:** neste elemento analisa-se a viabilidade dos processos, os requisitos técnicos e os fatores económicos relacionados com o projeto.

Fatores que influenciam e moldam os cinco elementos de atividade, tais como capacidades organizacionais, mudanças regulamentares, cliente, tendências mundiais, entre outros. As subsecções que se seguem apresentam a definição dos cinco elementos de atividade de NCD no contexto do presente projeto de dissertação.

4.1.1 Identificação de oportunidade

O estudo realizado por (Ibraigheeth and Fadzli, 2019) verificou, com recurso à literatura existente, que algumas das seguintes características eram comuns aos projetos de software fracassados:

- Qualidade que era pretendida não foi alcançada;
- Falha no cumprimento dos requisitos do cliente;
- Excede o orçamento planeado;
- Projetos atrasados, aumentando em 30% o tempo estimado de conclusão do projeto.

Em qualquer área de negócio e, em particular, nas áreas mais competitivas o fator *time-to-market* revela-se cada vez mais essencial. E isto, numa era cada vez mais informatizada aumenta a necessidade de existirem soluções que suportem os processos e, por conseguinte, na

necessidade de construir ou evoluir soluções mais rapidamente sem comprometer a qualidade das mesmas. E, portanto, as organizações de desenvolvimento de software acabam também por estar pressionadas a uma entrega de soluções cada vez mais rápido, sendo este um fator decisivo de competitividade entre estas.

No momento atual, a pandemia Covid-19 permitiu que as organizações acelerassem a digitalização dos seus processos e interações de 3 a 4 anos (McKinsey, 2020). Para além disso, no inquérito realizado pela (McKinsey, 2020) quase todos os inquiridos afirmaram que as suas organizações adotaram soluções temporárias para responder às novas necessidades mais rápido do que pensariam ser possível antes da crise pandémica.

De facto, o ano de 2020 pressionou as organizações a procurarem soluções rápidas para as suas necessidades e a reinventarem a forma como funcionam (Silver, 2021). Para dar resposta a essas necessidades a Microsoft revela que alguns dos seus clientes, de todas as indústrias – desde a Cruz Vermelha Americana à *Toyota* – recorreram à plataforma *Power* para melhorarem os seus processos, bem como a comunicação e colaboração, permitindo que o seu foco se concentrasse nos problemas estratégicos da organização (Silver, 2021).

Este tipo de plataformas designa-se plataformas de desenvolvimento *low-code*. Estas plataformas baseiam-se em interfaces gráficas, que permitem que pessoas sem conhecimento tecnológicos e de diferentes domínios de conhecimento desenvolvam aplicações com relativa facilidade (Waszkowski, 2019) (Sahay et al., 2020).

Apesar da perspetiva de crescimento do uso destas ferramentas (Vincent et al., 2020) a literatura existente não apresenta estudos significativos e comparativos deste tipo de ferramenta. Apesar das empresas líderes neste mercado apresentarem informação significativa sobre os seus produtos, os potenciais utilizadores destas plataformas não têm como analisar, em termos práticos, as vantagens e desvantagens destas plataformas em relação às ferramentas e processos de desenvolvimento mais comuns e/ou tradicionais.

Desta forma, verifica-se a oportunidade de realizar um estudo que permita a análise e comparação entre soluções, bem como a análise da aplicabilidade deste tipo de ferramenta nas organizações. A inclusão deste tipo de estudo no mercado científico, por um lado, desafia a comunidade científica a dar continuidade a este tipo de trabalho e, por outro lado, auxilia a

investigação de organizações e/ou pessoas que pretendem analisar o possível investimento numa plataforma *low-code*.

4.1.2 Análise de oportunidade

As plataformas de desenvolvimento *low-code* suportam o rápido desenvolvimento, implementação, execução e gestão de aplicações (Vincent et al., 2020). Estas plataformas são fornecidas por organizações conhecidas pela sua oferta *software as a service* (SaaS), pelas suas capacidades de gestão de processos de negócio (BPM) ou pela sua especialização para o desenvolvimento rápido de aplicações (Vincent et al., 2020).

Tal como referido na secção 3.6 Principais plataformas low-code os líderes do mercado de plataformas low-code são: *Salesforce*, *Outsystems*, *Mendix*, *Microsoft*, *ServiceNow*, *Appian* e *Kony* (Vincent et al., 2020) (Rymer and Koplowitz, 2019).

Os fornecedores deste mercado têm evoluído os seus produtos, com funcionalidades mais amplas que requerem equipas mais pequenas e menos especializadas, de forma a facilitar a entrega de aplicações (Vincent et al., 2020). Esta evolução surge como resposta aos desafios enfrentados pelas organizações de TI ao nível da entrega de aplicações (Vincent et al., 2020).

Sobretudo no último ano a popularidade deste tipo de ferramentas tem aumentado devido ao aumento da procura de soluções rápidas, durante a pandemia Covid-19. De facto, a pandemia pressionou as organizações a adotarem soluções digitais num curto espaço de tempo, de forma a responder às necessidades atuais.

Figura 83 Popularidade do termo *low code development plataforms* de 2017 até 2021 (Google Trends, 2021)

A Figura 83 apresenta o gráfico de interesse ao longo do tempo captado no (Google Trends, 2021). O gráfico mostra a evolução da popularidade do termo *low code development plataforms* em todo o mundo, desde o dia 1 de janeiro de 2017 até 1 de fevereiro de 2021. Tal como se

pode desde 2017 há um aumento da popularidade deste termo, sendo que esta popularidade atinge o seu pico em agosto de 2020 e janeiro de 2021.

Segundo a Gartner até 2023 mais de 50% de médias e grandes organizações terão adotado uma plataforma de desenvolvimento *low-code* (Vincent et al., 2020). Ao nível do mercado global de plataformas de desenvolvimento *low-code* (Research and Markets ltd, 2020) prevê que até 2030 estas plataformas gerem uma receita de 187 mil milhões de dólares, prevendo-se um avanço a um ritmo rápido.

Com a transformação digital e o aparecimento de novas tecnologias as organizações de TI têm sido pressionadas a adotar processos e ferramentas que permitam reduzir tempo e recursos, de forma a manterem-se competitivas no mercado (Research and Markets ltd, 2020). A procura de novas formas de desenvolver software surge para reagir ao fracasso de projetos de software. (Ibraigheeth and Fadzli, 2019) cita um estudo realizado por *Verner et al* onde se analisaram 70 projetos falhados, sendo que se concluiu que 93% desses projetos falharam por não cumprirem os prazos de entrega, 81% dos projetos foram subestimados e 73% dos projetos apresentavam equipas sem experiência necessária.

Desta forma, as organizações de IT procuram aumentar a rapidez do desenvolvimento de aplicações, sem pôr em causa a qualidade das mesmas. Consequentemente, as organizações pretendem reduzir os custos resultantes de atrasos nos projetos e reduzir a complexidade técnica dos projetos. A redução desta complexidade permite que seja mais fácil enquadrar equipas nos projetos, reduzindo o impacto da falta de competências técnicas.

Neste sentido, as plataformas *low-code* surgem como uma alternativa para a resolução dos principais desafios destas organizações. Estas plataformas permitem aumentar a rapidez de desenvolvimento em 5 a 10 vezes mais (Sanchis et al., 2020). Este aumento deve-se ao facto destas plataformas fornecerem ferramentas que permitem desenvolver e configurar visualmente as aplicações, ao contrário do processo tradicional de desenvolvimento de código manualmente (Sanchis et al., 2020). Por consequência, estas ferramentas reduzem a complexidade técnica, permitindo que programadores experientes ou *citizen developers* possam, igualmente, desenvolver aplicações. Dessa forma, o processo de contratação nas organizações de IT pode tornar-se mais flexível, uma vez que deixará de ser necessário procurar profissionais com competências técnicas muito específicas no mercado.

A adoção deste tipo de ferramenta numa organização requer, inicialmente, uma análise interna e externa, de forma a validar a aplicabilidade desta nova forma de desenvolver aplicações. Externamente estas organizações apresentam como oportunidade o aumento da procura de desenvolvimento de soluções em várias indústrias, necessidade resultante da transformação digital. A crise pandémica tem permitido acelerar a transformação digital e aumentar a procura por soluções que, num curto espaço de tempo, cheguem ao mercado e deem resposta às necessidades dos clientes. Por outro lado, a principal ameaça para estas organizações é, como já foi referido, o aumento da competitividade, com concorrentes a procurarem novas formas de entregar soluções com qualidade em pouco tempo.

Ao nível interno é necessário que as organizações analisem o impacto da adoção destas ferramentas no seu negócio, tanto ao nível de custo como ao nível da adequação das novas ferramentas para o ciclo de desenvolvimento utilizado atualmente nas organizações e da adequação ao nível do tipo de soluções que desenvolvem. De facto, a adoção deste tipo de ferramentas não representa apenas uma mudança tecnológica, mas uma mudança na forma como as organizações desenvolvem soluções – impactando o planeamento e organização de equipas, relação com o cliente, metodologias de desenvolvimento, entre outras.

4.1.3 Geração de ideias

Através de pesquisa de literatura e discussão de ideias definiram-se as seguintes questões de investigação:

- **Q1:** Qual a adequabilidade das ferramentas *low-code* para suportar o desenvolvimento de aplicações com diferentes arquiteturas (e.g. monolítica e orientada a serviços)? E, por outro lado, quais são as principais limitações na adoção de cada uma dessas ferramentas?
- **Q2:** Qual o impacto da adoção de uma ferramenta *low-code* ao nível de custo, tempo e qualidade em relação à adoção de uma ferramenta de desenvolvimento tradicional?
- **Q3:** Tendo em conta os seguintes fatores ao nível organizacional: (i) metodologia de desenvolvimento e (ii) perfil de competências da equipa de desenvolvimento, qual o nível de adaptação requerido para a utilização de uma ferramenta *low-code* no processo de desenvolvimento de uma organização?

De forma a responder às questões apresentadas verificou-se que seria necessário definir, pelo menos dois casos de estudo, com um domínio específico. Assim, para cada caso de estudo

definido sugeriu-se a implementação de mais do que uma solução, de forma a ser possível realizar uma análise e comparação entre as soluções implementadas. Contudo, para estas soluções surgiram as seguintes alternativas:

- **Solução 1:** Desenvolvimento de duas soluções através de duas ferramentas *low-code*;
- **Solução 2:** Desenvolvimento de duas soluções através de duas ferramentas *low-code* e uma solução através de uma ferramenta/processo de desenvolvimento tradicional;
- **Solução 3:** Desenvolvimento de duas soluções – uma através de uma ferramenta de *low-code* e outra através de uma ferramenta/processo de desenvolvimento tradicional;

4.1.4 Seleção da ideia

De forma a ser selecionada uma das ideias referidas na subsecção anterior recorreu-se ao método *Analytic Hierarchy Process* (AHP). Para tal é necessário, inicialmente, construir a árvore hierárquica de decisão que apresenta o objetivo da decisão, os critérios de decisão e as alternativas, que foram identificadas na subsecção anterior. Ao nível de critérios de decisão definiram-se os seguintes:

- **Tempo:** duração estimada para a finalização do projeto de tese;
- **Adequação:** relação e capacidade de resposta das soluções às questões de investigação previamente definidas;
- **Complexidade:** tendo em conta o número de soluções a desenvolver, bem como o número de ferramentas/tecnologias a utilizar.

A Figura 84 apresenta a árvore hierárquica de decisão para o contexto em análise.

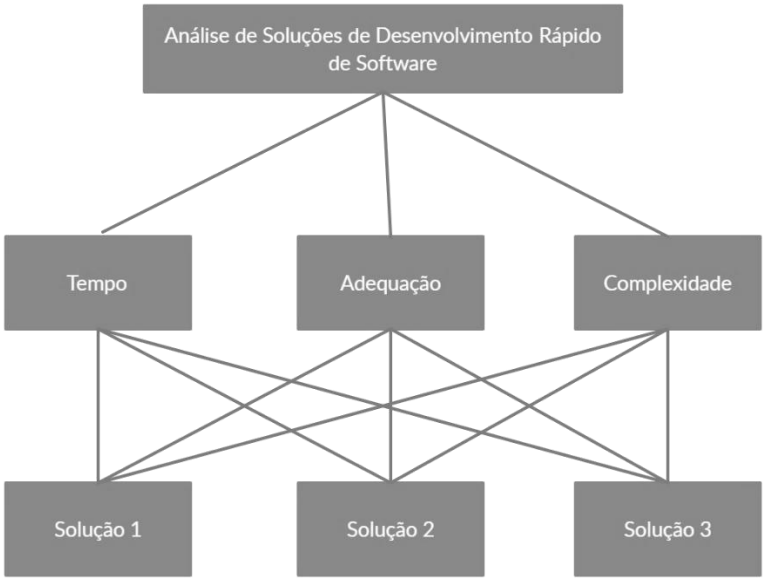


Figura 84 Árvore hierárquica de decisão

Para a atribuição de níveis de importância nas comparações seguiu-se a Escala Fundamental - Níveis de importância e comparação de (Saaty, 1991).

Inicialmente, realiza-se a comparação entre os três critérios definidos que resulta na matriz de comparação apresentada na Tabela 4.

Tabela 4 Matriz de comparação de critérios

	Tempo	**Adequação**	**Complexidade**
Tempo	1	1/5	1/2
Adequação	5	1	5
Complexidade	2	1/5	1
Soma	8	7/5	13/2

Depois normalizam-se os valores desta matriz de comparação, dividindo-se o valor da matriz pelo total da sua coluna. De seguida, calcula-se a média aritmética dos valores de cada linha da matriz, de forma a obter-se o denominado vetor de prioridades, tal como se verifica na Tabela 5.

Tabela 5 Matriz normalizada dos critérios

	Tempo	**Adequação**	**Complexidade**	**Prioridade Relativa**
Tempo	1/8	1/7	1/13	0,11
Adequação	5/8	5/7	10/13	0,70
Complexidade	2/8	1/7	2/13	0,18

Em seguida, de forma a avaliar-se a consistência das prioridades relativas calcula-se a Razão de Consistência (RC), através da fórmula IC= (λmax-n)/(n-1), sendo n=3. O valor de λmax é obtido através dos seguintes cálculos: multiplicação da matriz normalizada com vetor de prioridades; cálculo da média dos valores resultantes da multiplicação com os valores do vetor de prioridades. Assim, o valor de λmax=3.05. Aplicando a fórmula obtém-se o resultado 0,025.

Segue-se o cálculo de RC, sendo RC=IC/0.58=0.025/0.58=0.04. Como 0.04 < 0.1 conclui-se que os valores das prioridades relativas estão consistentes.

Na fase seguinte definem-se as matrizes de comparação para cada critério, tendo em conta cada uma das soluções alternativas identificadas. A Tabela 6, Tabela 7 e Tabela 8 apresentam estas matrizes e o respetivo vetor de prioridade:

Tabela 6 Matriz de comparação para tempo

	Solução 1	**Solução 2**	**Solução 3**	**Prioridade Relativa**
Solução 1	3/5	2/3	3/6	0,59
Solução 2	1/5	2/9	2/6	0,25
Solução 3	1/5	1/9	1/6	0,16

Tabela 7 Matriz de comparação para adequação

	Solução 1	**Solução 2**	**Solução 3**	**Prioridade Relativa**
Solução 1	3/5	9/13	3/7	0,57
Solução 2	1/5	3/13	3/7	0,29
Solução 3	1/5	1/13	1/13	0,12

Tabela 8 Matriz de comparação para complexidade

	Solução 1	**Solução 2**	**Solução 3**	**Prioridade Relativa**
Solução 1	3/5	3/7	9/13	0,64
Solução 2	1/5	1/7	1/13	0,14
Solução 3	1/5	3/7	3/13	0,29

Por último de forma a obter-se a prioridade composta para as alternativas constrói-se uma matriz com as prioridades relativas segundo os critérios e multiplica-se com o vetor de prioridades calculado inicialmente. Deste cálculo obtiveram-se os seguintes resultados: Solução 1 = 0,56, Solução 2 = 0,26 e Solução 3 = 0,15.

Assim, pode verificar-se que a solução 1 apresenta o maior valor, sendo considerada a solução mais indicada. A solução 1 apresentou maiores prioridades relativas nos três critérios. A solução 2, em relação à solução 3 era mais adequada para o contexto do projeto, contudo a solução 3 seria mais adequada tento em conta a complexidade. Uma vez que o critério adequação apresentou um maior peso na prioridade de critérios, a solução 2 é a segunda alternativa mais indicada.

4.1.5 Definição da conceção

Tal como foi referido anteriormente, a escassez de estudos comparativos e críticos da adoção de plataformas *low-code* aumentam as dificuldades de uma pessoa ou organização estudarem esta tecnologia e analisarem o seu potencial investimento. Este estudo pretende analisar duas das plataformas mais promissoras do mercado e estudar a sua aplicabilidade e adequação em termos de arquitetura de uma aplicação/sistema, tipo de projeto, metodologias de desenvolvimento e competências técnicas. Sendo assim, ao nível das plataformas mais promissoras do mercado pretende-se utilizar as ferramentas *Outsystems* e *Mendix*.

Estas plataformas são duas das líderes no mercado e pelo resultado da análise da *Forrester* apresentado na secção 3.6 Principais plataformas low-code, a *Outsystems* e a *Mendix* apresentam a melhor oferta – a primeira é classifica com 4.58 e a segundo com 4.49, numa escala de 0 a 5. O facto de estas serem as melhores classificações a nível de oferta e a sua aproximação foram consideradas relevantes para a escolha destas plataformas. Para além disso, a análise das principais características das plataformas na secção 3.6.5 Comparação das

funcionalidades das plataformas permitiu verificar que as plataformas apresentam características semelhantes. Assim, considera-se que a comparação de soluções em duas plataformas com características e ofertas semelhantes poderá ser interessante, uma vez que, à partida, são esperados comportamentos idênticos, o que poderá desafiar a sua comparação ao nível de segurança e qualidade, por exemplo.

4.2 Proposta de Valor

O valor para o cliente corresponde a uma perceção pessoal, do lado da procura, da vantagem de um cliente com a oferta de uma organização e pode ocorrer através da redução de sacrifícios, presença de benefícios e a combinação de ambos (Woodall, 2003). O *perceived value* é o resultado da análise da utilidade de um produto com base nas perceções do que é recebido (Zeithmal, 1988).

Desta forma, no âmbito deste projeto, é oportuno analisar os benefícios e sacrifícios para o cliente. Sendo que, neste caso, considera-se que as organizações de IT e os seus clientes o público-alvo deste estudo.

Tabela 9 Benefícios e Sacrifícios

Benefícios		**Sacrifícios**
Atributos	**Resultado**	
Q1	Entendimento das principais vantagens e desafios de plataformas *low-code*; Entendimento da adequação de uma arquitetura de aplicação na plataforma *low-code*;	Adaptação e conhecimento de plataformas *low-code*; Entendimento no contexto dos processos de desenvolvimento;
Q2	Entendimento das principais vantagens e desafios das plataformas *low-code* em relação ao processo de desenvolvimento tradicional;	
Q3	Entendimento da aplicabilidade das plataformas tendo em conta fatores como metodologia de desenvolvimento e competências técnicas;	

A Tabela 9 apresenta os benefícios e sacrifícios no âmbito deste estudo. Os atributos considerados correspondem às questões de investigação identificadas e às quais se pretende dar resposta. Os resultados destes atributos correspondem ao entendimento de plataformas *low-code*, bem como a sua aplicabilidade ao nível de metodologias de desenvolvimento e competências técnicas e as principais vantagens e desafios em relação ao processo de desenvolvimento tradicional. Contudo, apesar destes benefícios, existem sacrifícios associados, não monetários, uma vez que se o entendimento e aplicação deste estudo exige um esforço no conhecimento de plataformas *low-code*, bem como no entendimento da sua adaptação em diferentes contextos.

A realização do estudo no âmbito deste projeto de dissertação apresenta, tal como foi referido, benefícios e sacrifícios diretos para o cliente. Contudo, poderão ser considerados benefícios indiretos. Isto é, o estudo propõe a análise e comparação de soluções em plataformas *low-code* e a sua adequação, em vários aspetos, às organizações de TI, contudo, indiretamente o estudo conduz as organizações à adoção ou não deste tipo de plataforma.

Neste sentido, considera-se que a abordagem de adoção e estudo deste tipo de plataforma numa organização será uma abordagem mais interessante na definição da proposta de valor. A Figura 85 apresenta a proposta de valor segundo o modelo de *value proposition* de (Osterwalder et al., 2015), apresentando com **(*)** as atividades, objetivos ou serviços indiretos resultantes do estudo proposto neste projeto.

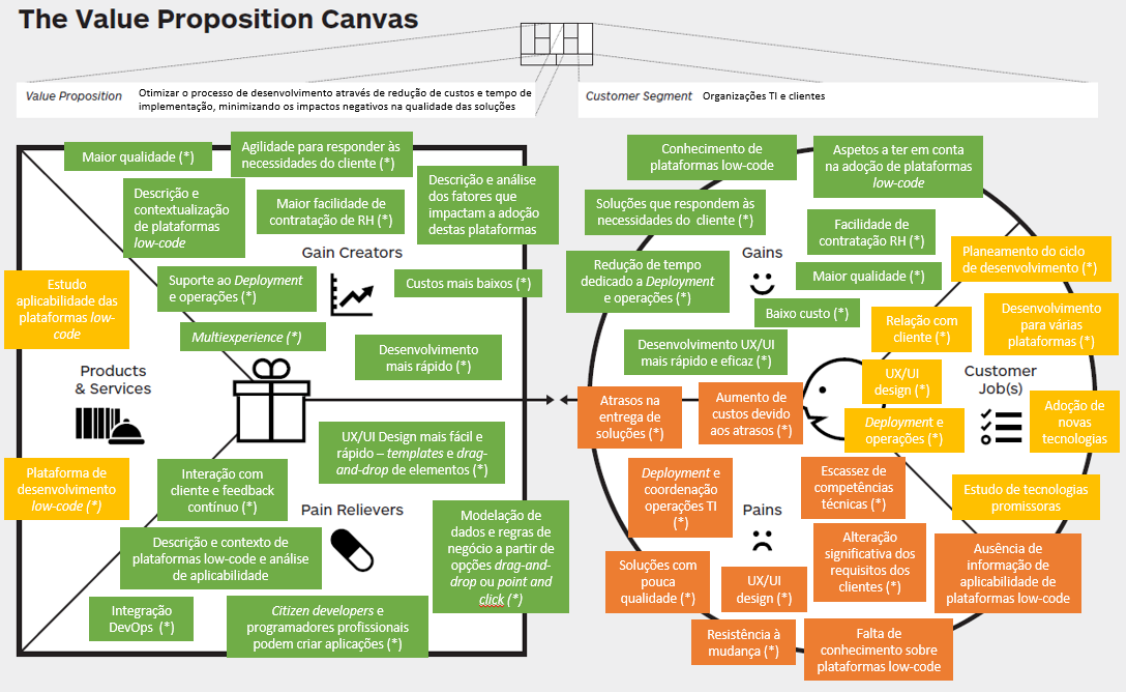


Figura 85 Modelo de proposta de valor segundo modelo de *value proposition* de Osterwalder

A proposta de valor identificada corresponde à otimização dos processos de desenvolvimento através de redução de custos e tempos de implementação, minimizando os impactos negativos na qualidade das soluções. No seguimento desta proposta de valor, o modelo definido apresenta, no lado direito, a abordagem para o cliente – tarefas (*customer jobs*) que serão impactadas com o novo produto/serviço; benefícios (*gains*) que se esperam obter do novo produto/serviço e principais barreiras (*pains*) que atrapalham o cliente na realização das suas tarefas. O lado esquerdo do modelo proposto apresenta os produtos/serviços, os benefícios e ganhos (*gain creators*) que estes fornecem ao cliente, bem como as principais barreiras que o produto ou serviço pretende resolver (*pain relievers*).