INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Modernização do Portal Web Cartão Continente

TIAGO PINTO LOUREIRO MARTINS RIBEIRO
Outubro de 2022

POLITÉCNICO
DO PORTO

Modernização do Portal Web Cartão Continente

Tiago Ribeiro

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

Orientador: Professor Paulo Maio

Porto, 15 de outubro de 2022

Resumo

Para cimentar a sua presença *online*, o Cartão Continente decidiu lançar em 2019 um Portal Web, complementar à aplicação móvel de fidelização, que permite aceder a funcionalidades já disponíveis na aplicação móvel como a visualização de cupões e movimentos, assim como efetuar tarefas administrativas de gestão de conta e cartões. O *Website* construído com recurso à plataforma low-code de desenvolvimento rápido OutSystems conta com cerca de 3000 acessos diários e uma duração média de sessão de 2 minutos.

Na altura em que o site foi lançado, OutSystems apenas permitia a construção de "Traditional Web Apps", modelo de aplicação baseado em ASP .NET Web Forms - tecnologia hoje obsoleta. Desta forma, a solução atual tem dificuldades em responder às expectativas tanto do negócio como dos clientes. Além disso, existem dificuldades crescentes na manutenção da mesma, devido à iminente descontinuação deste modelo aplicacional por parte da OutSystems, assim como a elevada divida técnica presente no produto, o que cria obstáculos no desenvolvimento de funcionalidades.

Neste sentido, este projeto de dissertação estuda as principais plataformas low-code que podem dar resposta ao problema, desenhando e desenvolvendo uma solução nas principais plataformas - OutSystems e Mendix. De modo a perceber as vantagens da solução preconizada em relação à atual, as soluções são analisadas e comparadas de forma individual e em conjunto, usando ferramentas internas a cada plataforma e ferramentas independentes como, por exemplo, *LightHouse*, para avaliar atributos de qualidade e levantar possíveis melhorias às soluções alcançadas.

Deste projeto resulta uma prova de conceito à modernização do Portal Cartão Continente que tem melhor desempenho em todas as métricas avaliadas, a documentação de todo o processo de desenho e desenvolvimento da mesma incluindo alternativas e as limitações encontradas, assim como sugestões de como as ultrapassar.

Palavras-chave: Cartão Continente, Desenvolvimento rápido de aplicações, Low-code, OutSystems, Mendix

Abstract

To cement its online presence, Cartão Continente decided to launch a Web Portal in 2019, complementary to the loyalty mobile application, which allows access to functionalities already available in the mobile application such as the visualization of coupons and movements, as well as perform administrative tasks of account and card management. The website built with the rapid application development tool OutSystems has around 3000 daily accesses and an average session duration of 2 minutes.

At the time the site was launched, OutSystems only allowed the construction of "Traditional Web Apps", an application model based on ASP .NET Web Forms - technology that is obsolete today. As such, the current solution has difficulties in meeting the expectations of both the business and the customers. In addition, there are increasing maintainability issues due to the imminent discontinuation of this application model by OutSystems, as well as the high technical debt present in the product, which creates obstacles in the development of new features.

In that regard, this dissertation project studies the main low-code platforms that address the problem, designing and developing a solution in the main platforms - OutSystems and Mendix. In order to evaluate the advantages of the new solution compared to the current one, the solutions are analyzed and compared individually and together, using tools internal to each platform and independent tools such as, for example, LightHouse, to evaluate quality attributes and raise possible improvements to the solutions achieved.

This project results in a proof of concept for the modernization of the Cartão Continente Portal that performs better in all the metrics evaluated, the documentation of the entire design and development process including alternatives and limitations found, as well as suggestions on how to overcome them.

Keywords: Cartão Continente, Rapid application development, Low-code, OutSystems, Mendix

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos e Siglas

API	Application Programming Interface.
B2B	Business to Business.
B2C	Business to Consumer.
BPM	Business Process Management.
BPMN	Business Process Model and Notation.
CLS	Cumulative Layout Shift.
DSL	Domain Specific Language.
FCP	First Contentful Paint.
hpaPaaS	High-Productivity application Platform as a Service.
HTTP	Hypertext Transfer Protocol.
IDE	Integrated Development Environment.
ISV	Independent Software Vendor.
JSON	JavaScript Object Notation.
LCP	Largest Contentful Paint.
PWA	Progressive Web App.
REST	Representational state transfer.
SaaS	Software as a Service.
SEO	Search Engine Optimization.
SPA	Single Page Application.
SSO	Single Sign-On.
TBT	Total Blocking Time.
TTI	Time To Interactive.
WYSIWYG	What You See Is What You Get.

Capítulo 1

Introdução

Este capítulo introduz o trabalho realizado no âmbito da dissertação de Mestrado, contextualizando e descrevendo o problema a que pretende dar resposta. São também elencados os objetivos e a abordagem seguida para a concretização dos mesmos. Por fim, é apresentada a estrutura do presente documento.

1.1 Contexto

O retalho foi, durante muito tempo, uma área com pouca inovação. Com a introdução do E-commerce nos anos 80, foram abertos novos horizontes e descobertas novas formas de chegar ao consumidor. Este tipo de comércio online permite a expansão das marcas, atingindo um alcance global, uma redução de custo e risco e uma oportunidade única de fazer marketing personalizado (Clack 2020).

Na era digital em que nos encontramos atualmente, os consumidores esperam que grande parte dos serviços estejam disponíveis online, pelo que a presença das marcas nestes meios é vital. De acordo com o relatório 'Online Shopping Statistics' (BigCommerce 2022), em 2020, mais de 80% dos consumidores compraram online e espera-se que 22% de todas as vendas de retalho sejam por este meio em 2023, o que representa um aumento de 8% face a 2019.

A Sonae MC, empresa do grupo Sonae, é líder no setor de retalho alimentar em Portugal. Com mais de 30 anos de experiência no setor, gere um grande número de insígnias como o Continente, Continente Modelo, Continente Online, Bagga, Go Natural, entre outros. A Sonae BIT, empresa ligada à área de sistemas de informação da Sonae MC, é a responsável por criar novas soluções tecnológicas, evoluindo a experiência de retalho (SonaeMC 2021).

Uma das soluções criadas pela Sonae BIT é o Portal Cartão Continente, uma aplicação Web complementar à aplicação móvel, que pretende marcar a presença do Cartão Continente na Web. Este portal contém algumas funcionalidades presentes na aplicação móvel como a visualização de cupões e transações e recuperação de descontos, assim como funcionalidades específicas a este produto como, por exemplo, tarefas mais administrativas de gestão de conta e cartões.

Nascido em 2019, o Portal Cartão Continente conta atualmente com uma média de 3000 acessos diários. As páginas anónimas mais visitadas são a de entrada, a de 'Conheça a aplicação Cartão Continente' e a página de login. Quanto às páginas autenticadas, os utilizadores concentram a sua navegação nas páginas de área pessoal, cupões e movimentos com um tempo de sessão média de cerca de 2 minutos.

Uma vez que um dos principais requisitos do negócio prende-se com a rapidez de entrega e possibilidade de iterações rápidas sobre o produto, a solução foi construída com a plataforma low-code OutSystems. Esta plataforma permite o desenvolvimento de aplicações Web com recurso a pouco ou nenhum código, reduzindo a complexidade de implementação e, consequentemente, acelerando a entrega. Por outro lado, o controlo sobre o código gerado é muito menor pelo que surgem alguns desafios.

1.2 Descrição do problema

O Portal Web atual do Cartão Continente foi desenvolvido em "Traditional Web Apps" de OutSystems que são baseadas em ASP .NET Web Forms - tecnologia introduzida no lançamento da framework ASP .NET em 2002 (Rick-Anderson 2021). Com a introdução da framework .NET Core em 2016, esta tecnologia deixou de ser suportada, o que representa um desafio para a evolução de soluções criadas com base na mesma (Ardalis 2022). Originalmente criada num contexto muito diferente do atual, WebForms tem dificuldades em dar resposta às necessidades atuais do desenvolvimento Web.

Para colmatar estes problemas, OutSystems lançou em 2019 as "Reactive Web Apps", um novo método de criar aplicações Web dentro da plataforma, baseadas num modelo reativo suportado por React, uma biblioteca Javascript que permite criar Single Page Application (SPA) - aplicações que dinamicamente mudam o conteúdo de uma única página em vez de redirecionar para outras, o que resulta numa percepção de desempenho mais elevado e uma experiência mais similar a aplicações móveis (Alves 2019).

Além da barreira tecnológica, que dificulta o desenvolvimento de novas funcionalidades, o Portal atual é também de difícil manutenção uma vez que apresenta elevada dívida técnica.

Visto que o produto a desenvolver encontra-se na área de E-commerce, mais particularmente programas de fidelização, alguns aspetos tornam-se especialmente importantes como, por exemplo, o desempenho, a segurança e a usabilidade. Considerando que cada vez mais os consumidores acedem a estas aplicações a partir dos seus dispositivos móveis (O'Shea 2017), uma das principais considerações é a otimização da experiência para este tipo de equipamentos - uma das insuficiências apontadas ao Portal atual pela equipa de negócio.

O problema passa, portanto, por perceber como o atual Portal pode ser melhorado usufruindo do novo paradigma de programação que as Reactive Apps oferecem, convergindo com as expectativas tanto do negócio como dos utilizadores.

Tratando-se de uma modernização de um produto atualmente em produção, é expectável que as funcionalidades do mesmo sejam mantidas ou melhoradas, sem causar perturbação ao cliente.

1.3 Objetivos

O principal objetivo prende-se com o desenho e desenvolvimento de uma prova de conceito de uma nova versão do Portal Cartão Continente usando Reactive Web Apps de OutSystems. Esta prova de conceito permitirá à equipa de negócio responsável pelo Portal Cartão Continente avaliar as vantagens e viabilidade de fazer esta migração por inteiro.

Numa vertente de investigação e avaliação da solução atingida e de possíveis alternativas, foram também definidos os seguintes objetivos:

- Comparar a plataforma OutSystems com outras plataformas low-code.
- Comparar Traditional e Reactive Web Apps de OutSystems.
- Analisar e levantar possíveis melhorias ao Website gerado pela plataforma OutSystems considerando atributos de qualidade.

1.4 Abordagem

No sentido de perceber a adequabilidade das Reactive Web Apps de OutSystems ao Portal Cartão Continente, será feito um estudo do estado de arte com o objetivo de perceber as vantagens das plataformas low-code face a métodos tradicionais de desenvolvimento, as metodologias tipicamente adotadas no desenvolvimento rápido de aplicações e as principais ferramentas de low-code existentes explorando as suas funcionalidades. Será também feita uma análise e comparação das Traditional Web Apps (que suportam o Portal atual) e das Reactive Web Apps de modo a perceber as vantagens e desvantagens das diferentes abordagens.

Após este estudo, será feito o desenho da solução adotando boas práticas, estilos e padrões de engenharia informática como princípios GRASP e SOLID e modelo C4. Este desenho será independente da tecnologia, não tendo em conta características específicas das plataformas que vão ser usadas para o desenvolvimento, o que irá auxiliar na análise da aplicabilidade das ferramentas à solução que se pretende atingir.

Com base no desenho, serão desenvolvidas provas de conceito nas plataformas OutSystems e Mendix - plataformas líder do mercado low-code - com o objetivo de analisar e comparar o processo de desenvolvimento e a solução obtida por cada um dos produtos. Serão usadas tanto ferramentas internas a cada plataforma como ferramentas independentes como, por exemplo, Lighthouse, para avaliar atributos de qualidade e levantar possíveis melhorias às soluções alcançadas.

1.5 Estrutura do documento

O presente documento encontra-se dividido em sete capítulos:

- **Introdução:** introduz e enquadra o projeto de dissertação, contextualizando e descrevendo o problema a que pretender dar resposta. Apresenta os objetivos e a abordagem seguida para a concretização dos mesmos.
- **Desenvolvimento rápido de aplicações:** apresenta um estudo do estado da arte das metodologias tipicamente adotadas no desenvolvimento rápido de aplicações e as principais ferramentas de low-code existentes no mercado, com o intuito de perceber as situações em que estas plataformas podem ser preferíveis a métodos tradicionais de desenvolvimento.
- **Engenharia de requisitos:** apresenta os requisitos funcionais e não funcionais ligados aos casos de uso escolhidos para a prova de conceito a desenvolver.
- **Desenho da solução:** detalha o processo de desenho da solução. É apresentada a arquitetura proposta para a solução (independente das características específicas das plataformas de desenvolvimento), usando boas práticas de engenharia de software.

- **Implementação:** descreve uma abordagem para a implementação das soluções em OutSystems e Mendix, assim como o seu desenvolvimento segundo a abordagem preconizada.
- **Avaliação da solução:** apresenta o processo de avaliação das soluções implementadas nas diferentes plataformas, seguindo a metodologia proposta.
- **Conclusão:** apresenta um breve sumário do trabalho desenvolvido, os objetivos alcançados, as limitações e possível trabalho futuro.

Capítulo 2

Desenvolvimento rápido de aplicações

Neste capítulo é feito um estudo sobre o estado da arte das metodologias tipicamente adotadas no desenvolvimento rápido de aplicações e as principais ferramentas de low-code existentes no mercado, com o intuito de perceber as situações em que estas plataformas podem ser preferíveis a métodos tradicionais de desenvolvimento.

2.1 Metodologias de desenvolvimento rápido

Com a evolução da área de desenvolvimento de software, novas metodologias têm sido criadas e adaptadas de modo a dar resposta aos novos desafios.

Inicialmente, as principais metodologias eram baseadas num modelo de cascata - modelo de desenvolvimento proposto em 1970 por Royce com fases sequenciais bem definidas: análise de requisitos, desenho, implementação, verificação e manutenção (Royce 1987). Este modelo, principalmente pela sua simplicidade e estrutura bem definida, rapidamente se tornou o padrão da indústria, ajudando a ultrapassar várias dificuldades sentidas na altura. No entanto, com a sua aplicação na prática, foram surgindo várias críticas a este modelo, especialmente pelo seu caráter utópico, argumentando que é demasiado irrealista e impraticável. Os principais problemas apontados são (Patel e Jain 2013):

- Adequação a um projeto real: raramente um projeto segue o fluxo sequencial alvitrado pelo modelo.
- Modelo preditivo: é difícil um cliente prever todas as necessidades. O modelo exige a capacidade de definir tudo à cabeça e não tem capacidade de responder às mudanças.
- Falta de visibilidade no desenvolvimento: uma versão demonstrável do software só estará disponível no final do projeto pelo que, muitas vezes, o cliente não percebe se a solução irá corresponder aos requisitos.

Num contexto moderno de desenvolvimento de software, onde as prioridades do negócio alteram constantemente, um modelo sequencial não é aplicável. É neste sentido que nos anos 80 começam a surgir metodologias alternativas, focadas num processo iterativo e incremental, com ciclos de *feedback* constantes.

De seguida, são apresentadas as metodologias RAD, que introduz muitos dos conceitos base das metodologias atuais, e Scrum, a metodologia de desenvolvimento de software mais popular atualmente.

2.1.1 RAD

Conceptualizado nos anos 80, o conceito de RAD foi formalizado pela primeira vez por James Martin em 1991 no seu livro Rapid Application Development e refere-se a um processo cujo ciclo de vida foi desenhado de forma a proporcionar um desenvolvimento mais rápido e com maior qualidade face a um processo tradicional (Martin 1991).

A filosofia RAD assenta em colocar menos ênfase na etapa de planeamento e, por outro lado, focar-se mais no desenvolvimento iterativo de protótipos. Desta forma, é possível gestores de projeto e equipa de negócio medirem de forma mais precisa o progresso do desenvolvimento, comunicando em tempo real possíveis problemas e alterações, o que resulta num maior alinhamento da visão do produto com o resultado final atingido (Martin 1991).

Esta abordagem tem como objetivo aumentar a velocidade de desenvolvimento, reduzir os custos e aumentar a qualidade do software. Para isso, assenta em alguns aspetos fundamentais (Beynon-Davies et al. 1999):

- Construir protótipos de forma iterativa que vão ser transformados na solução final.
- Usar workshops como forma de reunir requisitos e rever funcionalidades e interações.
- Usar ferramentas CASE para suportar a modelação, protótipo e codificação, automatizando ao máximo estas tarefas.
- Trabalhar em iterações de pequena duração, garantindo refinamento e melhoria contínua.
- Fornecer guias de sucesso e obstáculos a evitar.

Devido ao seu carácter adaptativo, técnicas RAD são muito usadas em áreas em que os requisitos mudam constantemente, uma vez que a sua estrutura é pensada de forma a garantir que a equipa de desenvolvimento constrói exatamente o que os utilizadores estão à espera. De uma forma geral, a metodologia RAD segue 4 fases (LucidChart 2018):

- Planeamento de requisitos: durante esta fase, o cliente apresenta à equipa de desenvolvimento as necessidades, objetivos e expectativas que vão resultar em requisitos. Em conjunto, estes requisitos são analisados, avaliados e validados tendo em conta possíveis problemas e restrições.
- Desenho de utilizador: a equipa de desenvolvimento constrói protótipos de forma iterativa. Durante esta fase, os clientes acompanham de perto todas as fases do desenvolvimento, para validar que as suas expectativas estão a ser cumpridas. O cliente dá o seu parecer, apontando possíveis melhorias até que se chegue a uma solução que satisfaça as necessidades de todos envolvidos.
- Construção rápida: com base nos protótipos construídos na fase anterior, a equipa de desenvolvimento converte estes artefactos num modelo funcional. Esta fase encapsula os processos de preparação, codificação e testes (unitários, integração, sistema). O cliente ainda tem a possibilidade de sugerir alterações. máximo estas tarefas.
- Cutover: fase de lançamento. Já numa fase madura do software, este sofre testes exaustivos de forma a aumentar a segurança do lançamento.

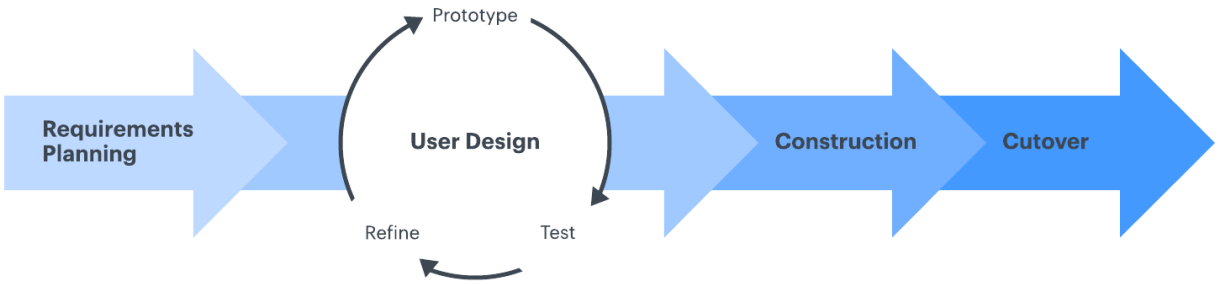

Figura 2.1: Etapas do processo RAD (LucidChart 2018).

Para adotar esta metodologia é importante considerar as suas vantagens e desvantagens. As principais vantagens são (OutSystems 2021d):

- Rapidez: os clientes recebem uma versão funcional do software muito mais cedo do que com metodologias tradicionais.
- Custo: devido ao ciclo de feedback e constante refinamento, o processo de desenho e implementação é muito mais eficiente e eficaz.
- Satisfação da equipa: a divisão de funcionalidades em tarefas pequenas e de fácil gestão ajudam a equipa a continuar motivada.

Por outro lado, para trabalhar de forma eficaz neste modelo de desenvolvimento são necessários alguns pré-requisitos que nem todas as organizações estão dispostas ou têm possibilidade de garantir. Nesse sentido, as principais desvantagens são (OutSystems 2021d):

- Escala: este tipo de metodologia está pensada para equipas pequenas e circunscritas. Uma vez que a comunicação é um aspeto fulcral da mesma, quanto mais pessoas envolvidas, mais difícil é a sua aplicação.
- Compromisso: um método iterativo e interativo requer constante atenção do cliente. A necessidade de alocação continuada e reuniões frequentes pode ser um entrave para clientes habituados a uma metodologia tradicional de desenvolvimento.
- Foco na interface: o cliente julga a qualidade da solução com base nos protótipos. A interação do cliente é normalmente com a interface e não com os sistemas de back end, pelo que existe a tendência de dar mais atenção a esta parte do software.

A adoção da metodologia RAD é, portanto, indicada em casos em que as entregas podem ser facilmente divididas, a comunicação com o cliente e o utilizador final é fácil e a equipa de desenvolvimento tem o conhecimento e acesso às ferramentas necessárias para adotar um processo de mudança contínua.

Atualmente, a metodologia formal descrita originalmente por James Martin em 1991 já não é seguida, no entanto, as ideias base foram incorporadas em metodologias ágeis como, por exemplo, o Scrum. O conceito de RAD deixou de estar tão ligado a uma modelo específico, tornando-se mais abrangente, englobando todos os processos, ferramentas e filosofias que aceleram a tradução de requisitos em software funcional (Ferreira 2009).

2.1.2 Scrum

Procurando uma alternativa às metodologias tradicionais de desenvolvimento de software como o modelo em cascata, Ken Schwaber e Jeff Sutherland, em 1995, apresentaram pela primeira vez a framework Scrum. Scrum é uma framework leve e simples de entender para desenvolver e manter produtos complexos e consiste num conjunto de papéis, artefactos e regras que, em conjunto, guiam o desenvolvimento. Assenta no empirismo, uma vez que assume que o conhecimento é derivado da experiência e, nesse sentido, procura melhoramento contínuo não só do produto, mas também do processo. É um processo iterativo e incremental que pretende aumentar a previsibilidade e controlar o risco (Sutherland e Schwaber 2020).

O desenvolvimento é dividido em Sprints - ciclos iterativos de desenvolvimento - que devem ser de duração constante, normalmente entre 2 semanas e 1 mês. Em cada Sprint, os objetivos devem ser claros e divididos em tarefas pequenas, de preferência independentes e testáveis (Sutherland e Schwaber 2020).

Em Scrum existem 3 papéis (Sutherland e Schwaber 2020):

- Product Owner: faz a ponte entre as necessidades do cliente e os requisitos passados à equipa de desenvolvimento. O principal objetivo deste papel é decidir o que vai ser desenvolvido e quando, maximizando o valor entregue pela equipa de desenvolvimento. É da responsabilidade do Product Owner criar e gerir os requisitos de acordo com a sua prioridade e valor para o cliente. Deve trabalhar em grande proximidade com a equipa de desenvolvimento.
- Scrum Master: responsável pelo cumprimento dos princípios Scrum, o Scrum Master é um facilitador na medida em que ajuda a minimizar interferências que possam afetar a produtividade da equipa de desenvolvimento. Funciona como um mediador entre a equipa de desenvolvimento e o exterior.
- Equipa de desenvolvimento: responsável por construir o produto, a equipa de desenvolvimento deve ser multifuncional e capaz de organização própria.

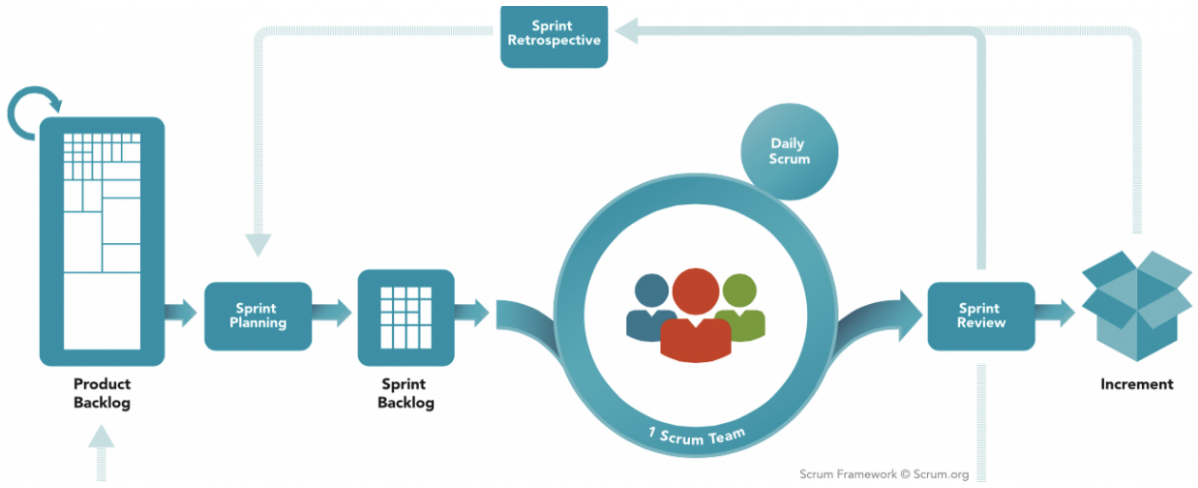

Figura 2.2: Processo Scrum (scrum.org).

Outro aspeto importante do Scrum são os eventos (figura 2.2). Estas reuniões, também chamadas de cerimónias, existem em todas as Sprints e são previamente marcadas, seguindo uma calendarização e duração específicas, de modo a formar uma rotina e evitar tempo perdido. As cerimónias têm como objetivo melhorar a comunicação da equipa, aumentando a visibilidade do trabalho desenvolvido. São elas (Sutherland e Schwaber 2020):

- Sprint: iteração de duração fixa, onde as tarefas são planeadas, implementadas e testadas, focando-se na entrega de valor para o cliente.
- Daily: reunião diária de curta duração (cerca de 15 minutos) onde se acompanha o progresso do trabalho da Sprint. Cada elemento da equipa deve responder a 3 questões principais: o que fiz ontem? O que vou fazer hoje? Vejo algum entrave para atingir o objetivo da Sprint?
- Planning: cerimónia efetuada no início de cada Sprint, onde a equipa discute e se compromete com as tarefas que vai implementar durante a Sprint. Estas tarefas são escolhidas de acordo com a sua prioridade no Backlog.
- Review: acontece no fim de uma Sprint e corresponde à altura em que a equipa de desenvolvimento apresenta as funcionalidades implementadas ao Product Owner. O principal objetivo é a obtenção de feedback.
- Retrospective: cerimónia efetuada no fim de cada Sprint, onde a equipa reflete sobre a execução da Sprint, elencando pontos de ação para melhoria contínua.
- Grooming: reunião em que o Backlog é revisto, tarefas são criadas, refinadas e priorizadas.

Scrum utiliza, também, 3 artefactos que ajudam a gerir o trabalho (Sutherland e Schwaber 2020):

- Product Backlog: responsabilidade do Product Owner, o Product Backlog é uma lista de requisitos para mudanças adicionais ao produto que está em constante mudança.
- Sprint Backlog: lista de tarefas correspondentes ao compromisso da equipa para uma determinada Sprint. Só a equipa de desenvolvimento pode adicionar tarefas ao Sprint Backlog durante a Sprint.
- Potentially Releasable Product Increment: no final de cada Sprint a equipa entrega um incremento ao produto que pode ser lançado.

Com ciclos de desenvolvimento curtos e adaptabilidade do processo, Scrum é capaz de dar resposta às necessidades das equipas de desenvolvimento de hoje, pelo que se tornou a metodologia ágil mais adotada (Waheed et al. 2018).

As principais dificuldades na adoção desta framework surgem nas fases iniciais de adaptação, uma vez que a equipa tem pouca experiência no processo e tem, normalmente, dificuldades em estimar o esforço. Assentando no conhecimento empírico, podem ser necessárias várias iterações até a equipa ajustar o processo às suas necessidades (Despa 2014).

2.2 Low-code e no-code

Cada vez mais as empresas manifestam grandes dificuldades em atender a todas as necessidades tecnológicas que enfrentam usando métodos tradicionais de desenvolvimento, baseados em codificação manual.

Para se manterem competitivas, muitas começaram a procurar outras formas de construir software que deem resposta às principais limitações que métodos tradicionais de desenvolvimento apresentam como, por exemplo, um grande investimento inicial num departamento de informática, a contratação de profissionais especializados e, principalmente, a produtividade das equipas de desenvolvimento e consequentemente o tempo de chegada ao mercado (Caspio 2021). Num inquérito feito a várias organizações com o objetivo de perceber as principais metas da transformação digital foram identificadas quatro prioridades (OutSystems 2019):

- Melhorar a agilidade e acelerar a inovação (22%).
- Reduzir custos e melhorar a eficiência (17%).
- Crescer em novos mercados (15%).
- Ir de encontro às preferências e tendências dos consumidores (14%).

Para dar resposta a estes problemas surge o low-code - termo cunhado em 2014 pela Forrester Research. Low-code é uma nova forma de desenvolvimento que possibilita a entrega mais rápida de software ao oferecer um ambiente de desenvolvimento simplificado, normalmente visual, que requer pouca, ou nenhuma, codificação manual.

Da mesma forma que a evolução das linguagens de programação (figura 2.3) do Assembly para o C, para o C# têm vindo a abstrair a complexidade de codificar software, a programação visual surge como um novo paradigma de alto nível para simplificar ainda mais o processo, de modo a torná-lo mais intuitivo e acessível a todos os perfis, não só programadores (Veiga 2018).

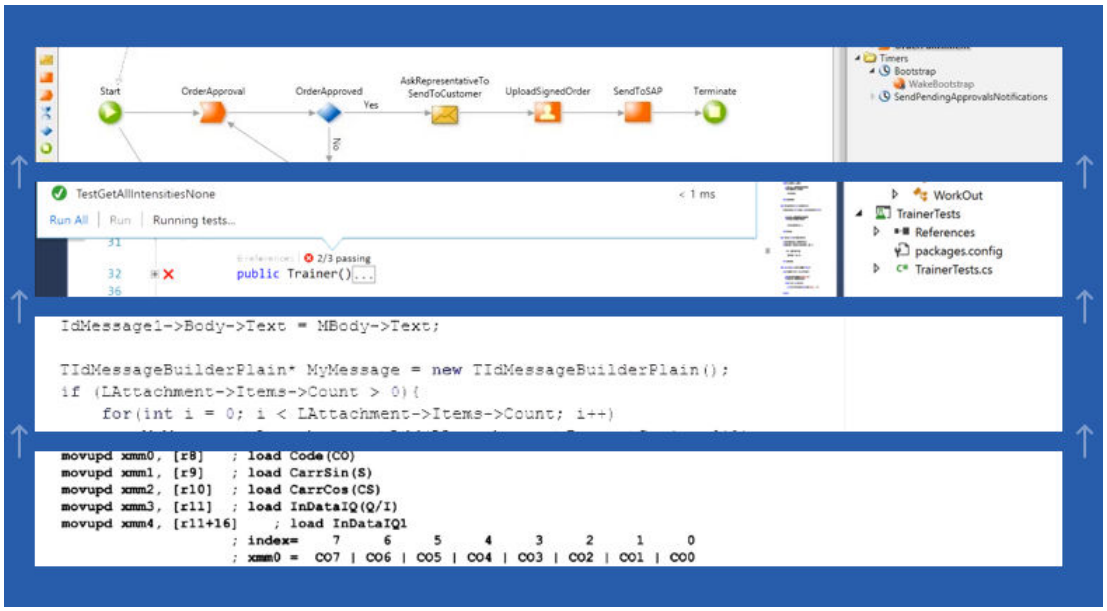

Figura 2.3: Níveis de abstração (Veiga 2018).

As plataformas low-code disponibilizam várias ferramentas integradas que permitem gerir todo o ciclo de vida de uma aplicação desde da sua conceção, até à entrada em produção e fase de manutenção. Normalmente comercializadas num modelo Platform-as-a-Service (PaaS), estas plataformas estão implantadas na nuvem, pelo que, o esforço e custos relacionados com infraestrutura encontram-se já cobertos (Alexander 2021b).

Plataformas no-code pretendem atingir objetivos similares, mas uma vez que não permitem a extensão das funcionalidades a partir de código externo a sua aplicabilidade é muito menor e são, portanto, mais adequadas a necessidades específicas e limitadas como, por exemplo, a criação de protótipos ou a construção de *dashboards* operacionais internas (Jednaszewski 2022). Plataformas low-code, por outro lado, ao permitir essa extensão podem ser usadas para soluções mais complexas como a criação de produtos Business to Business (B2B) e Business to Consumer (B2C) ou a modernização de sistemas *legacy*. Estas ferramentas são normalmente constituídas por (Alexander 2021b):

- Um IDE: um ambiente visual onde se podem definir interfaces, lógica e fluxos de negócios, modelos de dados e código externo.
- Conectores a vários serviços: ligação a bases de dados e serviços internos ou externos à plataforma.
- Gestor do ciclo de vida da aplicação: ferramentas para *debug*, *deployment* e monitorização.

2.3 Aplicabilidade e adoção de plataformas low-code

Segundo projeções da Gartner, mais de 70% das aplicações em 2025 serão desenvolvidas numa plataforma low-code, o que representa um aumento de 45% face a 2020. Espera-se também que a percentagem de aplicações desenvolvidas por *Citizen Developers* - profissionais fora da área da informática - aumente radicalmente (Alexander 2021a). Em 2021, devido ao elevado custo e esforço associado ao desenvolvimento de software, 77% das organizações já adotaram tecnologias low-code e 33% já usaram uma destas plataformas para desenvolver aplicações críticas para o seu negócio (Roe 2021).

Considerando as características destas plataformas, as principais vantagens face ao desenvolvimento tradicional são (Sanchis et al. 2020):

- Velocidade: agilizando o processo de desenvolvimento, teste e implantação das soluções, uma aplicação pode chegar ao mercado cerca de 10x mais rápido.
- Redução de custos: a possibilidade de desenvolver internamente as aplicações de forma mais rápida aumenta a taxa de retorno sobre o investimento.
- Redução de complexidade: a equipa de desenvolvimento pode focar-se nos requisitos de negócio, uma vez que a complexidade técnica está altamente abstraída.
- Envolvimento de vários perfis: devido à facilidade de aprendizagem destas plataformas, a colaboração do negócio com as equipas de desenvolvimento é mais transparente.
- Facilidade de inovação: com o foco afastado das especificidades técnicas, as empresas podem concentrar-se em inovar.

Em linha com as vantagens enumeradas, as principais razões para a adoção destas plataformas segundo um relatório publicado em 2019 (OutSystems 2019) baseado num inquérito

a 3300 profissionais de informática são: a aceleração da transformação digital (66%), a aumento da capacidade de resposta ao negócio (66%), a redução de dependência de profissionais altamente especializados (44%), a redução de dívida técnica (28%), proteção contra tecnologia descontinuada (22%) e envolvimento de perfis não técnicos no desenvolvimento (20%), como ilustrado na figura 2.4.

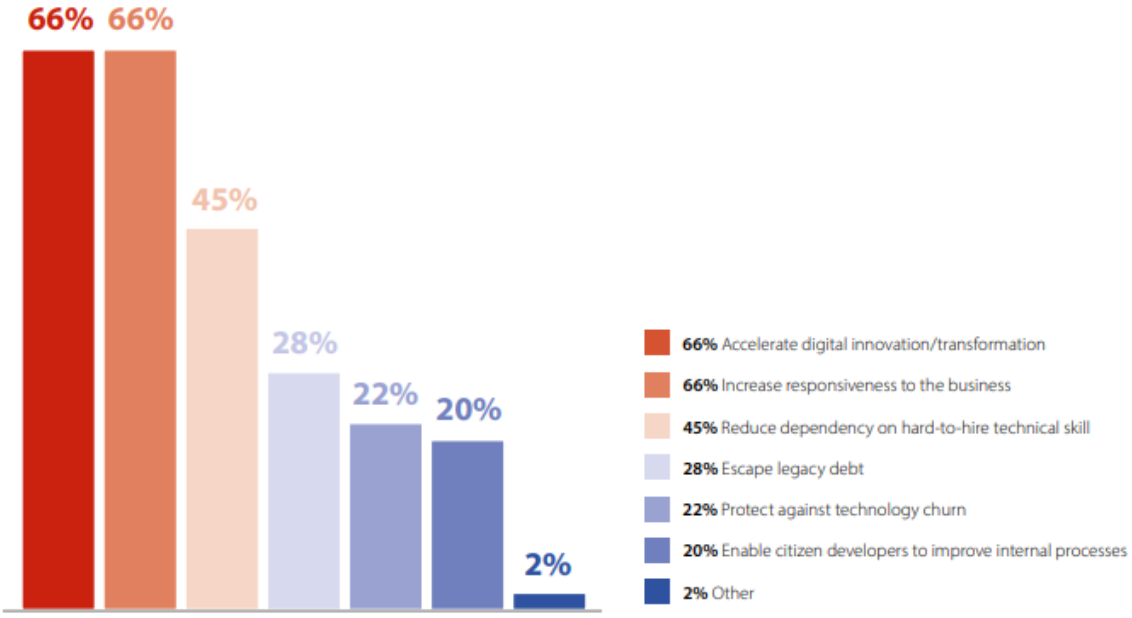

Figura 2.4: Razões para considerar low-code (OutSystems 2019).

Não obstante, existem também fatores negativos e desafios a considerar antes de adotar uma plataforma low-code. São eles (Setrag Khoshafian 2021):

- Dependência do fornecedor: o fornecedor mantém controlo total sobre a solução desenvolvida. Atualizações tecnológicas são normalmente mais lentas uma vez que dependem completamente da velocidade e interesses do fornecedor. O código visual desenvolvido é específico a cada plataforma e não portável. Ainda que seja possível obter o código fonte gerado, este é normalmente ininteligível e demasiado trabalhoso de alterar se não suportado pela plataforma.
- Preço: o custo das subscrições a estes serviços é difícil de prever. Especialmente em contratos de curto e médio prazo, existe o risco das subscrições rapidamente não se alinharem com as expectativas do comprador.
- Recursos e comunidade: muitas destas plataformas são ainda pouco maduras no mercado e, portanto, a comunidade que as suporta é pequena pelo que os recursos disponíveis são escassos.
- Caixa negra: não existe visibilidade sobre o código gerado o que dificulta processos de *debugging* e otimização. Normalmente, este nível de abstração causa, também, impactos negativos no desempenho das aplicações.

Quando questionadas, as organizações identificaram como fatores inibidores da adoção de plataforma low-code (figura 2.5) os seguintes: falta de conhecimento acerca das plataformas low-code (47%), dependência do fornecedor (37%), aplicabilidade (32%), preocupações de escalabilidade (28%), preocupações de segurança (25%) (OutSystems 2019).

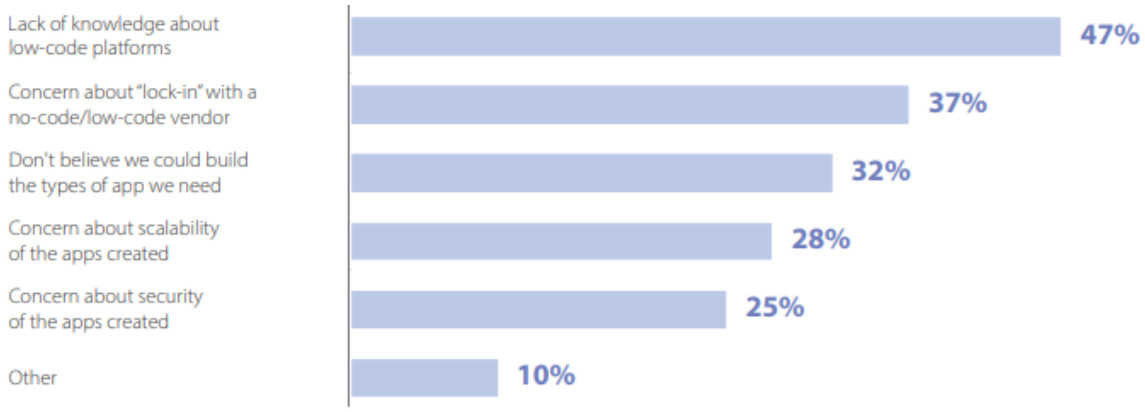

Figura 2.5: Razões para não escolher low-code (OutSystems 2019).

Considerando as vantagens e desvantagens destas plataformas, os principais casos de uso a que atualmente dão resposta são (Alexander 2021b):

- Entrega de aplicações móveis e Web.
- Desenvolvimento de aplicações de Business Process Management (BPM).
- Desenvolvimento de aplicações nos modelos Software as a Service (SaaS) e Independent Software Vendor (ISV).
- Suporte de estratégias de *Citizen Development*.
- Juntar equipas com vários perfis.

Pode-se concluir, portanto, que atualmente este tipo de abordagem ao desenvolvimento é procurada quando o foco está em entregas rápidas e na construção de produtos padrão com poucos recursos humanos. Por outro lado, estas plataformas ainda não dão resposta a requisitos muito específicos, que requerem muito controlo sobre o código produzido ou uma grande otimização (Hosking 2021).

2.4 Principais plataformas low-code

A Gartner e a Forrester Research são empresas que realizam pesquisas especializadas de mercado, gerando métricas e análises a respeito de tecnologias de forma a que os seus clientes tomem decisões estratégicas. Nesse sentido, estas duas organizações de renome publicam relatórios de análise de mercado das plataformas low-code em que apresentam os seus pontos positivos e negativos e as classificam como líderes, visionárias, desafiadoras e de nicho.

Na impossibilidade de estudar todas as plataformas mencionadas nos relatórios da Gartner (figura 2.6) e Forrester (figura 2.7), optou-se por analisar as plataformas que surgem como líderes em ambos os relatórios: OutSystems, Mendix e PowerApps (Microsoft).

Figura 2.6: Quadrante mágico low-code (Gartner 2021)

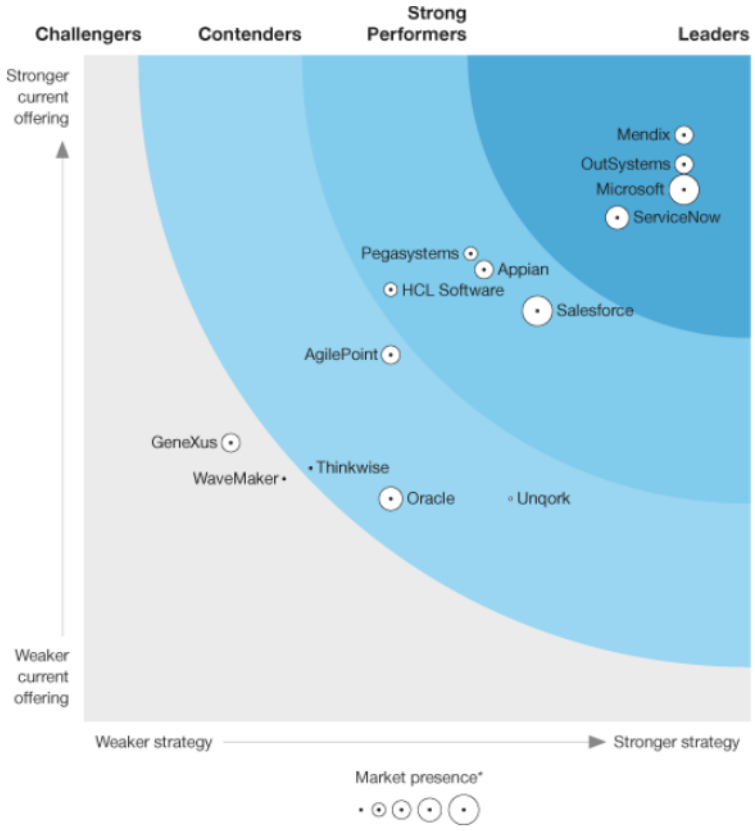

Figura 2.7: Forrester Wave low-code (Forrester 2021)

2.4.1 OutSystems

OutSystems é uma plataforma low-code que combina aspetos de programação visual e orientada a modelos com inteligência artificial e conceitos DevOps, numa infraestrutura nativa à nuvem. Direcionada principalmente a grandes empresas, OutSystems permite gerar aplicações Web (ReactJS) e aplicações móveis (Cordova + ReactJS), assim como código de servidor (C#) (OutSystems 2021a).

Na versão atual, OutSystems 11, a plataforma disponibiliza várias ferramentas de desenvolvimento e gestão que, de forma integrada, permitem o controlo de todo o ciclo de vida de uma aplicação. Quanto às ferramentas de desenvolvimento, OutSystems oferece 3 produtos principais e 2 Builders (aplicações no-code para casos de uso específicos que se integram com as ferramentas principais) (OutSystems 2021b).

- Service Studio: ambiente visual What You See Is What You Get (WYSIWYG) *drag and drop* de desenvolvimento (figura 2.8) usado para criar todas as partes da aplicação: modelo de dados, lógica de negócio, interface gráfica, processos e integrações. Esta ferramenta usa inteligência artificial para sugerir próximos passos, garantir integridade dos módulos e gerir dependências.
- Integration Studio: ambiente que permite a criação de componentes em C# que estendem a funcionalidade da plataforma.
- Forge: repositório de componentes, bibliotecas e conectores disponibilizados pela comunidade que encapsulam funcionalidades comuns.
- Workflow Builder: ferramenta no-code focada na construção de processos de negócio. Gera aplicações OutSystems que podem ser estendidas e alteradas no Service Studio.
- Experience Builder: aplicação no-code que permite rápida prototipagem de fluxos aplicacionais comuns como, por exemplo, registo de um utilizador. Gera aplicações OutSystems que podem ser estendidas e alteradas no Service Studio.

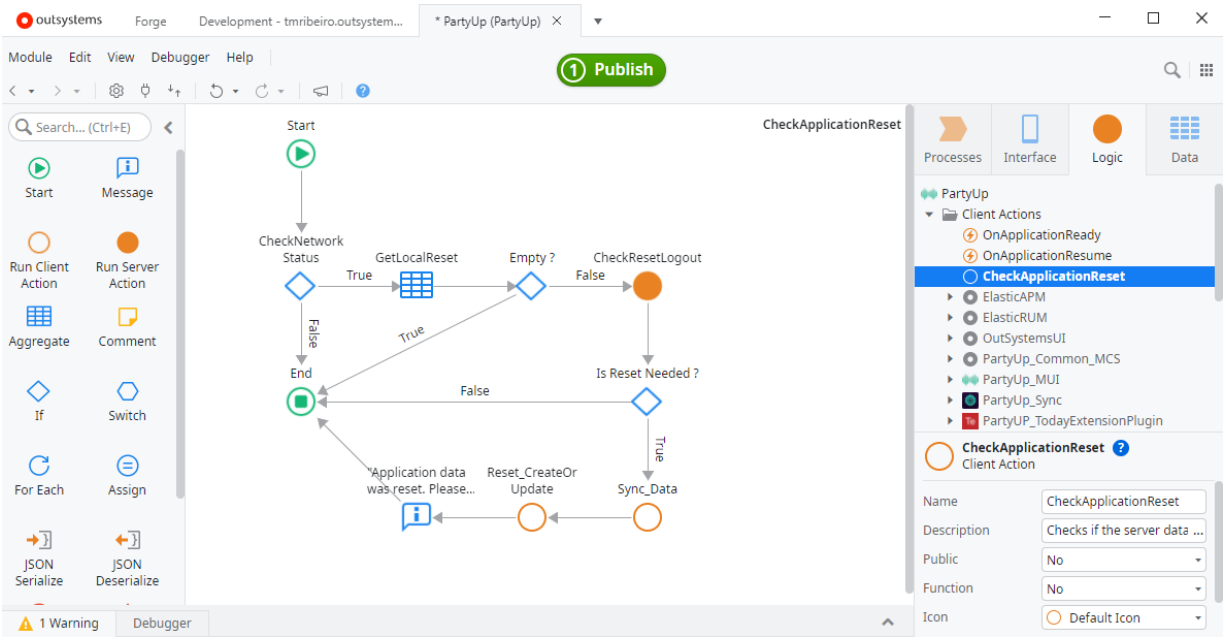

Figura 2.8: Service Studio IDE (OutSystems 2021)

Quanto às ferramentas de gestão, OutSystems dispõe de 2 ferramentas principais - o Service Center e o Lifetime - e uma ferramenta anexa - o Architecture Dashboard (OutSystems 2021b).

- Service Center: aplicação Web que gere os aspetos operacionais de um determinado ambiente. Permite efetuar configurações sobre aplicações em específico, publicar soluções, monitorizar *logs*, gerir regras de segurança e aceder a relatórios de desempenho das aplicações.
- Lifetime: aplicação Web que centraliza a gestão de todos os ambientes (desenvolvimento, teste, etc.). Simplifica o processo de *deployment* entre ambientes, seguindo práticas de DevOps. Permite a gestão de utilizadores de IT.
- Architecture Dashboard: ferramenta que, com recurso a inteligência artificial, analisa padrões de código de forma a identificar dívida técnica. Sugere alterações e melhorias de desempenho, manutenção, arquitetura e segurança.

As várias ferramentas interagem entre si para formar a plataforma OutSystems, como exemplificado na figura 2.9.

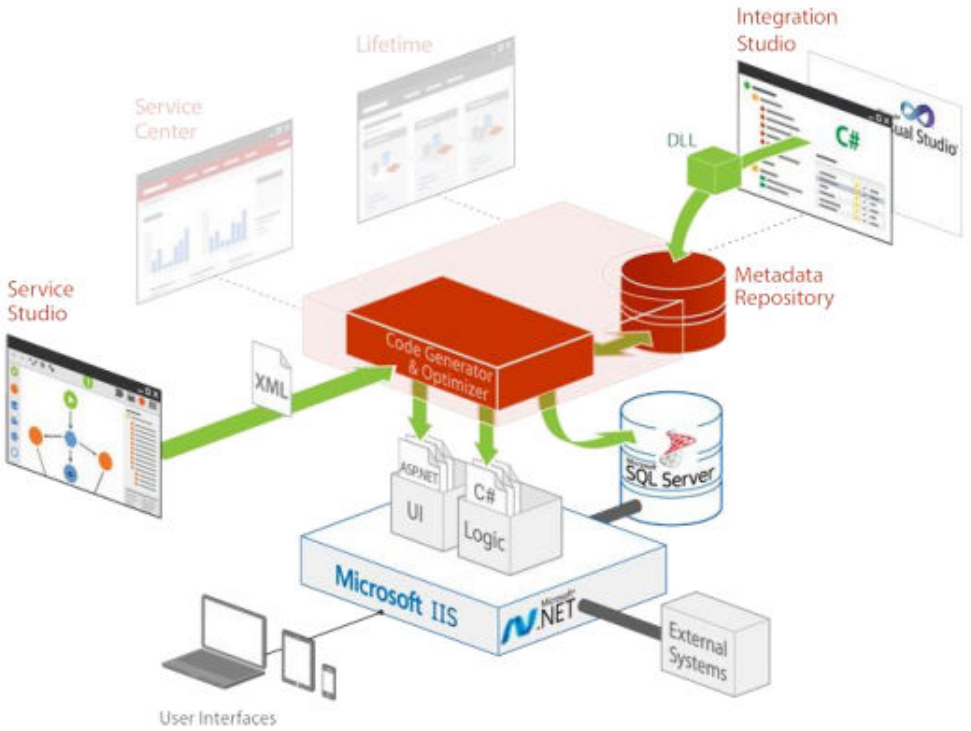

Figura 2.9: Arquitetura OutSystems (OutSystems 2021)

Segundo Gartner, as principais forças desta plataforma prendem-se com a qualidade e abrangência das funcionalidades disponibilizadas que permitem a inovação. O OutSystems UI - framework de design - oferece padrões que cobrem grande parte dos casos de uso. Quanto às fraquezas, é apontada a falta de soluções específicas para cada indústria e o modelo de subscrição que é complexo, pouco transparente e inflexível (Wong 2021).

2.4.2 Mendix

Mendix é uma plataforma de low e no-code de alta produtividade - High-Productivity application Platform as a Service (hpaPaaS) - para desenvolvimento de aplicações. Oferece apoio em todas as etapas do desenvolvimento, desde a criação dos requisitos, até ao desenvolvimento e lançamento das aplicações (Mendix 2022c).

É baseada numa arquitetura nativa à cloud e, com ajuda de inteligência artificial, permite a construção de soluções Web, gerando uma SPA baseada em tecnologias padrão da indústria como HTML, CSS e Javascript (Mendix 2021d) e soluções móveis nativas com recurso à tecnologia React Native (Mendix 2021b). A possibilidade de criação de aplicações móveis híbridas baseadas em Cordova é ainda suportada na versão atual - Mendix 9 - mas será removida no futuro (Mendix 2021c).

A plataforma oferece 3 ferramentas principais e 1 ferramenta anexa: um IDE no-code, um IDE low-code, o portal do programador e um agregador de dados (Mendix 2021e).

- Mendix Studio: editor colaborativo visual no-code (figura 2.10) que é indicado para o desenvolvimento por parte de *Citizen Developers*. Possibilita a construção de interfaces com *widgets* predefinidos e customizados, criação de modelos de dados e de fluxos de negócio.

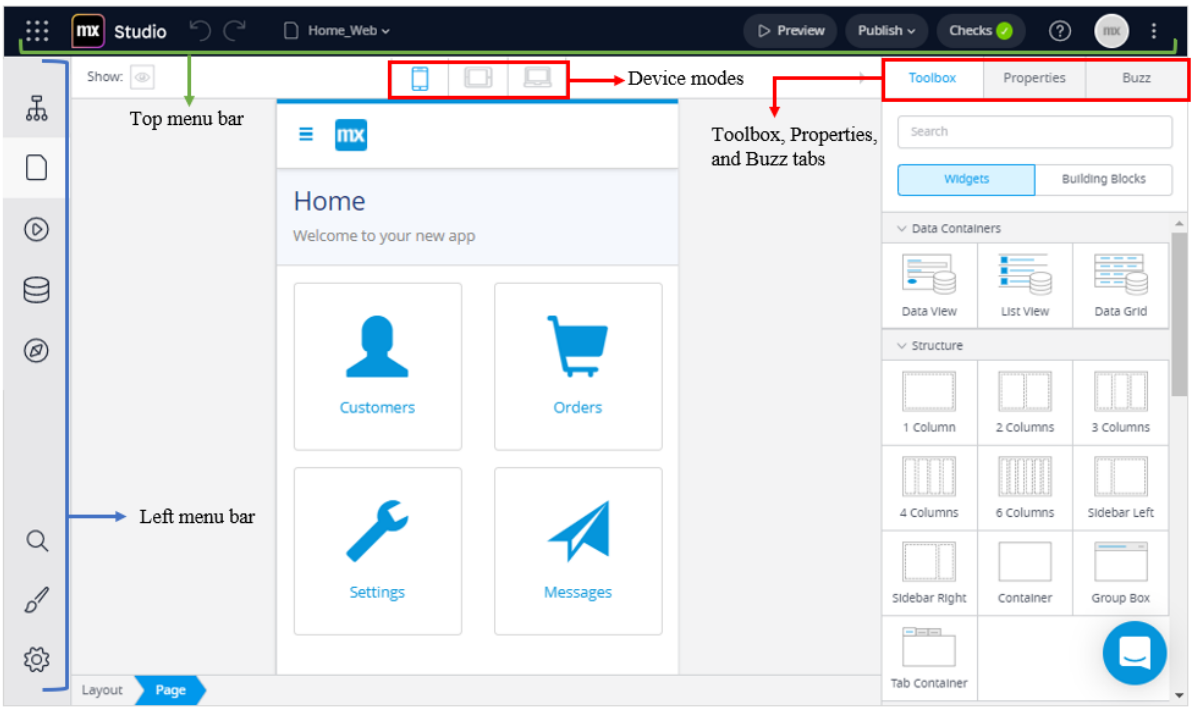

Figura 2.10: Mendix Studio IDE (Mendix 2021)

- Mendix Studio Pro: editor colaborativo low-code que permite a profissionais de IT estender as funcionalidades da plataforma com código customizado.
- Developer Portal: ferramenta Web que integra vários componentes relacionados com o desenvolvimento de aplicações. Contém módulos de gestão de projeto e *feedback*, gestão e monitorização de aplicações, controlo de acessos e permissões e um mercado de componentes disponibilizados pela comunidade.

- Data Hub: produto licenciado separadamente que permite a centralização de dados de uma organização, de modo a facilitar a sua integração com aplicações desenvolvidas na plataforma.

Segundo a Gartner, as principais forças da Mendix prendem-se com a extensão das funcionalidades oferecidas e a inovação que permitem, ao suportar casos de uso de IOT, Data Hub, entre outros. A visibilidade é outro ponto forte, uma vez que tem crescido muito, especialmente depois da sua aquisição por parte Siemens. As fraquezas apontadas estão principalmente ligadas à estratégia da empresa, uma vez que apresenta pouca presença em alguns mercados e está intimamente ligada aos interesses da empresa mãe (Wong 2021).

2.4.3 PowerApps

A PowerApps, plataforma comercializada pela Microsoft, é uma ferramenta low-code com uma visão holística sobre o desenvolvimento. Diferencia-se pelo facto de oferecer todo o poder da Power Platform que inclui Power Automate, Dataverse, Power BI, Power Virtual Agents, Microsoft Office 365, Dynamics 365 e serviços na nuvem Azure (Vivek 2021).

Permite a construção de aplicações Web e móveis, fluxos de negócio customizados e aplicações de suporte a linhas de negócios. Existem 3 ferramentas principais que permitem desenvolver 3 tipos de aplicações diferentes.

- Power Apps Studio: editor drag and drop com uma interface semelhante ao Microsoft Power Point (figura 2.11), que permite a criação de aplicações de tela, facilitando a construção de interfaces customizadas às necessidades (Microsoft 2021b).

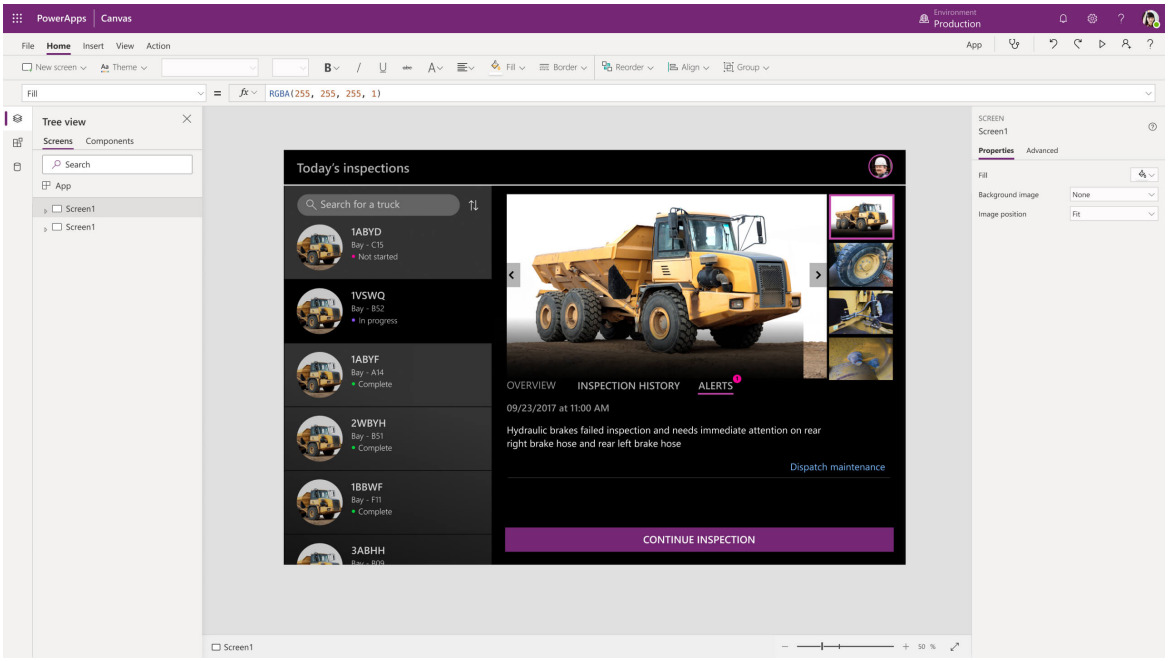

Figura 2.11: Power Apps Studio IDE (Microsoft 2021)

- App Designer: editor visual que permite a criação de aplicações orientadas a modelos. Ao contrário de aplicações de tela, em que o utilizador tem controlo total sobre a interface, nas aplicações orientadas a modelos o foco está nos dados e nas relações entre eles. É recomendado para casos de uso relacionados com processos de negócio como, por exemplo, a integração de novos colaboradores (Microsoft 2021a).

- Power Apps Portals Studio: editor WYSIWYG direcionado à criação de aplicações Web para utilizadores externos à organização (Microsoft 2021c).

É possível, também, estender as funcionalidades da plataforma com código customizado para interagir com dados e metadados, criar lógica de negócio e integrar com conectores externos (Vivek 2021).

Segundo a Gartner, as principais forças da Power Platform, prendem-se com a viabilidade do modelo de negócio, uma vez que os serviços da Microsoft já são usados por grande parte das empresas, e a capacidade de inovação apoiada pelo modelo de inteligência artificial GPT-3. Como fraquezas, são apontadas as dificuldades em responder a casos de uso mais complexos e a confusão causada pela constante mudança de modelos de subscrição e renomeação de serviços (Wong 2021).

2.4.4 Comparação das plataformas low-code

A seguinte tabela (2.1) compara as funcionalidades das plataformas low-code estudadas. As categorias são baseadas na taxonomia de avaliação apresentada por Sahay et al. 2020 e no guia de avaliação de plataformas low-code apresentado por coMakeIt 2020.

Funcionalidade	OutSystems	Mendix	PowerApps
Interface gráfica			
Editor drag and drop	✓	✓	✓
Editor WYSIWYG	✓	✓	✓
Componentes predefinidos	✓	✓	✓
Temas base	✓	✓	✓
Templates base	✓	✓	✓
Workflows predefinidos	✓	✓	
Personalização de workflows	✓	✓	
Rastreador de progresso	✓	✓	✓
Estatísticas de uso	✓		
Interoperabilidade			
Conectores a serviços externos	✓	✓	✓
Conectores a bases de dados externas	✓	✓	✓
Geração automática de conectores REST	✓	✓	✓
Segurança			
Segurança aplicacional	✓	✓	✓
Segurança da plataforma	✓	✓	✓
Colaboração			
Colaboração offline	✓	✓	✓
Colaboração online	✓	✓	
Escalabilidade			
Escalabilidade do servidor	✓	✓	✓
Escalabilidade da base de dados	✓	✓	✓
Escalabilidade vertical	✓	✓	✓
Escalabilidade horizontal	✓	✓	✓
Escalabilidade autónoma		✓	
Escalabilidade automática			

Tabela 2.1: Comparação de plataformas low-code

Funcionalidade	OutSystems	Mendix	PowerApps
Lógica de negócio			
Editor de workflows gráfico	✓	✓	
Editor assistido por inteligência artificial	✓	✓	
Editor visual do modelo de dados	✓	✓	✓
Mecanismos de construção de aplicações			
Geração de código	✓		
Interpretação de modelos		✓	✓
Estratégias de implantação			
Implantação local	✓	✓	
Implantação na nuvem	✓	✓	✓
Tipos de aplicações suportadas			
Aplicações Web	✓	✓	✓
Aplicações Progressive Web App (PWA)	✓	✓	✓
Aplicações móveis	✓	✓	
Automação de processos	✓		✓
BPM	✓	✓	✓
Sistemas embebidos			

Tabela 2.2: Comparação de plataformas low-code (continuação)

Pode-se concluir que as 3 plataformas estão bastante próximas em relação às funcionalidades que disponibilizam. No entanto, a oferta da Microsoft, a PowerApps, destaca-se pela negativa nos aspetos relativos à personalização e funcionalidades mais avançadas como, por exemplo, o editor assistido por inteligência artificial.

Capítulo 3

Engenharia de requisitos

O presente capítulo apresenta os requisitos funcionais e não funcionais ligados aos casos de uso escolhidos para a prova de conceito a desenvolver.

3.1 Engenharia de requisitos

A engenharia de requisitos é uma disciplina que engloba todas as tarefas e técnicas que resultam no entendimento dos requisitos de um sistema. Identifica os objetivos do sistema a desenvolver e a operacionalização de tais objetivos em serviços e restrições (Lamsweerde 2000).

É o primeiro processo no desenvolvimento de software e é de extrema importância para perceber o que o cliente pretende, analisando as suas necessidades de modo a averiguar a viabilidade, negociar uma solução, especifica-la e, por fim, valida-la. É composto por 7 fases distintas: arranque, elicitação, elaboração, negociação, especificação, validação e gestão (Lamsweerde 2000).

Nesta secção são descritos os requisitos funcionais e não funcionais relativos aos casos de uso a desenvolver. Estes requisitos foram definidos para a versão atual do Portal Cartão Continente pelo Product Owner do projeto.

3.1.1 Requisitos funcionais

Requisitos funcionais capturam o funcionamento esperado do sistema. Descrevem serviços, tarefas ou funções que o sistema deve executar (Malan, Bredemeyer et al. 2001).

Uma forma de capturar requisitos funcionais são os casos de uso. Cada caso de uso define uma interação entre um ator externo (um utilizador, um conjunto de utilizadores, um sistema, etc.) e o sistema de modo a chegar a um determinado objetivo. Um caso de uso deve usar linguagem do domínio não ambígua e descrever todo o fluxo de interação entre o ator e o sistema, incluindo cenários alternativos e de erro (Malan, Bredemeyer et al. 2001).

Na figura 3.1 é apresentado o diagrama de casos de uso para utilizadores autenticados e anónimos do Portal assim como o utilizador de *backoffice*.

Um utilizador anónimo do Portal corresponde a qualquer utilizador não autenticado, que apenas tem acesso às áreas publicas. Um utilizador autenticado é um cliente Cartão Continente que pode aceder às áreas públicas e áreas relativas à sua conta. Por outro lado, um utilizador de *backoffice* é um operador interno que gere conteúdo que aparece no Portal.

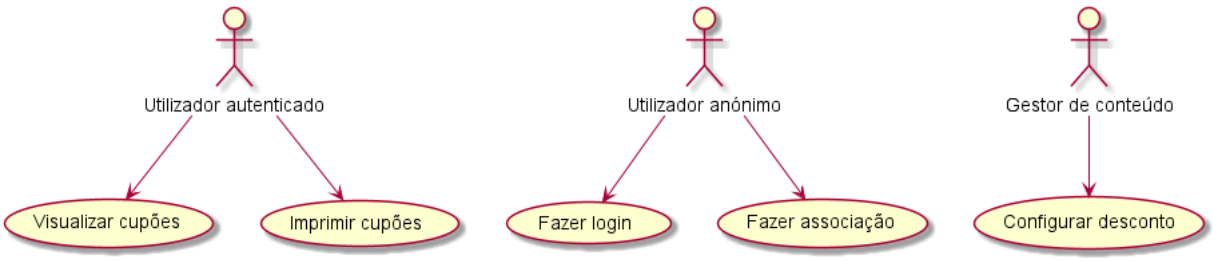

Figura 3.1: Diagrama de casos de uso.

UC01 Visualizar cupões

Um utilizador autenticado ao aceder à área de cupões deve visualizar numa grelha os cupões associados à sua conta segundo o desenho da figura 3.2. Estes devem aparecer em 3 colunas para *desktop* e 2 colunas para dispositivos móveis. Devem ser ordenados pela data de início. Os cupões cuja data de início seja maior do que a data atual devem ter opacidade para dar a entender que ainda não estão disponíveis.

Os cupões são obtidos a partir de um serviço. A resposta do serviço deve ser filtrada para retornar apenas os cupões que não sejam exclusivos para a aplicação móvel.

Deve existir um carrossel de insígnias que permita filtrar os cupões por marca. A ordem das marcas deve estar de acordo com a ordem dos cupões obtidos do serviço. A primeira insígnia deve estar selecionada por defeito.

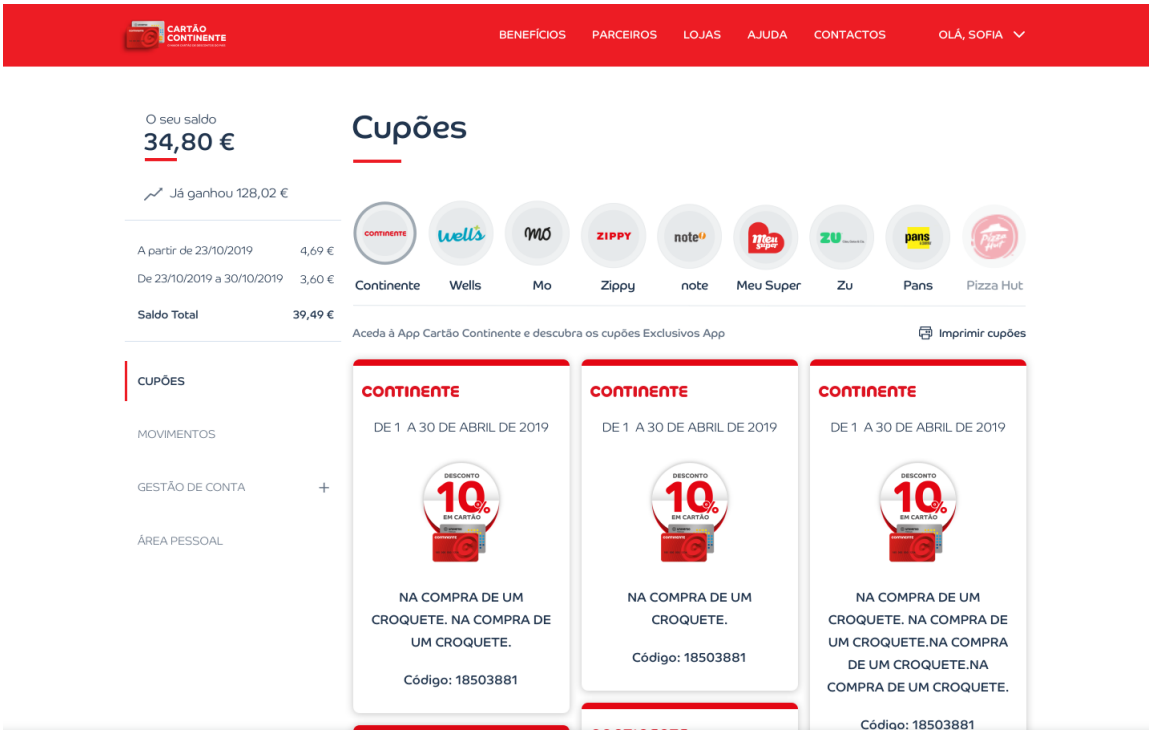

Figura 3.2: Mock da página de cupões.

UC02 Imprimir cupões

Na área de cupões deve ser mostrado um botão para efetuar a impressão de cupões como apresentado na figura 3.2. Ao clicar neste botão deve ser gerado um documento PDF com os cupões atualmente disponíveis para o utilizador. Os cupões devem ter todos a mesma altura e apresentar um código de barras que contenha o código do cupão. A primeira página do PDF gerado deve conter um cabeçalho alusivo ao cartão continente (figura 3.3) e a última página deve sempre conter os termos e condições do uso de cupões.

Esta funcionalidade só deve estar disponível para utilizadores *desktop*, não sendo possível aceder em dispositivos móveis.

Figura 3.3: Cabeçalho da impressão de cupões

UC03 Fazer login

Um utilizador não autenticado pode fazer o *login* de forma a ter acesso às páginas relativas à sua conta. Para isso, deve introduzir o número de telemóvel associado à sua conta. Deve ser considerado um input para o número de telemóvel e uma listagem de indicativos telefónicos. Ao avançar, deve ser enviado um *token* de acesso para o número inserido. Esse *token* deve ser introduzido pelo utilizador e validado pelo servidor. O reenvio do *token* deve ser limitado a 5 tentativas por utilizador com um tempo de espera crescente entre pedidos.

Se o número de telemóvel inserido corresponder a uma conta no sistema deve ser pedido e validado o código de acesso à conta. Se, por outro lado, o número de telemóvel não estiver ligado a nenhuma conta o utilizador deve ser reencaminhado para o fluxo de associação - UC04 (3.1.1).

UC04 Fazer associação

Um utilizador não autenticado pode realizar uma associação a um cartão continente já existente (mas não pode criar um novo pelo Portal). Para isso, deve, em primeiro lugar, introduzir o seu número de telemóvel. Deve ser considerado um input para o número de telemóvel e uma listagem de indicativos telefónicos. Ao avançar, deve ser enviado um *token* de acesso para o número inserido. Esse *token* deve ser introduzido pelo utilizador e validado pelo servidor. O reenvio do *token* deve ser limitado a 5 tentativas por utilizador com um tempo de espera crescente entre pedidos.

O utilizador sem conta deve ser reencaminhado para um ecrã onde insere o cartão continente da conta a que se quer associar. Se o número de telemóvel introduzido for o do titular da conta, o sistema deve fazer o *login*. Se o número de telemóvel não corresponder ao número de telemóvel do titular, a aplicação deve entrar num fluxo de validação de titularidade onde será enviado um *token* para o email e telemóvel do titular. O utilizador que se quer associar deve inserir este *token* e, depois de validado pelo servidor, a aplicação redireciona para a área autenticada.

UC05 Configurar descontos

Um utilizador com o perfil de Gestor de conteúdo pode aceder ao Backoffice (aplicação Web de configuração) e configurar descontos. Cada cupão está associado a um determinado desconto via insígnia e valor do desconto.

Deve ser apresentado um formulário que permita ao utilizador criar ou editar um desconto. Nesse formulário, deve existir uma *dropdown* com as insígnias disponíveis, um campo decimal para inserir o valor do desconto, e um campo para especificar o tipo de desconto (em euros ou percentagem). Todos os campos são obrigatórios e devem ser validados antes de submetidos. As informações gravadas devem ficar persistidas numa base de dados.

3.1.2 Outros requisitos funcionais e de qualidade

Requisitos não funcionais definem propriedades e restrições do sistema. Para a especificação destes requisitos foi usado o modelo FURPS+.

Funcionalidade

- As páginas da área pessoal só podem ser acedidas por um utilizador previamente autenticado. Numa tentativa de acesso por um utilizador anónimo deve ser lançada e registada uma exceção, reencaminhando o utilizador para uma página de erro.
- Um utilizador autenticado só pode aceder a informação relativa à sua própria conta.
- O *Website* deve estar disponível em português e inglês.
- As validações devem ser tanto do lado do cliente como do lado do servidor.
- Deve ser considerado um limite de tentativas por ação (como, por exemplo, na inserção do código ou no pedido de *tokens*).
- As ações dos utilizadores devem ficar registadas de forma assíncrona em *logs* de navegação.

Usabilidade

- Interface gráfica simples e intuitiva de forma a facilitar a navegação.
- Interface responsiva que se adapte aos vários dispositivos dos utilizadores.
- As mensagens de erro devem ser contextualizadas e claras.

Confiabilidade

- As falhas devem ser circunscritas à funcionalidade específica, de forma a não afetar outras partes da aplicação.
- O sistema deve conseguir recuperar de uma falha com o mínimo de intervenção humana.
- A aplicação deve apresentar alta disponibilidade (mais do que 98% do tempo).

Desempenho

- Os ecrãs devem ser apresentados em menos de 1 segundo.
- Qualquer operação não deve demorar mais do que 3 segundos a ser concluída.
- Devem ser apresentados indicadores de carregamento enquanto um pedido está a ser processado.
- O sistema deve ter a possibilidade de escalar verticalmente e horizontalmente.

Suportabilidade

- Os vários componentes devem poder ser testados de forma isolada.
- Deve ser possível realizar correções num curto espaço de tempo.
- Boas práticas de desenho e implementação devem ser adotadas para evitar dívida técnica.
- Deve ser possível aceder ao *Website* através dos principais *browsers* - Google Chrome, Microsoft Edge, Firefox e Opera.
- O sistema deve ser desenvolvido de forma iterativa, guardando as várias versões num sistema de controlo de versões.
- Deve ser possível voltar para uma versão anterior do sistema.

Restrições

- A prova de conceito final para apresentar ao negócio deve ser desenvolvida na plataforma OutSystems.
- A aplicação em OutSystems deve ser desenvolvida na infraestrutura da Sonae.

Capítulo 4

Desenho da solução

O presente capítulo detalha o processo de desenho da solução. É apresentada a arquitetura proposta para a solução usando boas práticas de engenharia de software. Será usado o modelo C4 para representar o sistema em diferentes níveis de granularidade. O desenho será independente da tecnologia, não tendo em conta características específicas das plataformas que vão ser usadas para o seu desenvolvimento.

4.1 Arquitetura

Para definir a arquitetura é preciso começar por perceber as várias partes que compõem a solução. Para esse efeito, foi criado um diagrama de nível 1 do modelo C4 (figura 4.1) que apresenta o contexto do sistema. Aqui são apresentados os três componentes principais: o sistema do Portal, o serviço externo de gestão de cupões e o serviço externo que envia SMS.

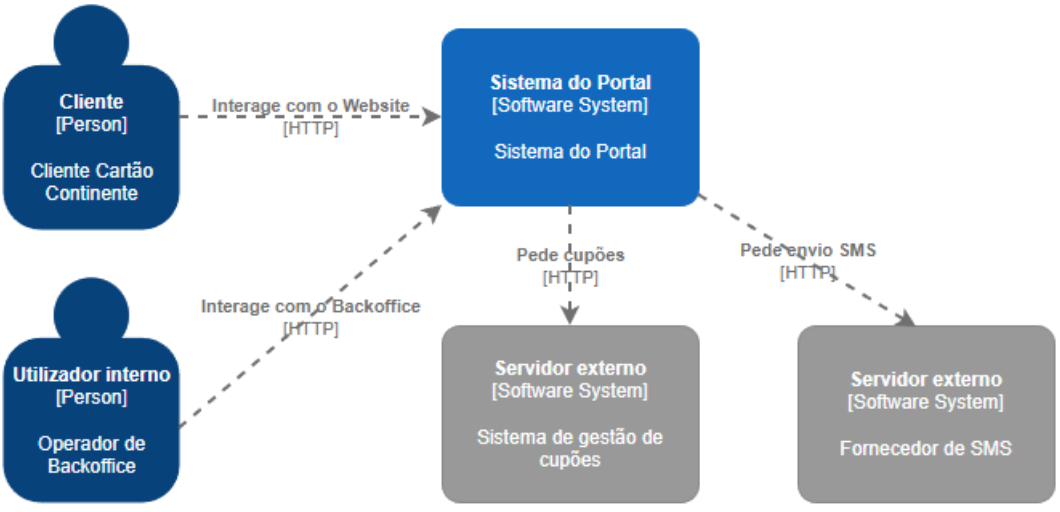

Figura 4.1: Vista nível 1 - contexto do sistema

Existem dois atores principais: o cliente cartão continente e o utilizador interno (operador de backoffice). Uma vez que estes atores têm casos de uso completamente díspares que devem estar segregadas por razões de complexidade, segurança e implantação, foi decidido a criação de uma aplicação de frontend para cliente final e outra para utilizador interno.

De forma a perceber os detalhes do sistema, foram analisados os casos de uso expostos em 3 e levantados os conceitos necessários para dar resposta às funcionalidade pedidas. Nesse sentido, foi criado o modelo de domínio apresentado na figura 4.2.

Um utilizador de Backoffice (gestor de conteúdo) cria, edita ou apaga descontos e insígnias. Um desconto pertence a uma insígnia e uma insígnia pode ter vários descontos. Uma insígnia tem vários cupões que estão associados a um desconto. Um desconto, por sua vez, pode tomar várias formas como desconto em euros ou desconto em percentagem do valor de compra.

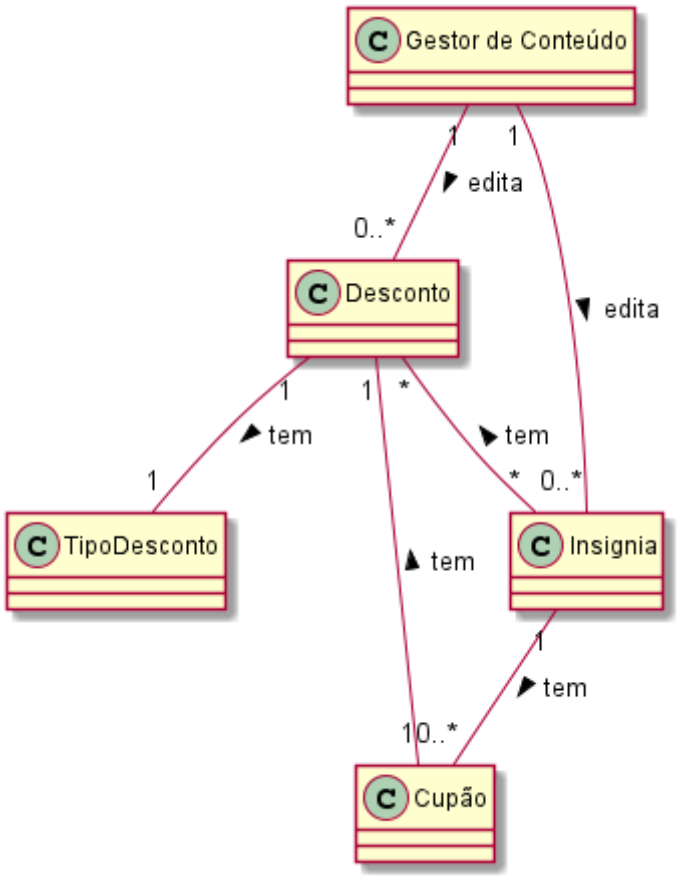

Figura 4.2: Modelo de domínio

Quanto à arquitetura macro do sistema, foi escolhida a arquitetura cliente servidor - arquitetura padrão na construção de websites. Um utilizador interage com uma aplicação que disponibiliza uma interface gráfica (cliente) e comunica com um servidor através da rede.

No que toca à arquitetura do servidor, existem duas alternativas principais: a arquitetura monolítica e a orientada a microsserviços. Muito em voga atualmente, a arquitetura orientada a microsserviços é tipicamente composta por várias aplicações com uma única responsabilidade que comunicam entre si para formar um sistema resiliente, flexível e escalável. Outras vantagens das arquiteturas orientadas a microsserviços são a facilidade de escalar equipas (cada uma a trabalhar num microsserviço) e a liberdade para adotar diferentes tecnologias. Por outro lado, as aplicações monolíticas concentram toda a funcionalidade numa única aplicação, pelo que pode ser mais difícil de escalar tanto a nível de desempenho como a nível de equipas. No entanto, esta abordagem é ainda muito popular atualmente, especialmente

pela sua relativa simplicidade em comparação com microsserviços, uma vez que a sua implementação e implantação é mais fácil, e continua a permitir uma arquitetura modular, em camadas, cada uma com uma responsabilidade específica.

Analisando os requisitos para o Portal Cartão Continente, foi considerado que as principais vantagens dos microsserviços não se aplicam, uma vez que o projeto é de baixa complexidade (poucos conceitos), a equipa é só uma e não existe nenhuma parte do sistema que precise de escalar independentemente das outras. Por estas razões, foi estabelecido que a complexidade acrescida de uma abordagem orientada a microsserviços não se justifica e, portanto, optou-se por uma arquitetura monolítica.

A figura 4.3 apresenta o sistema no nível 2 de granularidade do modelo C4 - diagrama de *containers*. Nesta granularidade é possível decompor o sistema do Portal nos componentes que o constituem.

O cliente final interage apenas com a aplicação do Portal. Esta está alojada numa aplicação *Web* que envia todos os recursos necessários para o *browser* do cliente. O operador de Backoffice interage apenas com a aplicação de Backoffice para efetuar configurações. O servidor é comum às duas aplicações de *frontend*, expondo uma interface REST para todas as operações necessárias. Cada *endpoint* só deve ser acedido por um utilizador com o perfil para tal. O servidor, por sua vez, interage com a base de dados e os sistemas externos.

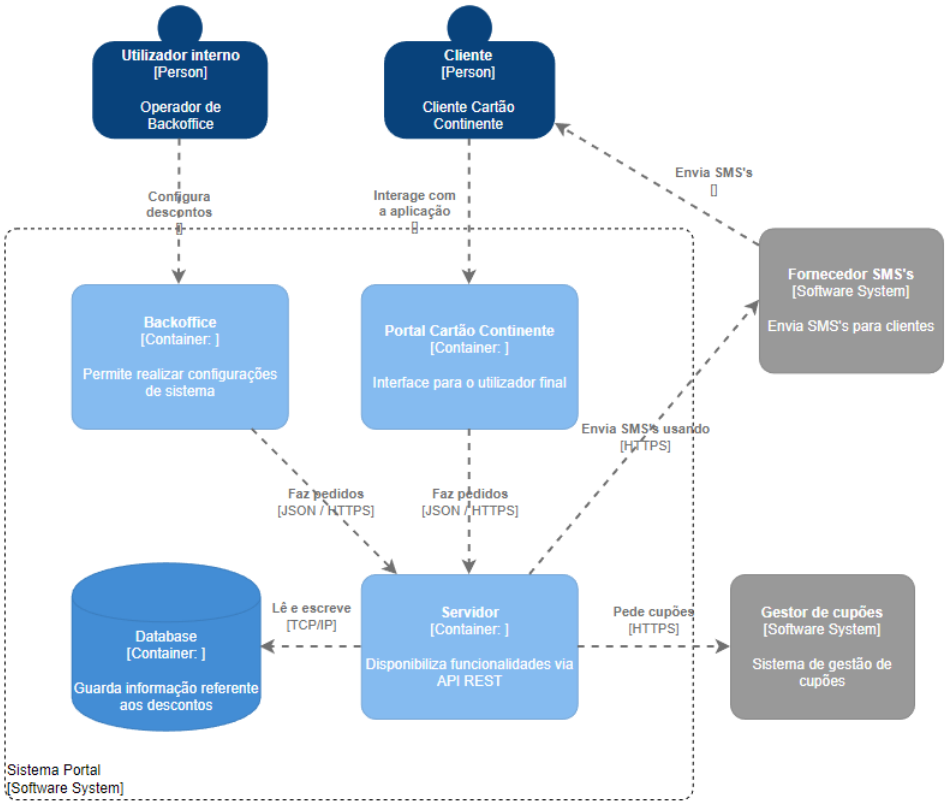

Figura 4.3: Vista nível 2 - diagrama de *containers*

4.1.1 Casos de uso

Para perceber melhor a interação entre as várias partes do sistema, foram criadas vistas de processos - em forma de diagramas de sequência - para cada um dos casos de uso.

UC01 Visualizar cupões (3.1.1)

A figura 4.4 ilustra a sequência de passos relativos ao processo de visualização de cupões. O processo inicia-se quando um utilizador autenticado acede ao ecrã de cupões. O Portal faz um pedido Hypertext Transfer Protocol (HTTP) ao servidor para receber a listagem de cupões.

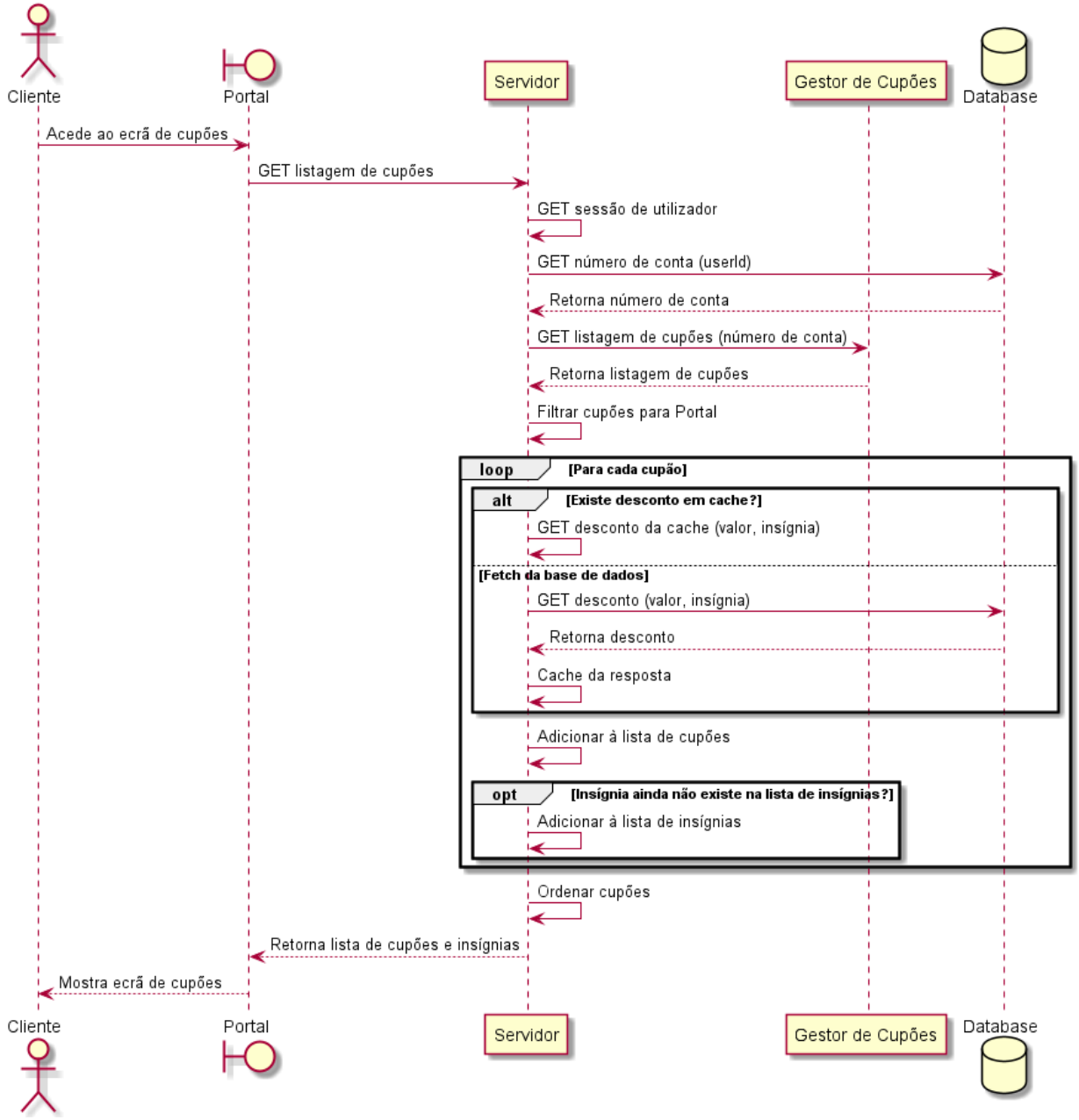

Figura 4.4: Vista de processo do UC01 (3.1.1)

O servidor obtém o identificador do utilizador a partir da sessão e usa-o fazer receber o número de conta da base dados. Com o número de conta é possível chamar o serviço externo de gestão de cupões e receber a listagem dos mesmos. Esta lista é filtrada para retornar apenas os cupões que têm a designação específica do Portal. Para cada elemento desta lista, é feita uma *query* para obter o desconto associado. Uma vez que é expectável que estes descontos se repitam e de forma a não sobrecarregar a base de dados é guardada uma cópia em memória do mesmo, de forma a poder ser reutilizada numa próxima iteração. É acrescentado um novo elemento à lista de insígnias sempre que surge um cupão cuja insígnia ainda não foi processada. Por fim, o servidor ordena os cupões e retorna-os em JavaScript Object Notation (JSON) para a interface.

UC02 Imprimir cupões (3.1.1)

O processo de impressão de cupões é iniciado quando um utilizador autenticado clica no botão de impressão na página de listagem de cupões. Conforme a vista de processo em 4.5, os dados necessários (cupões e termos e condições) são agrupados numa página a ser convertida para PDF por um *plugin* externo. Este PDF fica disponível para *download* por parte do cliente.

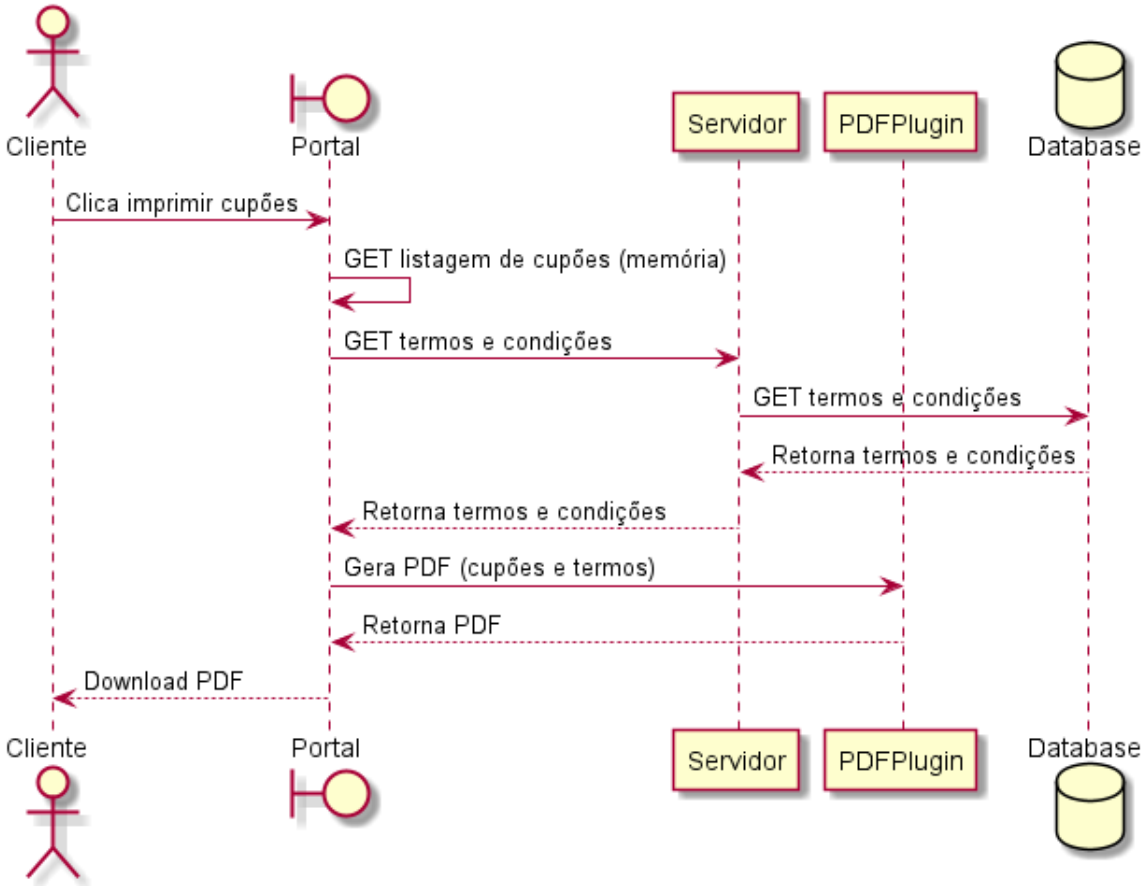

Figura 4.5: Vista de processo do UC02 (3.1.1)

UC03 Fazer login (3.1.1)

Um utilizador não autenticado pode clicar em "A minha conta"na barra de navegação do Portal para iniciar sessão. A figura 4.6 apresenta a vista de processo deste fluxo. Primeiramente, o utilizador insere o número de telemóvel e indicativo (método principal de identificação de cliente no Portal). O servidor vai gerar e guardar um *token* que vai ser enviado para o telemóvel do cliente através de um fornecedor externo de envio de SMS. O utilizador vai inserir o *token* que recebeu e o servidor vai validar contra o guardado na base de dados e consultar o estado da conta do cliente. Se a conta existir, é pedido o PIN de acesso. Se o PIN introduzido corresponder ao guardado na base de dados, o *login* é concluído com sucesso e o utilizador é redirecionado para a área autenticada. Se, por outro lado, não existir uma conta associada ao número de telemóvel validado, o utilizador será redirecionado para o fluxo de associação - UC04.

UC04 Fazer associação (3.1.1)

Um utilizador não autenticado e cujo identificador (número de telemóvel) não esteja associado a nenhuma conta pode entrar num processo de associação. O diagrama de sequência apresentado em 4.7 ilustra este fluxo. Em primeiro lugar, o utilizador deve inserir o cartão continente ao qual deseja se associar. O servidor gera um token aleatório e envia-o para o email do titular do cartão. O utilizador insere o token pedido e o servidor valida esse token contra o gerado anteriormente. Validado o token, são pedidas algumas informações adicionais como o email e o nome que serão associadas à conta. Por fim, o utilizador é redirecionado para a área autenticada.

UC05 Configurar descontos (3.1.1)

Um operador de *Backoffice* autenticado pode criar ou editar descontos. O diagrama de sequência apresentado em 4.8 ilustra este fluxo. Ao aceder ao ecrã de configuração de desconto, o servidor faz, concomitantemente, dois pedidos à base de dados para obter o desconto (em caso de edição) e para obter a lista de insígnias existentes, das quais o utilizador vai escolher uma para associar ao desconto. Estes dados são devolvidos para a aplicação, que os apresenta num formulário. O operador preenche o formulário. Os dados são validados, numa primeira instância do lado do cliente e, depois, revalidados do lado do servidor. As informações são persistidas na base de dados e o utilizador é informado sobre o sucesso da operação.

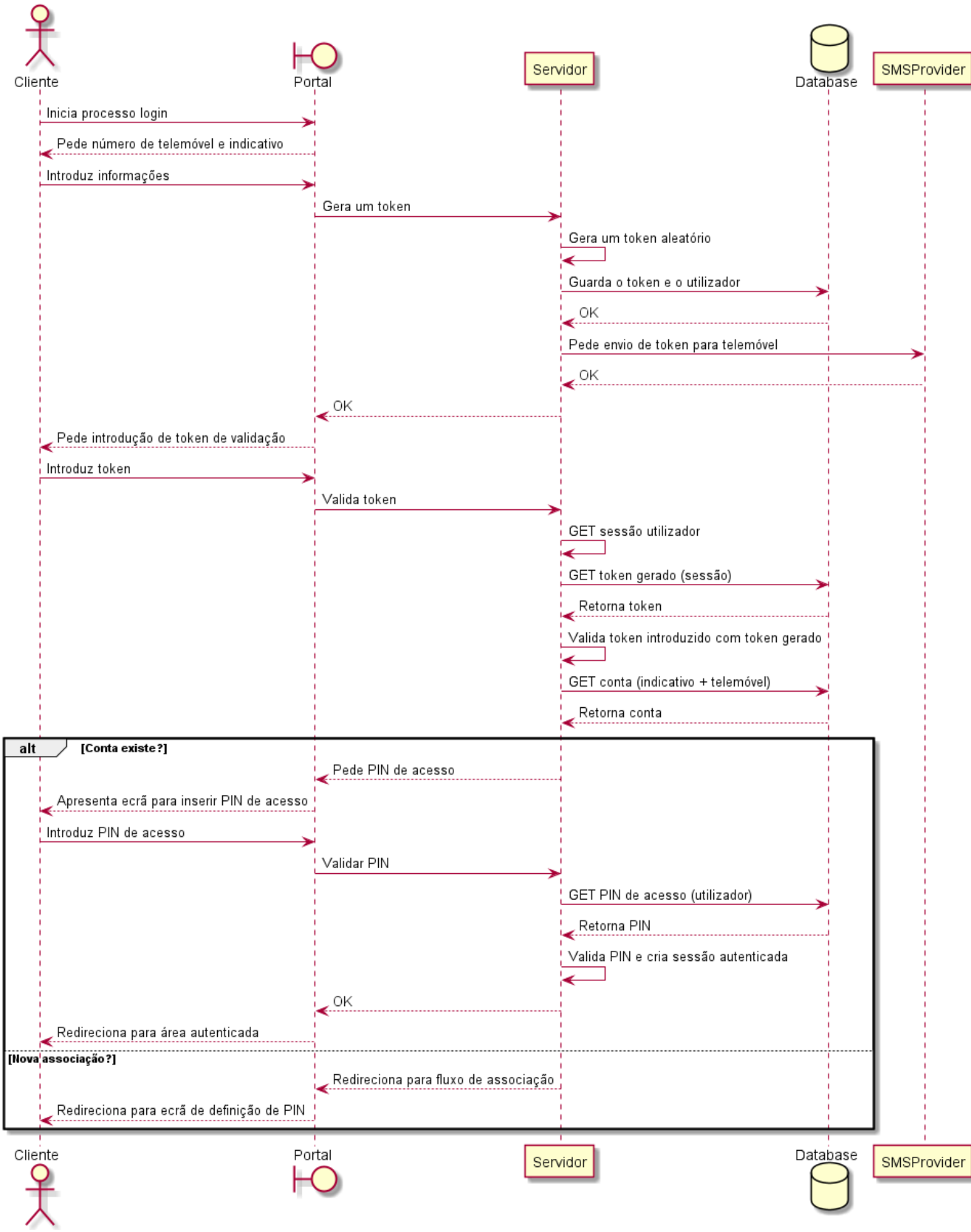

Figura 4.6: Vista de processo do UC03 (3.1.1)

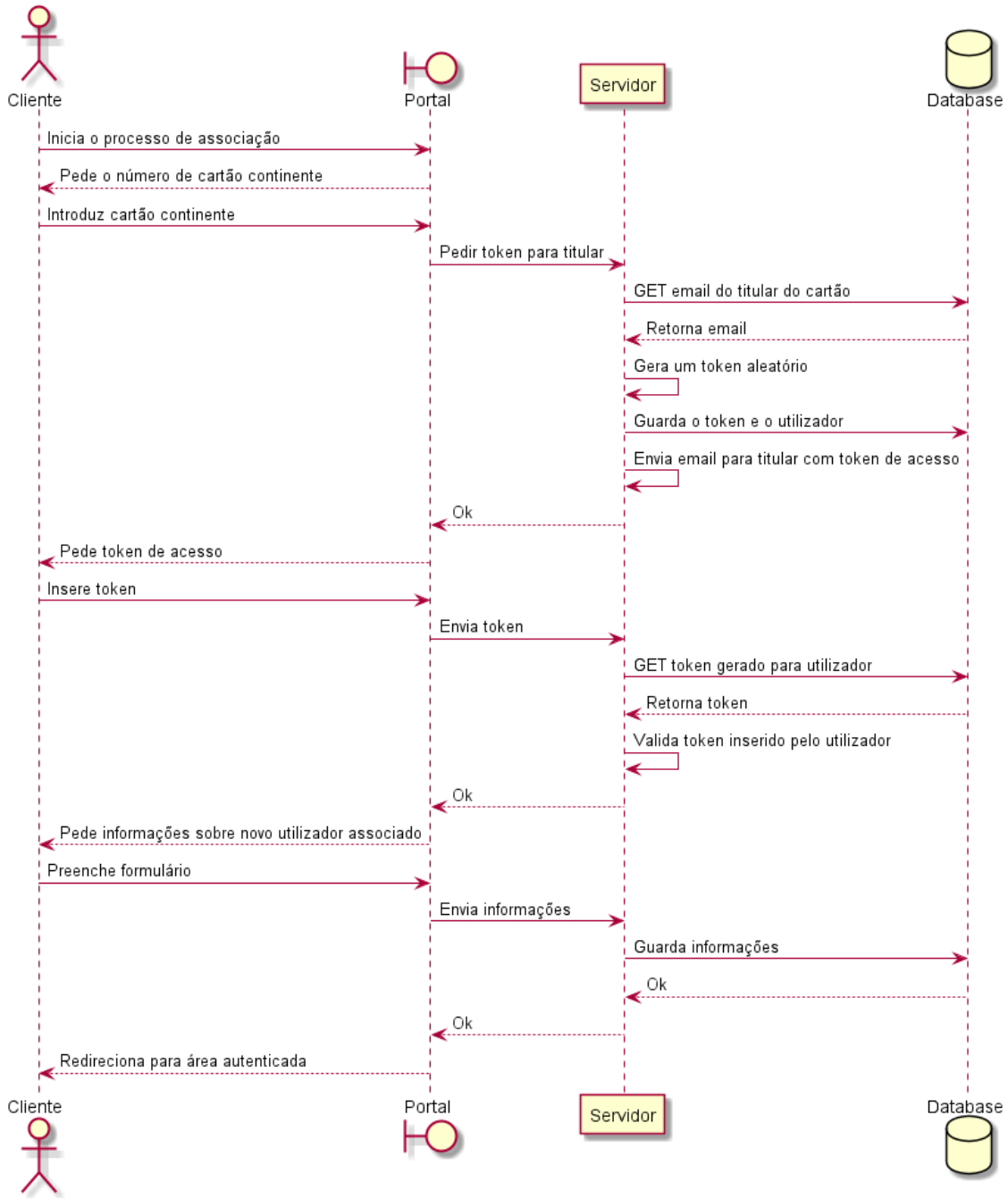

Figura 4.7: Vista de processo do UC04 (3.1.1)

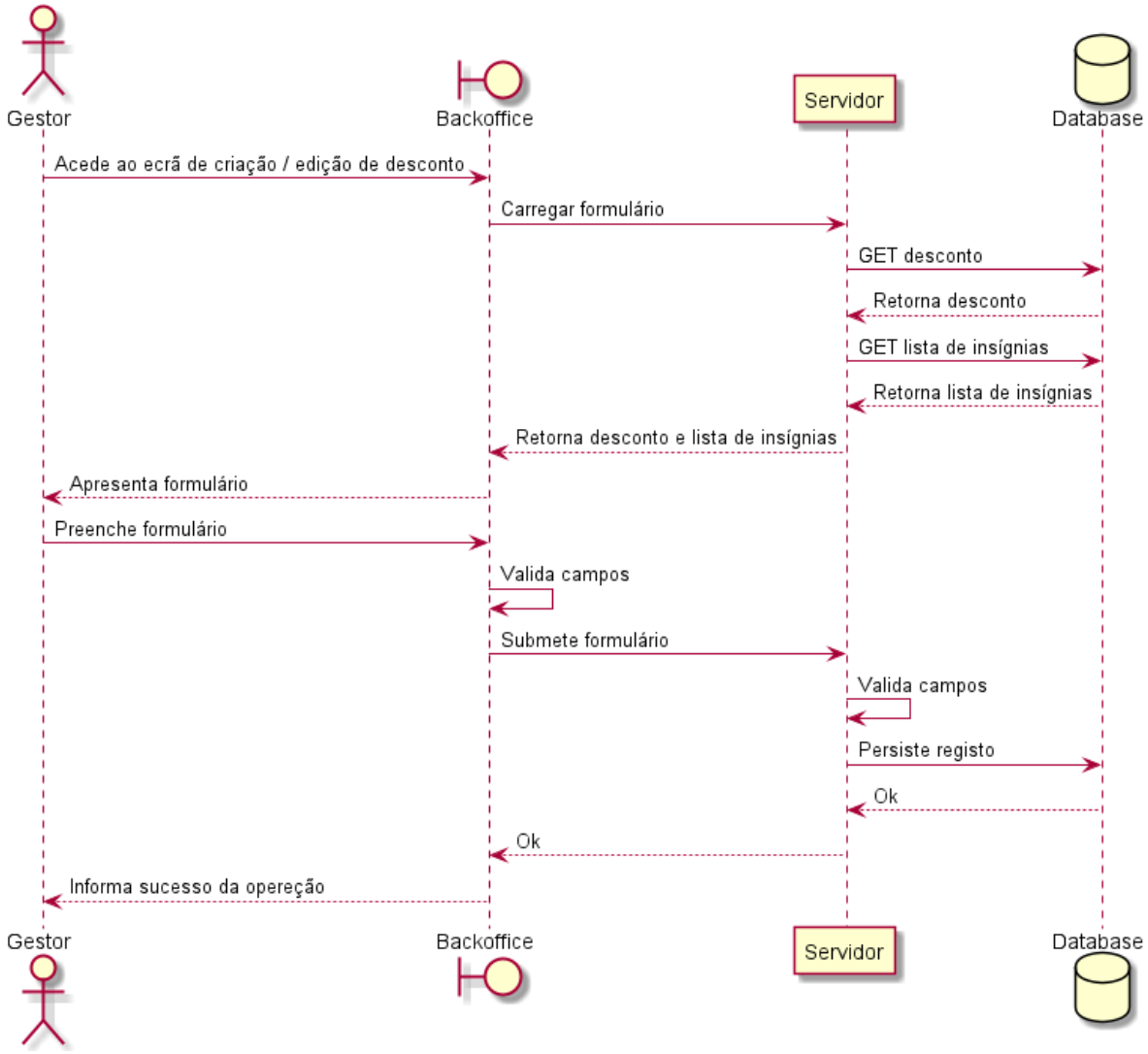

Figura 4.8: Vista de processo do UC05 (3.1.1)

Capítulo 5

Implementação

Este capítulo descreve o processo de implementação em OutSystems e Mendix de dois dos casos de uso apresentados no capítulo 3 Engenharia de requisitos. As outras funcionalidades foram implementadas em OutSystems e parcialmente implementadas em Mendix. Os casos de uso UC01 Visualizar cupões e UC05 Configurar descontos foram escolhidos para apresentar neste capítulo de forma a dar uma visão holística das funcionalidades das plataformas. Por um lado, o UC01 ilustra as capacidades relacionadas com uma implementação bastante customizada, direcionada a cliente final. Tem componentes de consumo de API's externas, interação com base de dados, processamento no servidor, gestão de sessões e autorizações, *caching* e comunicação cliente-servidor. Por outro lado, com a implementação do UC05 é pretendido demonstrar as capacidades das plataformas em estudo em relação a desenvolvimentos *boilerplate*, uma vez que se trata de um formulário direcionado a utilizadores internos, sem grandes preocupações a nível de customização da apresentação ou experiência de utilizador. Serão investigadas principalmente as componentes de geração automática de páginas e lógica a partir de um modelo de dados, assim como o uso de *templates*, sem divergir das funcionalidades nativas a cada plataforma.

É importante referir que foram utilizadas licenças diferentes para cada uma das plataformas. No caso de OutSystems, uma vez que o projeto foi desenvolvido na infraestrutra da Sonae, foi utilizada uma licença empresarial, sem restrições. Para Mendix foi utilizada a licença gratuita, que restringe o alcance da solução desenvolvida a nível de utilizadores finais (Mendix 2022e). Para este trabalho de investigação e comparação de funcionalidades, esta limitação não foi considerada relevante.

Quanto à organização do presente capítulo, em primeiro lugar será apresentada a abordagem adotada para todo o processo de implementação nas duas plataformas em estudo. De seguida, é descrita a implementação segundo a abordagem preconizada.

5.1 Abordagem

Foi formalizada uma abordagem de modo servir de guia no processo de implementação em ambas as plataformas em estudo. Numa primeira fase, a plataforma é estudada com base na documentação disponibilizada, assim como através da análise de aplicações exemplo encontradas na *forge* de OutSystems e no *Marketplace* de Mendix. Com base neste estudo, são descritos os principais conceitos específicos a cada ferramenta de modo a facilitar a compreensão e justificar as decisões de implementação. De seguida, será definida a arquitetura de módulos e aplicações seguindo as melhores práticas de cada plataforma. A solução implementada será explanada através de excertos de código visual.

A implementação terá por base o desenho proposto em 4 Desenho da solução, documentando eventuais deviações e limitações, analisando a concordância com o desenho preconizado.

De forma sistematizada, a descrição da implementação em cada uma das plataformas será composta pelas seguintes partes:

- Principais conceitos da plataforma
- Arquitetura específica para a plataforma
- Descrição da implementação
- Limitações e ajustes

Uma vez que o Portal atual está desenvolvido em OutSystems, na secção relativa a esta plataforma será também feito um breve estudo sobre a solução atual.

5.2 OutSystems

Esta secção apresenta a descrição da implementação dos casos de uso escolhidos na plataforma OutSystems segundo a abordagem apresentada em 5.1, assim como uma análise da solução atual, identificando pontos de melhoria para a nova implementação.

5.2.1 Conceitos

Uma aplicação é constituída por um conjunto de módulos definidos no *Service Studio* que constitui a unidade mínima de *deployment* no *Lifetime* - ferramenta responsável pela gestão do ciclo de vida de aplicações (OutSystems 2021c).

Uma aplicação pode ser dos seguintes tipos (OutSystems 2022a):

- Reactive Web App - aplicação SPA direcionada para a *Web*.
- Mobile App - aplicação móvel que pode ser instalada em Android e iOS.
- Tradional Web App - aplicação Web gerida do lado do servidor.
- Service - aplicação de *backend* para abstrair conceitos de negócio e expor funcionalidades para outras aplicações, seguindo uma arquitetura orientada a serviços.

Um módulo representa, em termos de arquitetura, o conceito mais pequeno. São normalmente usados para segregar as funcionalidades relativas a um conceito de negócio específico.

Um módulo pode ser dos seguintes tipos:

- Tipo principal da aplicação (Reactive, Mobile, Traditional ou Service) onde deve ser desenvolvida a interface.
- Blank - módulo vazio que permite a criação de qualquer tipo de elementos: de processo, interface, lógica e dados.
- Lirary - módulo que ajuda a definir as fundações para as aplicações OutSystems. Estes devem ser agnósticos a lógica de negócio e especificidades de aplicações. São normalmente usados para criar componentes de UI e funções de utilidade genérica, por exemplo funções matemáticas.

- Extension - módulo que faz de ponte com uma extensão criada fora da plataforma OutSystems, em C#, encapsulando o código criado em ações reutilizáveis que ficam disponíveis para todo o ecossistema da plataforma.

A organização de vários módulos e aplicações definem a arquitetura OutSystems. Para ajudar neste conceção, é recomendado a aplicação do *Architecture Canvas*. O *Architecture Canvas* ou *3 Layer Canvas* é uma ferramenta para facilitar o desenho de arquiteturas orientadas a serviços, promovendo a correta abstração de serviços e isolação de módulos funcionais e reutilizáveis. Desta forma, os módulos e aplicações vão preservar ciclos de vida independentes, reduzindo ao mínimo as interdependências e impactos de mudança, o que resulta numa arquitetura mais fácil de evoluir e manter (OutSystems 2022d).

O *Architeture Canvas* está dividido em três níveis: interface e processos de utilizador (ou *enduser*), conceitos de negócio e utilidades e integração (fundação). No nível mais acima - direcionado a utilizador final - são construídas interfaces e processos de utilizador, reutilizando lógica dos dois níveis abaixo para implementar os requisitos funcionais e não funcionais. No seguinte, o *core*, são construidos serviços em volta de conceitos de negócio, exportando entidades e regras de negócio, abstraindo a complexidade do primeiro nível. Por fim, no nível de fundação, são construidos serviços reutilizáveis, agnósticos à lógica de negócio, usando normalmente módulos do tipo *Library* (OutSystems 2022c).

A figura 5.1 representa alguns dos subníveis possíveis de cada um dos três níveis principais apresentados.

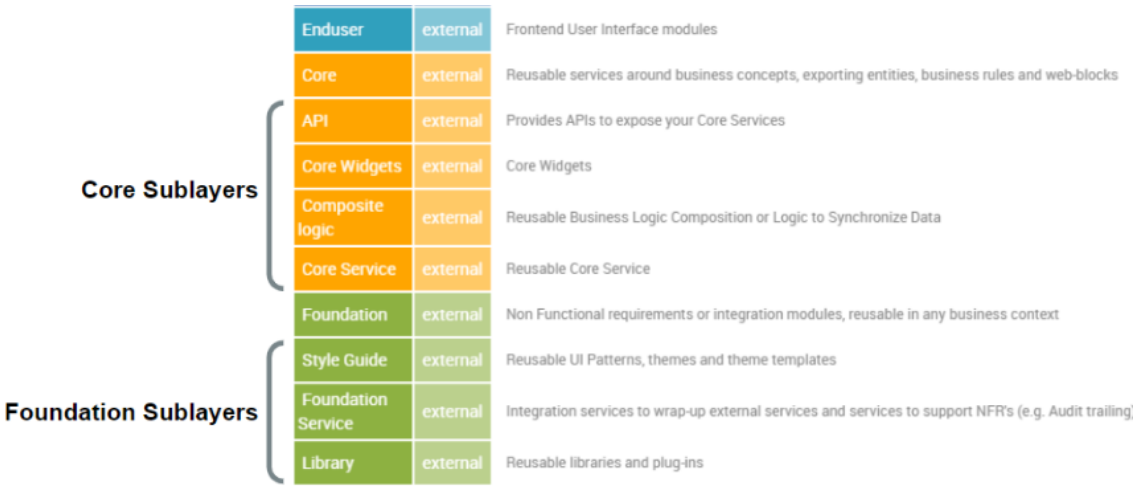

Figura 5.1: 3 Layer Canvas - Sublayers (OutSystems 2022)

Uma aplicação é classificada de acordo com o módulo de nível mais elevado que a constitui. Por exemplo, uma aplicação com um módulo de utilizador, dois módulos *core*, e cinco módulos de fundação vai assumir o nível mais alto e, por conseguinte, ser classificada como uma aplicação *enduser*.

De modo a validar a arquitetura construída podem ser usadas várias ferramentas internas a OutSystems como o *Discovery* e o *Architecture Dashboard*.

De uma forma geral, existem três regras fundamentais - ilustradas na figura 5.2. Um módulo nunca deve referenciar outro módulo de um nível acima, isto é, um módulo de fundação não pode consumir elementos de um módulo *core* ou *enduser* e um módulo *core* não deve consumir um módulo *enduser*. A violação desta regra significa normalmente que o elemento

consumido deve ser movido para um nível abaixo. Referências entre módulos do mesmo nível são possíveis nos níveis de *core* e fundação (uma vez que este devem ser reutilizáveis) mas não no nível de *enduser*. Por fim, nunca devem existir referências cíclicas, isto é, um módulo A não pode consumir um módulo B, se o módulo B já consumir o módulo A, uma vez que isto iria causar um estado de desatualização perpétua entre os módulos em questão (OutSystems 2022e).

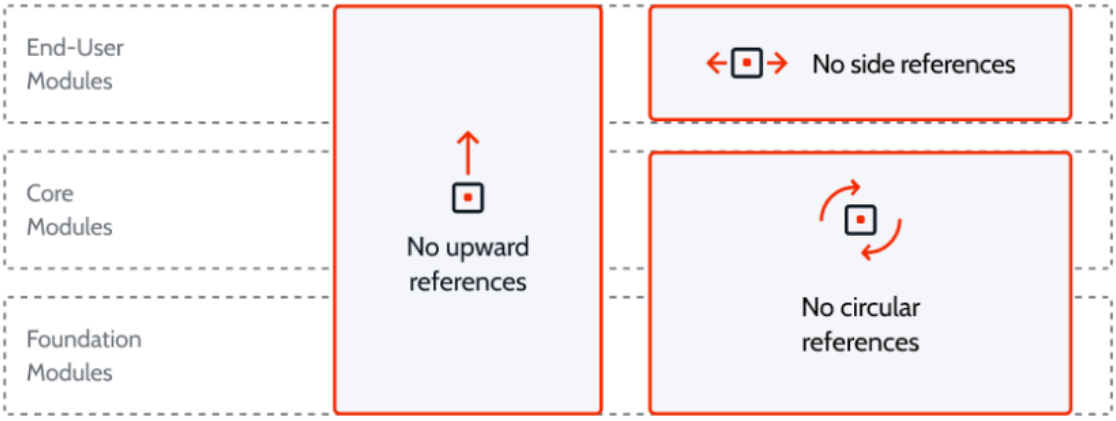

Figura 5.2: 3 Layer Canvas - Regras (OutSystems 2022)

Outro aspeto importante na definição dos módulos é o uso de convenções de nomenclatura. Ao seguir estas convenções, o propósito de cada módulo fica mais claro tanto para os programadores, como também para as ferramentas de classificação automática de módulos. Para isto, o nome de cada módulo deve estar associado a um sufixo, tal como (Arede 2021):

- UI - módulo que contém os ecrãs direcionados a cliente final.
- API - módulo que expõe API's para consumidores externos à plataforma.
- OAPI - módulo que expõe API's para outras aplicações OutSystems.
- CW - módulo que contém componentes de UI que encapsulam lógica de negócio.
- CS - módulo que contém entidades (tabelas) e lógica associada a esses conceitos.
- BL - módulo responsável por compor lógica de negócio distribuída em vários módulos CS.
- Sync - módulo responsável pela sincronização de dados de um sistema externo para um módulo CS.
- Pat - módulo que expõe padrões de UI reutilizáveis sem lógica de negócio.
- Th - módulo responsável por conter o tema da aplicação.
- IS - módulo que consome, encapsula e normaliza integrações com sistemas externos.
- Lib - módulo genérico que expõe funções reutilizáveis agnósticas a negócio.

Existem duas formas de expor funcionalidades de servidor num ambiente OutSystems: Server Actions e Service Actions. Ambas podem ser usadas em conjunto, e a escolha de uma ou de outra deve ser adequada a cada caso de uso, sendo um passo importante na definição da arquitetura de uma solução. As Server Actions criam uma dependência forte entre o módulo produtor e o módulo consumidor, pelo que são adequadas quando os ciclos de vida

destes módulos são similares. Por outro lado, as Service Actions criam uma relação de baixo acoplamento e são indicadas quando diferentes módulos podem ser promovidos a produção de forma independente (por exemplo, quando são geridos por equipas diferentes).

A tabela 5.1 resume as principais diferenças entre estes dois tipos de ações (OutSystems 2022b).

	Server Actions	**Service Actions**
Tipo de dependência	Forte	Fraca
Ciclos de lançamento	Mudanças na implementação obriga os consumidores a serem publicados.	Mudanças na implementação podem ser lançadas independentemente dos consumidores.
Comunicação entre serviços	Módulos produtores e consumidores correm no mesmo processo.	Módulos produtores e consumidores correm em processos diferentes e comunicam pela rede (HTTP).
Transações BD	Todas as transações da base de dados são feitas na mesma transação.	Diferentes processos requerem diferentes transações.
Esforço de desenvolvimento	Lógica simplificada e desenvolvimento rápido.	Requer lógica adicional para lidar com diferentes transações.

Tabela 5.1: Comparação entre Server e Service Actions

5.2.2 Arquitetura

Tendo em conta os conceitos e melhores práticas apresentados em 5.2, foi definida a arquitetura específica da solução a construir para a plataforma OutSystems, incluindo todos os casos de uso a desenvolver.

A figura 5.3 representa a arquitetura a nível de módulos. Os módulos foram divididos pelos principais conceitos de negócio presentes na aplicação.

Módulos específicos à aplicação a desenvolver contêm o prefixo 'Cartão', enquanto módulos que podem ser reutilizados têm um nome genérico. No caso dos módulos 'UserProfile_IS' e 'LoyaltyManager_IS' foi optado usar o nome do sistema externo em vez da funcionalidade que expõem (utilizadores e cupões, respetivamente), uma vez que no universo Sonae estas peças do ecossistema já estão naturalmente divididas pela área de negócio a que dão resposta.

Ainda que logicamente separados, os conceitos Utilizador, Cupão e Insígnia são usados em conjunto para responder a casos de uso como a visualização de cupões, pelo que foi criado também um módulo orquestrador - Cartao_BL - que é responsável por consumir e agregar resultados destes três módulos.

Foi considerada a criação de um módulo 'Cartao_CW' (*Core Widgets*) para encapsular lógica de negócio mas, uma vez que a complexidade atual da solução não o requer, optou-se por deixar esta responsabilidade ao nível dos módulos de UI.

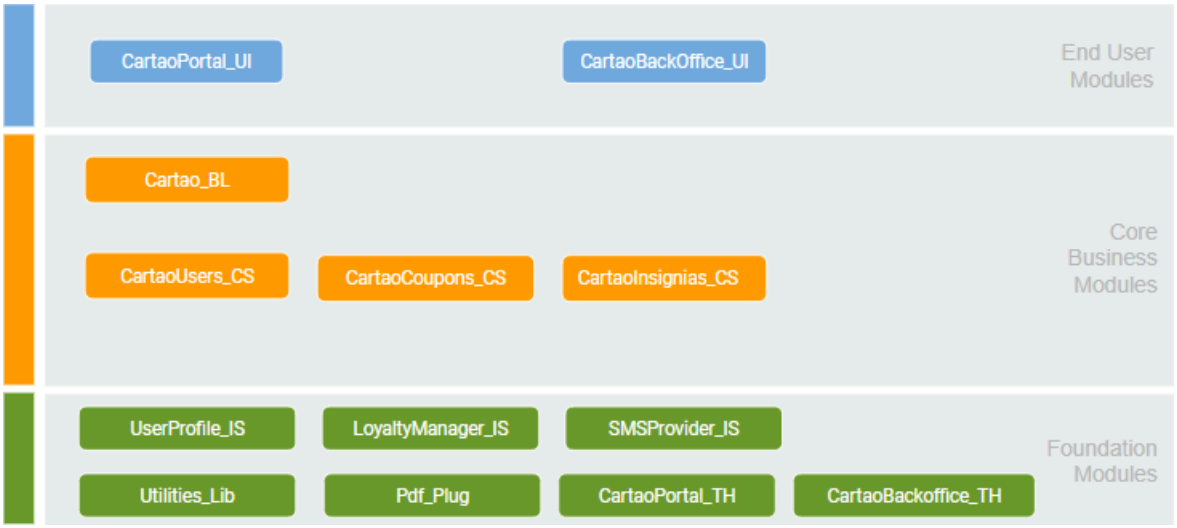

Figura 5.3: Arquitetura de módulos - OutSystems

A figura 5.4 apresenta a arquitetura desenvolvida em OutSystems a nível das aplicações, em que cada quadrado (que representa uma aplicação) contém um ou vários módulos. Uma vez que os módulos de tema (TH) são específicos a cada módulo de *enduser*, estes foram agrupados numa aplicação. A nível de *core*, todos os módulos, ainda que representem conceitos diferentes, seguem o mesmo ciclo de vida e estão bastante interligados, pelo que foram também agrupados numa aplicação 'CartaoCore'. Pela mesma razão, neste nível foi optado pelo uso de Server Actions entre este módulos. Por fim, cada módulo de fundação foi isolado na sua própria aplicação, uma vez que são independentes e podem seguir o seu próprio ciclo de vida. Deste modo, estas aplicações expõe Service Actions, de modo a criar dependências fracas com os seus consumidores, como explicado na tabela 5.1.

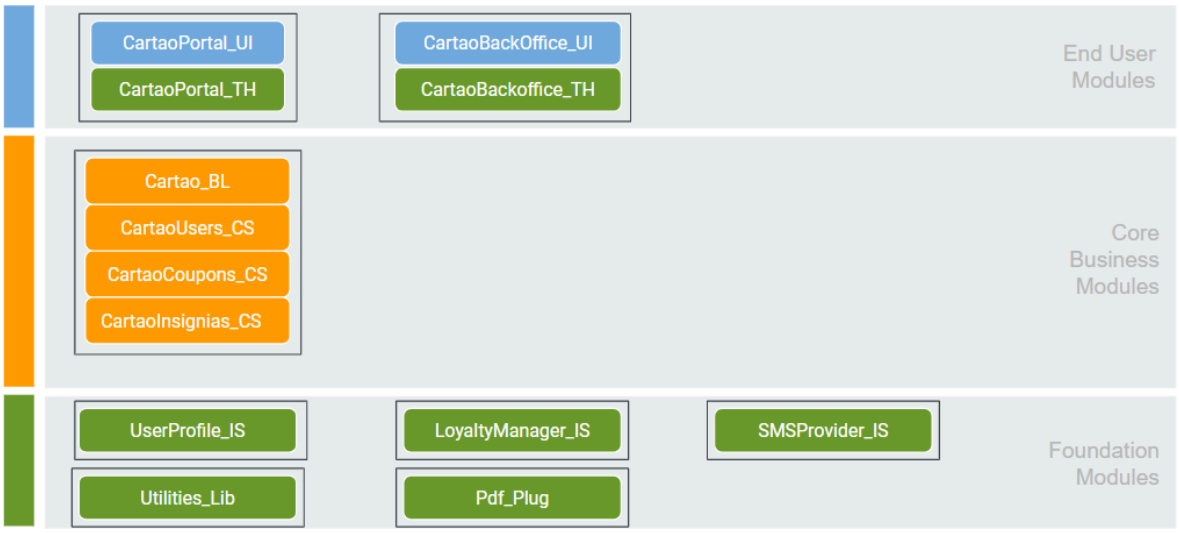

Figura 5.4: Arquitetura de aplicações - OutSystems

5.2.3 Análise da solução atual

Da solução que se encontra atualmente em produção, desenvolvida em *Traditional Web*, importa analisar com mais atenção o preponderante ponto de *bottleneck* - a página de apresentação dos cupões - de forma a identificar os principais problemas de desempenho e, com base nessa informação, implementar um algoritmo mais eficiente em termos de tempo de processamento.

A figura 5.5 apresenta um excerto da implementação atual da *Server Action* que retorna a listagem de cupões e insígnias.

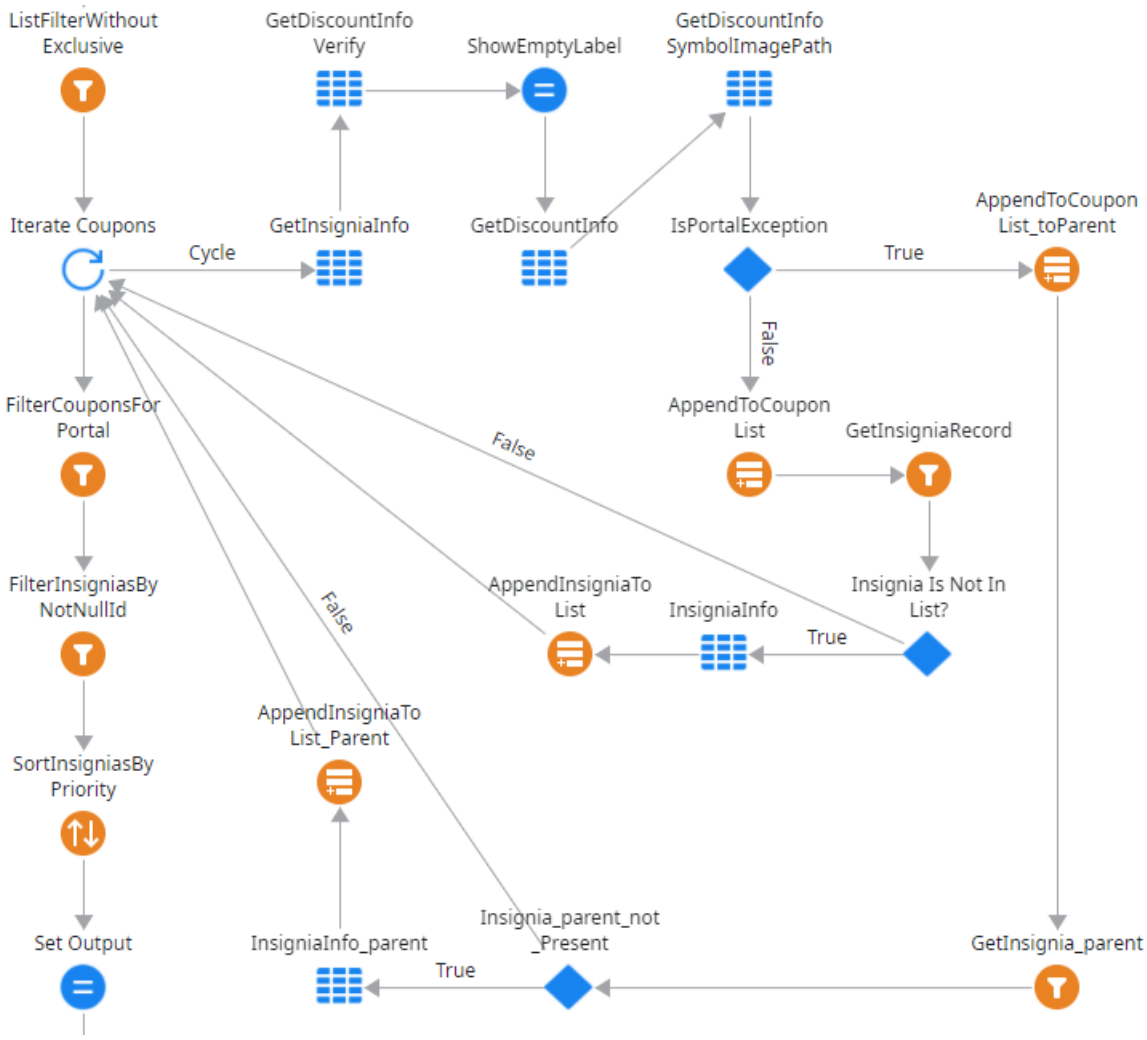

Figura 5.5: Excerto da implementação atual da *Server Action* que retorna a listagem de cupões e insígnias

O principal problema encontrado foi o número excessivo de acessos à base de dados feitas dentro do ciclo. Como podemos observar pela figura, são sempre feitas três *queries* para ir buscar informações relativas a descontos e uma ou duas *queries* à tabela de insígnias, dependendo se esta já foi adicionada à lista de retorno ou não. Mesmo considerando o melhor cenário, por cada cupão estão a ser feitas quatro chamadas à base de dados, o que rapidamente se tornam dezenas de acessos à camada de persistência num curto espaço de tempo. Pressupondo uma média de vinte cupões por conta, sempre que um utilizador entrar na página dos cupões são executados oitenta (20x4) pedidos. Convergindo estes dados

com a média de acessos diários ao Portal, cerca de três mil, e com o facto de cada sessão poder visitar esta página várias vezes, conclui-se que este processo deve ser aprimorado não só numa ótica de melhoria de experiência de utilizador, mas também numa perspetiva de otimização dos recursos do servidor e base de dados.

Além do problema discutido acima, foi também identificado um problema na forma como os cupões estão a ser filtrados. Na figura 5.5 são feitos três processos distintos de filtragem: remoção de cupões exclusivos para a aplicação móvel (ListFilterWithoutExclusive), remoção de cupões sem insígnia associada (FilterCouponsForPortal) e remoção de insígnias em estado inválido (FilterInsigniasByNotNullId). Cada filtragem representa uma leitura completa da lista. Uma vez que estas listas estão a ser construídas dinamicamente no ciclo, todas estas verificações podem ser feitas durante as iterações.

A nível da interface gráfica, foi encontrado bastante código duplicado. Devido a limitações do modelo aplicacional 'Traditional Web' discutidas anteriormente, para tornar o Portal responsivo para vários tipos de dispositivos com diferentes resoluções de ecrã (computadores, *tablets*, telemóveis), foi necessário implementar os componentes adaptados a estes três *breakpoints*, o que poluiu o código aplicacional e provou-se um desafio para manter e evoluir. Adicionalmente, para o correto posicionamento dos elementos ao redimensionar a janela, é necessário uma atualização completa da página, o que provoca uma experiência frustrante para o utilizador.

Conclui-se, portanto, que os principais focos para a nova implementação da página dos cupões são:

- Redução do número de acessos à base de dados
- Redução do número de iterações feitas às listagens
- Maior reutilização de código relativo à interface gráfica

5.2.4 Implementação

Nesta subsecção é descrita em detalhe a implementação dos dois casos de uso escolhidos, recorrendo à plataforma OutSystems. A implementação tem por base a descrição dos requisitos funcionais e não funcionais apresentados em 3, o desenho da solução em 4 e a arquitetura específica apresentada em 5.2.2.

Visualizar cupões

Partindo da análise feita em 5.2.3 e o conhecimento da área de negócio, foram definidos os seguintes premissas de modo a guiar a nova implementação:

- **A**: Num determinado momento um cliente tem cerca de vinte cupões
- **B**: Um cliente tem maioritariamente cupões da insígnia Continente
- **C**: Um cliente tem cupões de cerca de cinco insígnias
- **D**: Os cupões vêm preordenados pela insígnia
- **E**: Os cupões têm diferentes descontos associados
- **F**: Vários clientes recebem os mesmos descontos

Com base nas premissas **B**, **C** e **D** pode-se concluir que um mecanismo eficaz para a redução na latência da resposta é o uso de *cache* na consulta da informação da insígnia à base de dados. Uma vez que, durante o processamento dos cupões, a insígnia é repetida várias vezes este é um dos casos que pode beneficiar de guardar a informação na memória do servidor, de modo a evitar ao máximo chamadas extra à camada de persistência.

Por outro lado, com base em **E** é inteligível que na consulta dos descontos o mesmo mecanismo de *cache* não será tão vantajoso na ótica de processamento singular (apenas um utilizador). No entanto, considerando que a aplicação é acedida por milhares de utilizadores e vários clientes recebem os mesmos descontos (**F**), a *cache* de descontos no servidor também se manifestará numa melhoria de desempenho.

A figura 5.6 apresenta um excerto da nova implementação da lógica de servidor para retornar os cupões de um cliente.

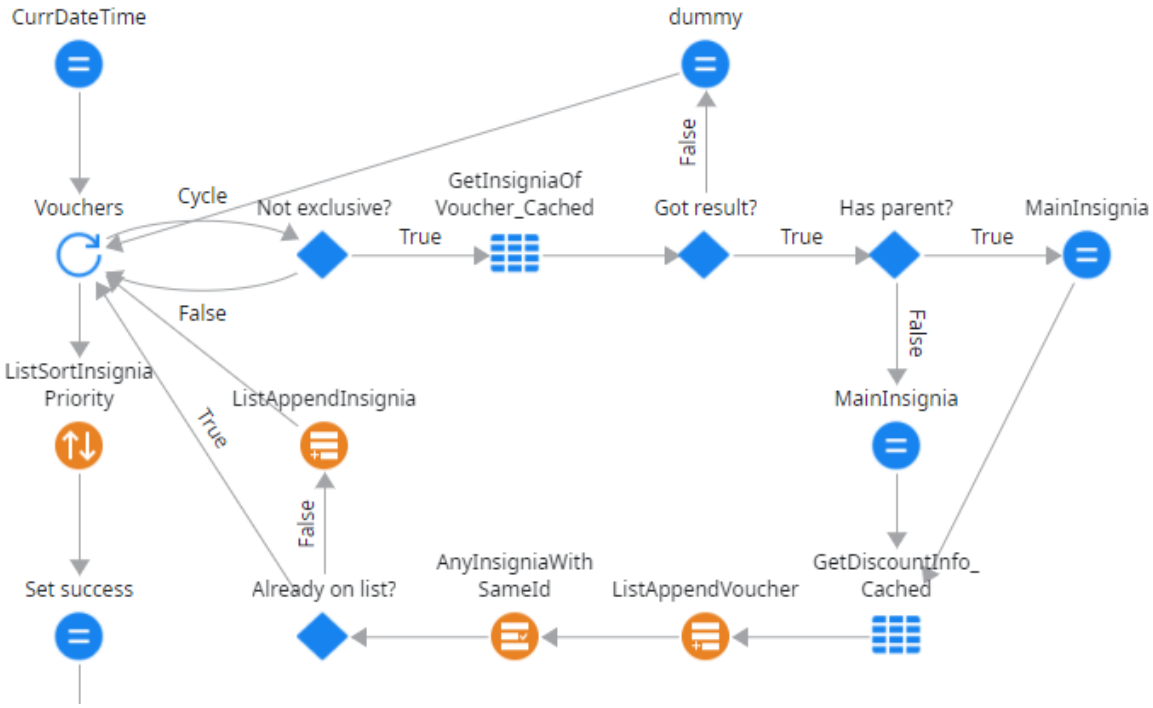

Figura 5.6: Excerto da nova implementação da *Server Action* que retorna a listagem de cupões e insígnias

Nesta nova abordagem a lógica foi simplificada, tendo os seguintes impactos:

- Redução de três consultas por cupão à tabela de descontos para uma (com *cache*)
- Redução de duas consultas por cupão à tabela de insígnias para uma (com *cache*)
- Maior gasto de memória ao manter *cache* no servidor
- Otimização do cálculo de atributos não dependentes da iteração, sendo movidos para fora do ciclo
- Remoção de três filtragens à lista resultante
- Melhoria no método usado para encontrar uma insígnia numa lista (*ListAny* em vez de *ListFilter*)

- Redução dos campos enviados como retorno da função, de modo a incluir apenas os necessários para a apresentação

As melhorias trazidas pela refatorização deste código vão ser discutidas mais à frente, no capítulo 6.

Em relação à interface gráfica, esta foi desenhada de acordo com os *mocks* apresentados no capítulo 3. Engenharia de requisitos. A figura 5.7 representa o resultado final desta página, onde é possível ver informações relativas ao saldo e aos cupões das diferentes insígnias. Como se tratou de um desenvolvimento personalizado, não foi possível utilizar muitos dos aceleradores que OutSystems disponibiliza. No entanto, tirou-se partido da *framework* de UI - OutSystems UI - que oferece vários padrões úteis ao desenvolvimento de *layouts*. Para a divisão do conteúdo de saldo e cupões foi usado o componente 'ColumnsSmallLeft', para a listagem de insígnias utilizou-se os padrões de listagem e 'ScrollableArea' e para a listagem de cupões foi utilizado o padrão de galeria 'Gallery'.

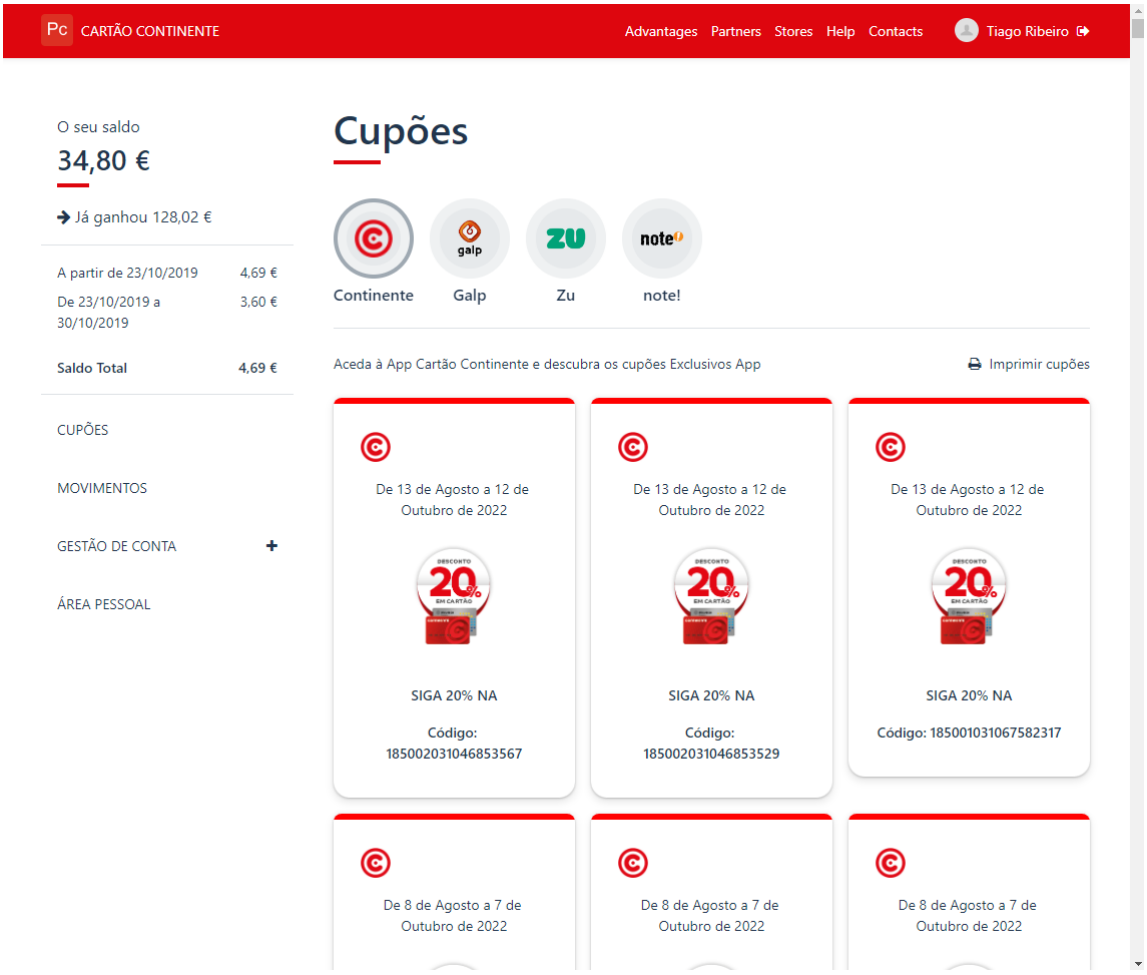

Figura 5.7: Página de listagem de cupões - vista Desktop

Segundo a filosofia de desenvolvimento OutSystems, inspirada na *framework* utilizada para gerar o código - ReactJS - a página foi dividida em componentes funcionais. A lógica de cada área específica foi encapsulada num bloco (termo OutSystems para designar uma peça de UI reutilizável), permitindo a segregação das funcionalidades em componentes isolados, o que traz diversas vantagens a nível de manutenção. Desta forma, foi construído um componente

para mostrar a área do saldo, um componente para a navegação e um componente para a área principal da listagem de cupões. Dentro da listagem de cupões, existem outros componentes mais pequenos que gerem a lógica de apresentação como, por exemplo, o componente que representa cada insígnia e o componente que representa um cupão.

A figura 5.8 representa o estado intermédio de carregamento da página. Na versão atual do Portal, a página é construída do lado do servidor e enviada para o *browser*, o que se traduz numa demora percetível ao navegar entre páginas, especialmente quando o serviço externo que gere os cupões está sobre maior carga e, portanto, mais lento. Para colmatar este problema, nesta nova versão, a navegação é efetuada instantaneamente e é mostrado um estado de carregamento enquanto o conteúdo dinâmico é obtido. Para esta implementação foi usado o componente 'EasyPlaceholder' obtido na *forge* da comunidade OutSystems, que permite definir um esqueleto do conteúdo.

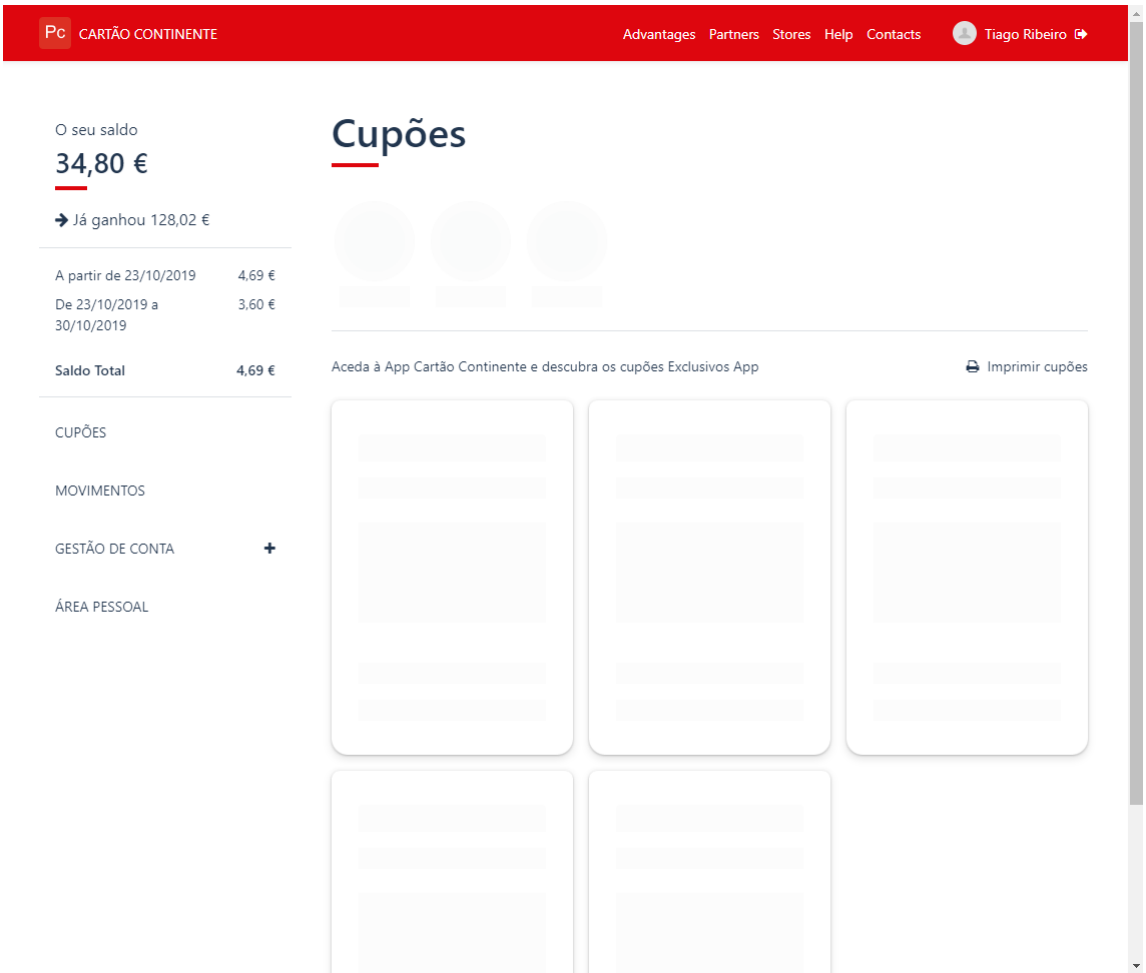

Figura 5.8: Página de listagem de cupões - vista Desktop em carregamento

Um dos requisitos mais importantes para a implementação deste caso de uso está ligado à experiência do utilizador em dispositivos móveis. Nesse sentido, foram escolhidos padrões de interface que permitissem a disposição adaptativa dos elementos. No caso do *layout*, os padrões 'ColumnsSmallLeft' e 'Gallery' foram úteis pois permitem definir comportamentos diferentes, consoante a resolução do ecrã. Na figura 5.9 (vista de um telemóvel) pode-se observar que a barra de navegação é substituída por um menu e a listagem de cupões apresenta apenas um por coluna.

Para melhorar o carregamento das páginas em equipamentos que recorrem a redes mais fracas, como por exemplo redes móveis, foi garantido que o menor número de recursos, nomeadamente *Javascript*, *CSS* e *JSON*, são enviados para o cliente.

Figura 5.9: Página de listagem de cupões - vista móvel (IPhone XR)

Configurar descontos

A implementação deste caso de uso foi suportada pelas capacidades de *scaffolding* da plataforma - possibilidade de gerar artefactos automaticamente, sem codificação manual.

OutSystems dispõe de funcionalidades para gerar ecrãs de listagem e detalhe com base numa entidade (tabela de base de dados). Estes ecrãs são baseados nas colunas da tabela principal, assim como as relações com outras tabelas através de chaves estrangeiras. A figura 5.10 apresenta o resultado desta operação que permitiu gerar uma listagem com paginação, em que é apresentado para cada desconto a insígnia correspondente, o valor e tipo de desconto e o momento em que foi criado. Para acrescentar mais uma coluna à listagem o programador precisa apenas de arrastar o campo correspondente da tabela para o editor WYSIWYG.

É possível navegar para o detalhe de um desconto carregando no nome ou no botão 'Add Discount'. A funcionalidade de pesquisa, apesar de presente na interface e no código gerado, não filtra corretamente os resultados, sendo necessária intervenção manual por parte do programador.

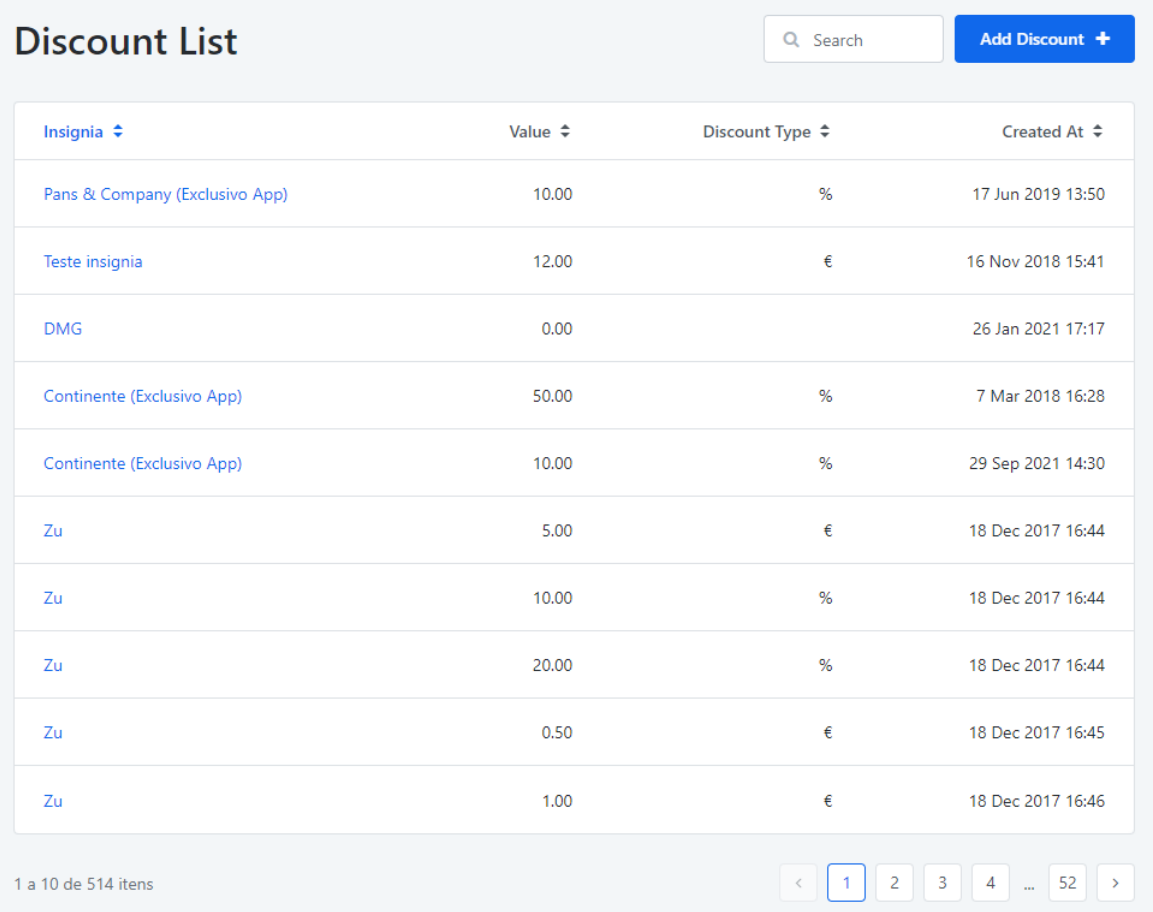

Figura 5.10: Página de listagem de descontos - Backoffice

Quanto ao ecrã de detalhe (figura 5.11), também gerado automaticamente, este apresenta um formulário com três campos: um decimal e dois em estilo de escolha múltipla. A plataforma teve a capacidade de gerar o código necessário distinguir a criação de um novo desconto e a edição de um existente e ir buscar à base de dados as informações necessárias para a apresentação das opções. Por defeito, mais campos foram apresentados no formulário

sendo estes removidos manualmente, uma vez que não podem ser editados pelo operador de *backoffice*. Como na listagem, mais campos podem ser adicionados ao formulário ao arrastar a coluna respetiva para o ecrã.

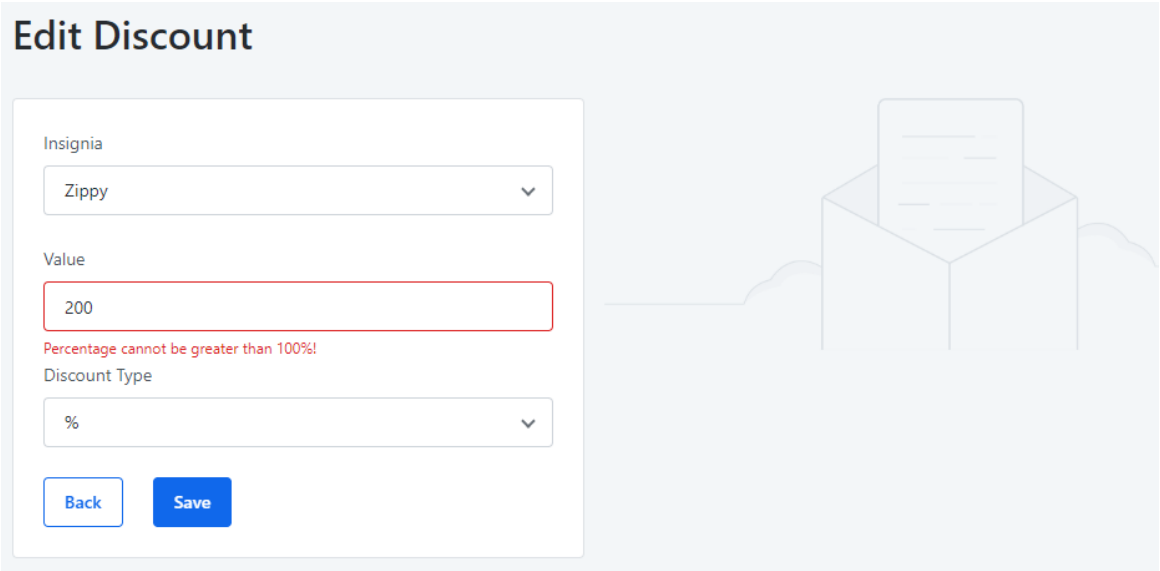

Figura 5.11: Página de detalhe de desconto - Backoffice

A lógica para persistir os dados não foi gerada automaticamente, portanto recorreu-se à implementação manual das regras de negócio e persistência na base de dados. Para validação dos dados inseridos foram considerados casos gerais como, por exemplo, que o valor inserido no campo 'Value' é um valor decimal, assim como casos particulares que dependem das opções escolhidas como o facto de um desconto em percentagem não poder ter o valor acima de 100%. Esta validações foram feitas do lado do cliente para melhorar o desempenho e experiência, assim como do lado do servidor para garantir a integridade dos dados face a possíveis pedidos maliciosos que evitem as validações do lado do cliente.

A figura 5.12 apresenta a lógica implementada do lado do cliente na ação de guardar. Caso alguma validação falhe, é apresentada uma mensagem de erro ao utilizador por baixo do campo correspondente. Em caso de sucesso, é feita uma chamada ao servidor para persistir os dados. No final, o utilizador é reencaminhado de volta para a listagem de descontos.

Na figura 5.13 é apresentada a lógica implementada do lado do servidor para persistir um desconto na base de dados. A tabela que guarda os descontos, assim como a lógica para interagir com os dados foi encapsulada no módulo *Core* CartaoInsignias_CS. Este módulo expõe apenas as ações de leitura, criação e atualização, escondendo a complexidade das mesmas atrás de uma *API* simples de utilizar. Desta forma, é garantido que todos os módulos consumidores estão agnósticos à lógica de negócio dos descontos e não conseguem fazer alterações inválidas.

Em primeiro lugar é usada a função 'IsBOUserOrThrow' para validar as permissões do utilizador que está a fazer o pedido, garantindo que é um utilizador autorizado para fazer esta ação. Uma exceção de segurança é lançada caso o utilizador não corresponder a um operador de *backoffice*. De seguida, são feitas as mesmas validações de integridade de dados feitas do lado do cliente. Caso alguma destas validações falhe, é lançada uma exceção 'BadRequest', que indica que o pedido foi mal formado. Por fim, existem duas ramificações no fluxo que tratam os dois casos possíveis: a criação de um novo desconto ou a atualização

de um já existente. Para o caso de atualização foi tomado o cuidado de primeiramente fazer uma *query* que bloqueia o registo para evitar atualizações concorrentes.

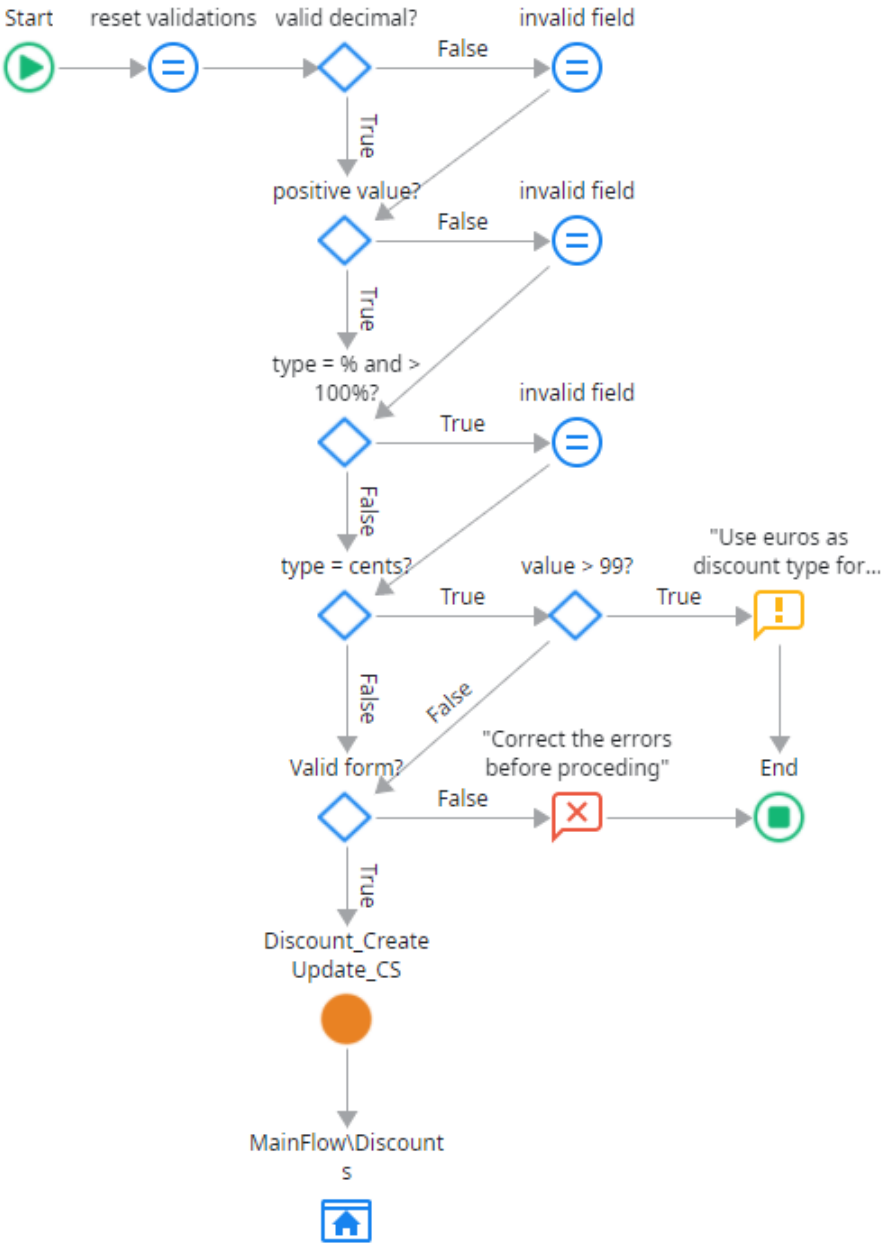

Figura 5.12: Lógica para guardar um desconto (cliente)

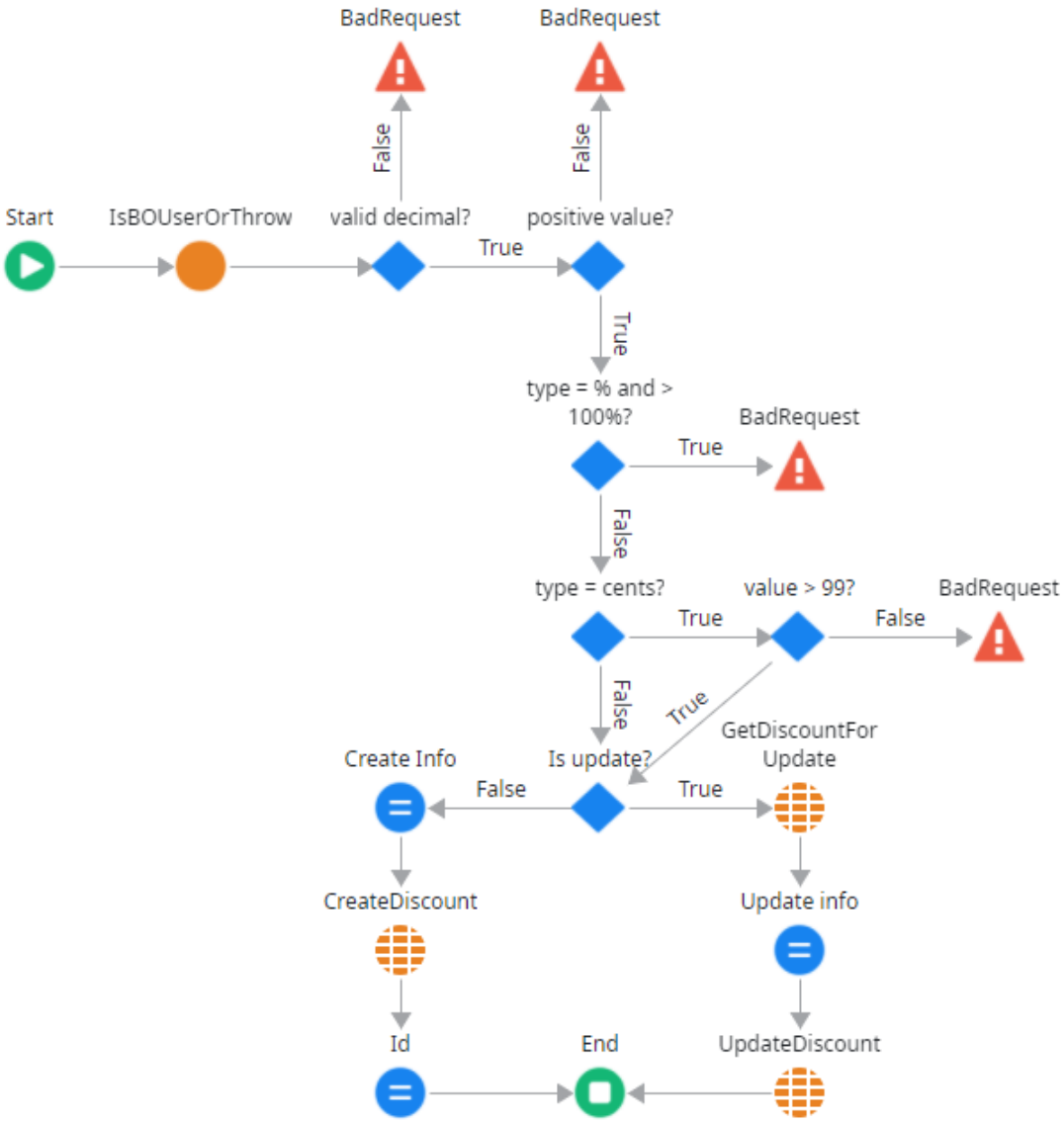

Figura 5.13: Lógica para guardar um desconto (servidor)

5.2.5 Limitações

A primeira limitação encontrada no desenvolvimento em OutSystems está relacionada com o editor de interfaces WYSIWYG. Apesar de permitir uma ideia do resultado final durante o criação da interface, este editor tem alguns problemas na renderização de cores, margens e outras propriedades CSS o que acaba por gerar confusão e frustração quando o programador tenta resolver problemas que só existem no IDE, o que resultou em muito do desenvolvimento ser feito a partir de alterações diretamente no *browser* que mais tarde foram copiadas para o IDE.

Ainda relacionado com o desenvolvimento de interfaces gráficas, uma das principais prioridades de *frameworks* modernas está na prototipagem rápida. Isto é alcançado com técnicas como o *hot reload* - técnica que permite atualizar o código e manter o estado da aplicação. Este comportamento não é possível em OutSystems, uma vez que para cada alteração é necessária a publicação do módulo, operação que demora tipicamente entre um a três minutos e que resulta numa perda de estado da aplicação. Este problema é exacerbado quando os elementos de UI estão divididos por vários módulos (como é recomendado para projetos de média e grande dimensão) onde é necessária a publicação sequencial dos mesmos, começando nos módulos produtores e acabando nos módulos consumidores.

Para evitar módulos de UI monolíticos, é recomendado que os ecrãs sejam segregados por diferentes módulos, consoante a área funcional de cada um. No entanto, ao consumir um ecrã de um módulo diferente, esse ecrã é considerado como uma aplicação diferente, o que obriga ao carregamento de todos os recursos que compõe a SPA novamente e que resulta numa transição lenta - precisamente o que se pretende evitar ao construir uma SPA.

Especialmente no desenvolvimento de lógica de *backend* foi notada a falta de *branching* dentro do ambiente de desenvolvimento. Algumas alterações num módulo, por exemplo alteração no contrato de uma ação, têm impactos imediatos nos módulos consumidores, tornando-os incompatíveis, o que impede a sua publicação. Da mesma forma, uma alteração que precise de ser disseminada por vários módulos deixa a aplicação como um todo num estado inconsistente e, possivelmente, não funcional. Principalmente numa perspetiva de desenvolvimento em equipa, esta limitação acarreta várias inconveniências. Esta limitação significa também que não podem coexistir várias versões de componentes como o OutSystemsUI, que serve de base para muitas aplicações, o que obriga a uma abordagem *Big Bang* em que ou todas as aplicações são atualizadas, ou nenhuma é.

Um problema encontrado no código gerado pela plataforma através do *scaffolding* foi a pesquisa na tabela de descontos. Além de não funcional, a pesquisa era refrescada a cada evento 'OnChange' do campo de texto. Na prática, isto significa que é feito um ou mais acessos à base de dados por cada evento de teclado, por exemplo cada letra inserida ou apagada. Esta abordagem resulta num aumento de carga e detrimento de desempenho da aplicação. Seria importante que a plataforma seguisse as boas práticas da indústria e aplicasse algum tipo de *debouncing* - técnica para disparar um evento apenas após alguns momentos de inatividade.

Na parte de integração com sistemas externos foi sentida a necessidade da plataforma ajudar no tratamento de erros, especialmente relacionados com a indisponibilidade do serviço. Políticas de *retry* têm que ser implementadas manualmente pelo programador e não existe forma automática de aplicar a mesma política a vários *endpoints*.

5.3 Mendix

Esta secção apresenta a descrição da implementação dos casos de uso escolhidos na plataforma Mendix segundo a abordagem apresentada em 5.1, fazendo alguns paralelos à implementação em OutSystems.

5.3.1 Conceitos

Uma aplicação Mendix é constituída por um conjunto de módulos definidos no *Mendix Studio Pro*. Ainda que a plataforma não obrigue a qualquer estrutura predefinida de módulos, os programadores são encorajados a usá-los como forma de segregar funcionalidades da aplicação em diferentes componentes. Uma aplicação Mendix pode ser dos seguintes tipos (Mendix 2022d):

- Web App - aplicação SPA direcionada para a Web.
- Native Mobile App - aplicação móvel (React Native) que pode ser instalada em Android e iOS e aceder às funcionalidades nativas de cada plataforma.

No processo de criação de uma aplicação, podem também ser escolhidos *templates* que servem como aceleradores para o desenvolvimento de funcionalidades específicas como, por exemplo, processos de integração de novos colaboradores, pedidos de compra ou até mesmo aplicações de realidade aumentada.

Por omissão, uma aplicação contém um módulo de sistema, um módulo de recursos de interface e um módulo de aplicação. Adicionalmente, o programador pode incorporar os seguintes tipos de módulos (Mendix 2021a):

- Módulo de aplicação (além do criado por defeito) - usado para dividir código em domínios funcionais. Pode ser usado tanto para *frontend* como para *backend*.
- Módulo de *add-on* - módulo independente usado adicionar funcionalidade a um módulo existente (por exemplo, um conector). Pode ser inspecionado pelos módulos consumidores. Semelhante a uma biblioteca em linguagens de programação tradicionais.
- Módulo de solução - tipo de modo utilizado para distribuição de uma solução (por exemplo, vender a um cliente). Pode apenas pertencer a uma aplicação de to tipo 'solução'.

É recomendado que cada módulo seja tratado como uma entidade autónoma e independente, que pode ser facilmente substituído por outro. Em relação à nomenclatura, Mendix não obriga a uma definição rigorosa, mas recomenda que os seus nomes sejam em Pascal Case (cada palavra que compõe o nome deve começar com letra maiúscula) e que identifiquem inequivocamente a responsabilidade do módulo, por exemplo, *ProductManagement* ou *SalesforceIntegration* (Mendix 2022b).

Em cada módulo é possível configurar as definições de segurança, que afetam todos os elementos contidos nesse mesmo módulo. Dentro de cada módulo é possível encontrar diferentes tipos de documentos, cada um com a sua Domain Specific Language (DSL). Por exemplo, em documentos de UI como páginas, podem ser encontrados elementos como caixas de textos e tabelas e em documentos de modelo de domínio, podem ser encontradas representações visuais das entidades (tabelas) da base de dados (Mendix 2021a).

Existem mais de vinte tipos de documentos, dentro dos quais os mais usados são:

- Page - usado para criar uma interface gráfica para o utilizador final. São compostas por diferentes componentes chamados *widgets*.
- Layout - estrutura base para uma página. Define o posicionamento de elementos comuns a todas as páginas, como o cabeçalho, o título e o rodapé.
- Snippet - componente reutilizável de interface. Encapsula lógica de apresentação e de negócio.
- Microflow - lógica aplicacional que corre no servidor. É composto por atividades que manipulam objetos e interagem com a base de dados e com o cliente.
- Nanoflow - lógica aplicacional que corre do lado do cliente. Usa a mesma DSL - baseada em Business Process Model and Notation (BPMN) - que o Microflow.

A organização destes vários documentos em módulos e aplicações diferentes definem a arquitetura Mendix. A plataforma não disponibiliza nenhuma ferramenta para validar a arquitetura, nem diretrizes rigorosas para a sua definição, dando bastante liberdade ao programador. Da mesma forma, a reutilização de lógica entre módulos e aplicações não está sujeita às mesmas restrições encontradas na plataforma OutSystems.

Existem duas formas principais de partilhar lógica e dados entre aplicações: expor uma Representational state transfer (REST) Application Programming Interface (API) ou promover funcionalidade a um módulo *add-on* - similar a uma biblioteca de uma linguagem de programação tradicional - que pode ser partilhado no *Marketplace* de forma privada ou pública. Entre módulos da mesma aplicação os recursos estão, por definição, disponíveis para todas as interações (Mendix 2021f).

5.3.2 Arquitetura

Em linha com o estudo dos conceitos da plataforma e com o desenho da solução definido em 4, foi definida a arquitetura específica da solução a construir para a plataforma Mendix, incluindo todos os casos de uso a desenvolver.

A figura 5.14 apresenta a arquitetura proposta a nível de módulos. Os módulos foram divididos pelos principais conceitos de negócio presentes na aplicação.

Os módulos a verde representam conceitos independentes do resto da aplicação, não apresentando qualquer lógica específica de negócio. Por esta razão, considerou-se a criação dos mesmos como módulos de *add-on*. Apesar de conceptualmente correta (visto que são módulos conectores e de integração), esta abordagem apresenta algumas limitações como, por exemplo, a impossibilidade de os editar recorrendo ao Mendix Studio e a impossibilidade de fazer *debug* (Mendix 2022a), pelo que foram criados com o tipo por defeito - módulo de aplicação. Os módulos rotulados 'Integration' consomem API's REST de sistemas do ecossistema SONAE, expondo-as com dados normalizados para as aplicações Mendix. O módulo 'PdfConnector' foi adicionado como dependência direta da aplicação 'CartaoPortal', como uma biblioteca.

A laranja estão os módulos da camada *Core* da aplicação, divididos por área funcional. Estes módulos interagem com os módulos de integração por API's REST e guardam informações na base de dados, expondo vários métodos (por API REST) para operações de listagem, criação e edição sobre as mesmas.

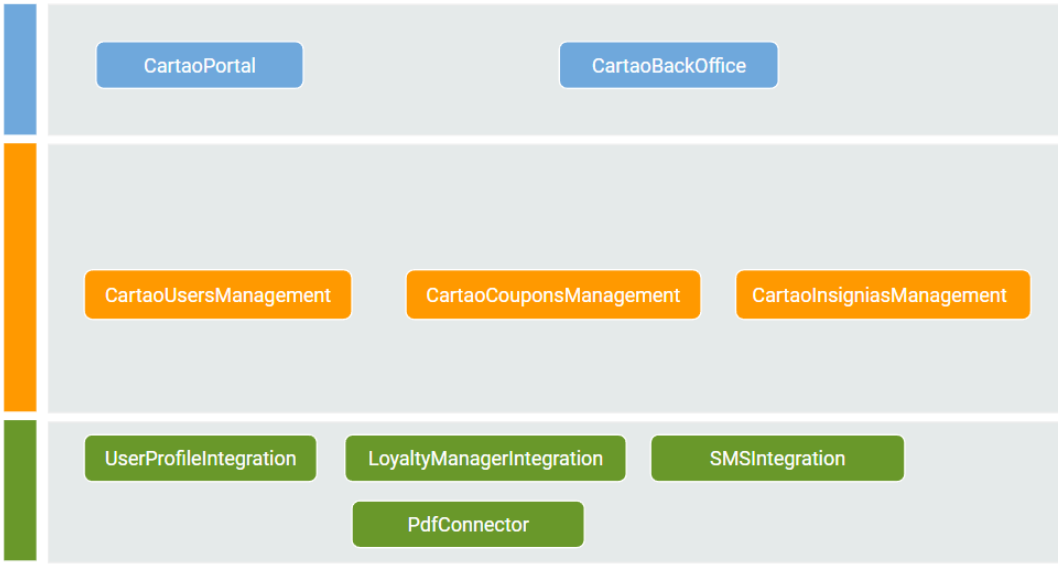

Figura 5.14: Arquitetura de módulos - Mendix

A azul estão os módulos da camada de UI do sistema. Cada módulo representa uma aplicação final para utilizador. Em Mendix, as páginas podem ser partilhadas entre módulos sem qualquer impacto no desempenho (aspeto que atualmente não é possível nas *Reactive Web Apps* de OutSystems), pelo que foi considerada a criação de vários módulos para encapsular cada área funcional da interface. No entanto, optou-se por apenas um módulo porque esta segregação pode ser feita recorrendo a pastas dentro do mesmo módulo, atingindo o mesmo objetivo e simplificando a implementação.

Relativamente às aplicações que agregam os módulos, representadas na figura 5.15, o racional é o mesmo explanado na arquitetura específica de OutSystems (5.2.2).

No caso específico de Mendix, a implementação seria mais simples se os módulos *Core* estivessem na mesma aplicação que os módulos de UI, uma vez que o consumo seria direto e não seria necessária a criação de REST API's. No entanto, como os dados das tabelas de Utilizadores, Cupões e Insígnias, necessitam de ser acedidos tanto pelo Portal como pelo Backoffice, foi necessário agrupar estes módulos numa aplicação à parte.

5.3.3 Implementação

Nesta subsecção é descrita em detalhe a implementação dos dois casos de uso escolhidos, recorrendo à plataforma Mendix. A implementação tem por base a descrição dos requisitos funcionais e não funcionais apresentados em 3, o desenho da solução em 4, a arquitetura específica apresentada em 5.3.2 e os pressupostos levantados na análise à solução atual do Portal em 5.2.3. Todos os desenvolvimentos foram realizados no Mendix Studio Pro.

Como ponto de partida, foram implementados *microflows* para importar os dados das tabelas de OutSystems para a base de dados de Mendix, de forma a garantir a consistência entre os dois sistemas. As figuras 5.16 e 5.17 ilustram o processo para obter entidades a partir de um serviço externo e persisti-las em Mendix. Na primeira figura é apresentado um *microflow* com apenas uma atividade (pedido REST), assim como as propriedades deste pedido onde é aplicado um 'Import Mapping'. Na segunda figura, é apresentado o 'Import Mapping' criado para mapear uma estrutura JSON, retornada pelo serviço REST, para uma entidade interna

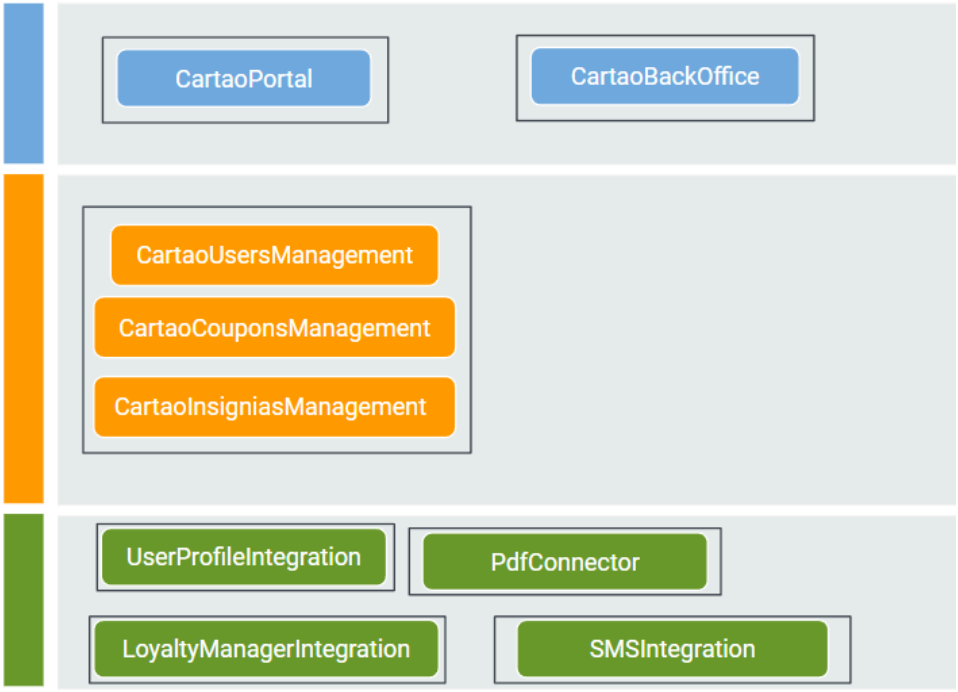

Figura 5.15: Arquitetura de aplicações - Mendix

de Mendix, definida no modelo de domínio do módulo. A inserção das linhas na tabela de base de dados é automática e gerida pela plataforma.

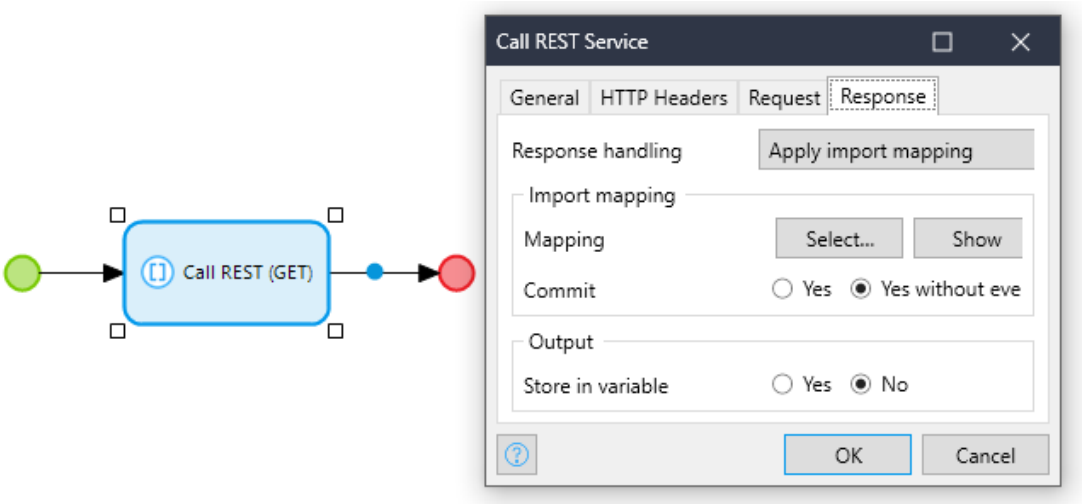

Figura 5.16: Pedido REST ao serviço de insígnias

Visualizar cupões

Depois de atingir paridade de dados nas Insígnias e Descontos, implementou-se a mesma lógica apresentada em 5.6 num *microflow* ligado à página dos cupões, de modo a pré-calcular todas as informações necessárias para a apresentação da interface (figura 5.18).

Em relação à interface gráfica, esta foi desenhada com ajuda dos dois modos de edição do IDE: o modo de design e o modo de estrutura. O modo de design é similar ao oferecido

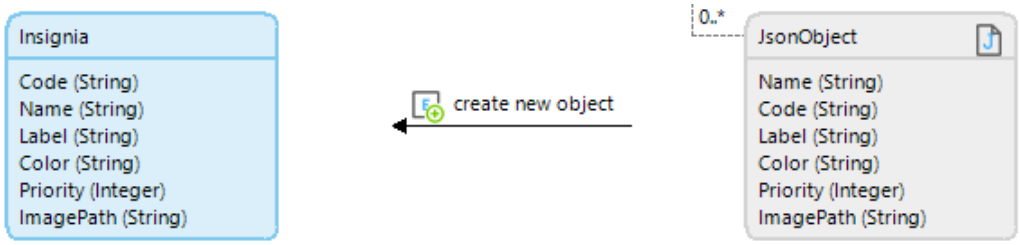

Figura 5.17: Insígnias Import Mapping

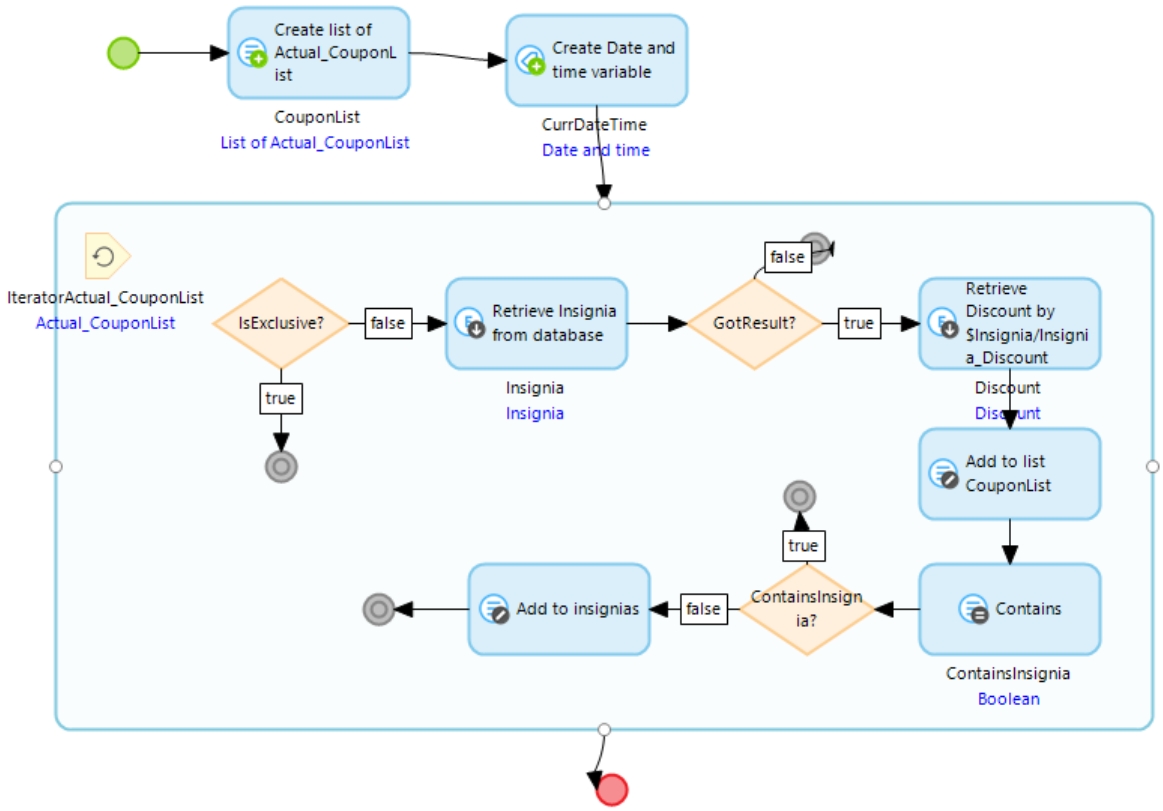

Figura 5.18: Excerto da nova implementação do microflow que retorna a listagem de cupões e insígnias

por OutSystems, baseando-se num editor WYSIWYG. O modo de estrutura, por outro lado, divide a interface em blocos lógicos, de forma a facilitar a visualização de cada *container* e as suas relações (figura 5.19).

O comportamento responsivo por defeito de Mendix é baseado em colunas, permitindo construir *layouts* que ocupam entre uma e doze colunas, que se adaptam ao tamanho do dispositivo do utilizador. Esta abordagem, ainda que funcional, apresenta alguns desafios na construção de interfaces completamente responsivas, no sentido em que não permite definir *breakpoints* de quebra de conteúdo. Para atingir o comportamento esperado foi necessário recorrer à construção manual de CSS.

À semelhança de OutSystems, Mendix também oferece alguns componentes base que aceleraram o desenvolvimento como, por exemplo, o padrão 'Gallery' que foi usado nas listagens de insígnias e cupões. Alguns componentes como o 'Separator' (separador de conteúdo) e

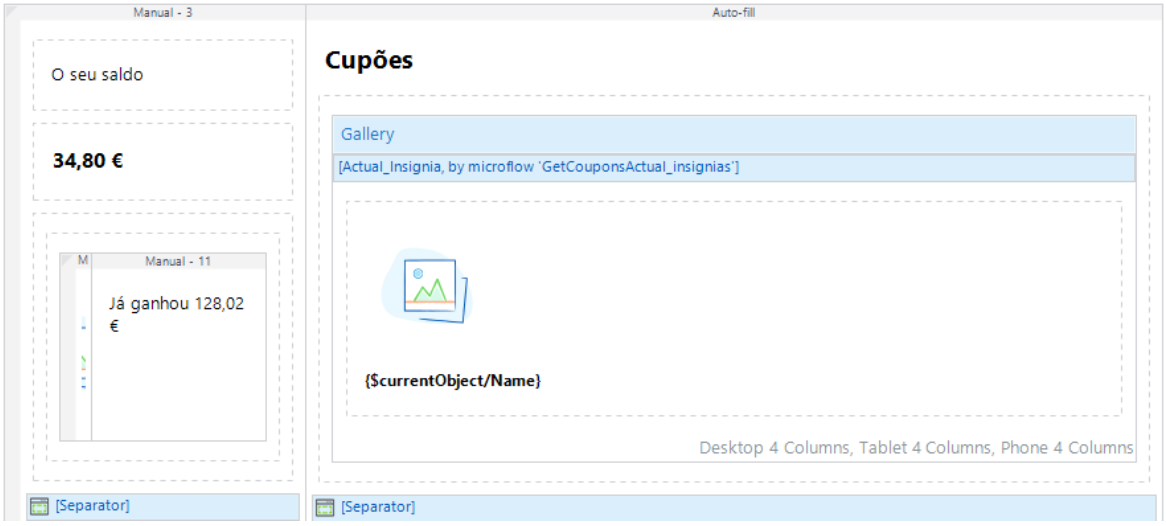

Figura 5.19: Modo de estrutura

o cupão foram construídos manualmente e isolados em 'Snippets' de forma a poderem ser reutilizados.

Na figura 5.20 é apresentado o resultado final desta página.

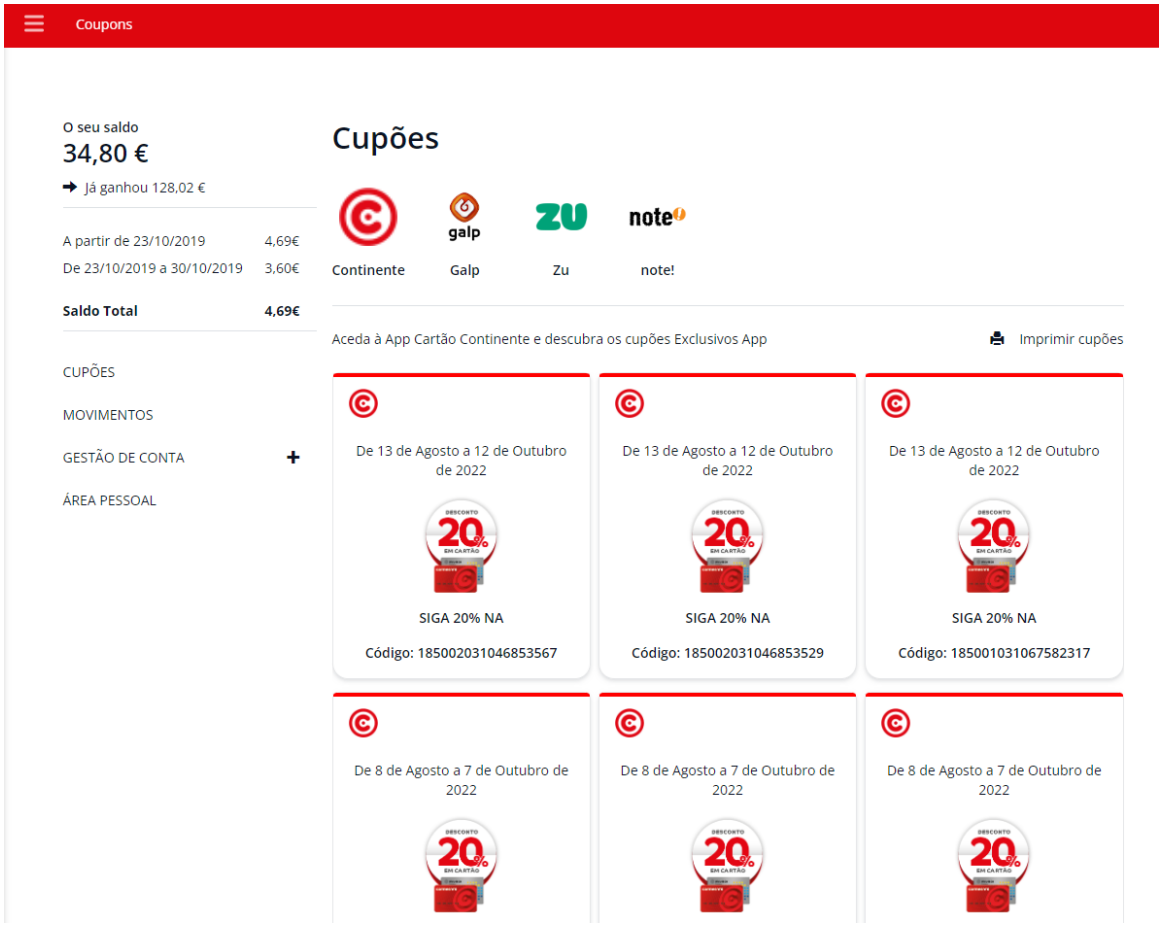

Figura 5.20: Página de listagem de cupões - vista Desktop

Configurar descontos

A implementação deste caso de uso foi suportada pelas capacidades de *scaffolding* da plataforma - possibilidade de gerar artefactos automaticamente, sem codificação manual.

Mendix, assim como OutSystems, dispõe de funcionalidades para gerar ecrãs com base numa entidade (tabela de base de dados). Com base nas colunas da tabela e no modo escolhido de apresentar a informação (listagem ou formulário, por exemplo), Mendix sugere uma implementação base que pode ser completamente customizada pelo programador.

A figura 5.21 apresenta o resultado desta operação de *scaffoling*, que permitiu gerar a listagem dos descontos presentes na base de dados. Esta operação resultou apenas na geração de itens simples, separados por linhas, com uma apresentação textual da informação contida em cada um, independentemente do tipo de dados de cada coluna. A paginação de registos é possível obter configurando as propriedades da listagem.

Apesar de um desconto estar intimamente ligado a uma insígnia, a plataforma não assumiu esta relação na listagem, deixando o ónus de implementação ao programador. Além disso, o título da página, o botão 'New Discount' e a lógica de redirecionamento ao clicar num item (de forma a ir para o seu detalhe) tiveram que ser implementadas manualmente. Algumas opções comuns neste tipo de ecrãs, como a pesquisa de registos e a ordenação de colunas também não foram considerados.

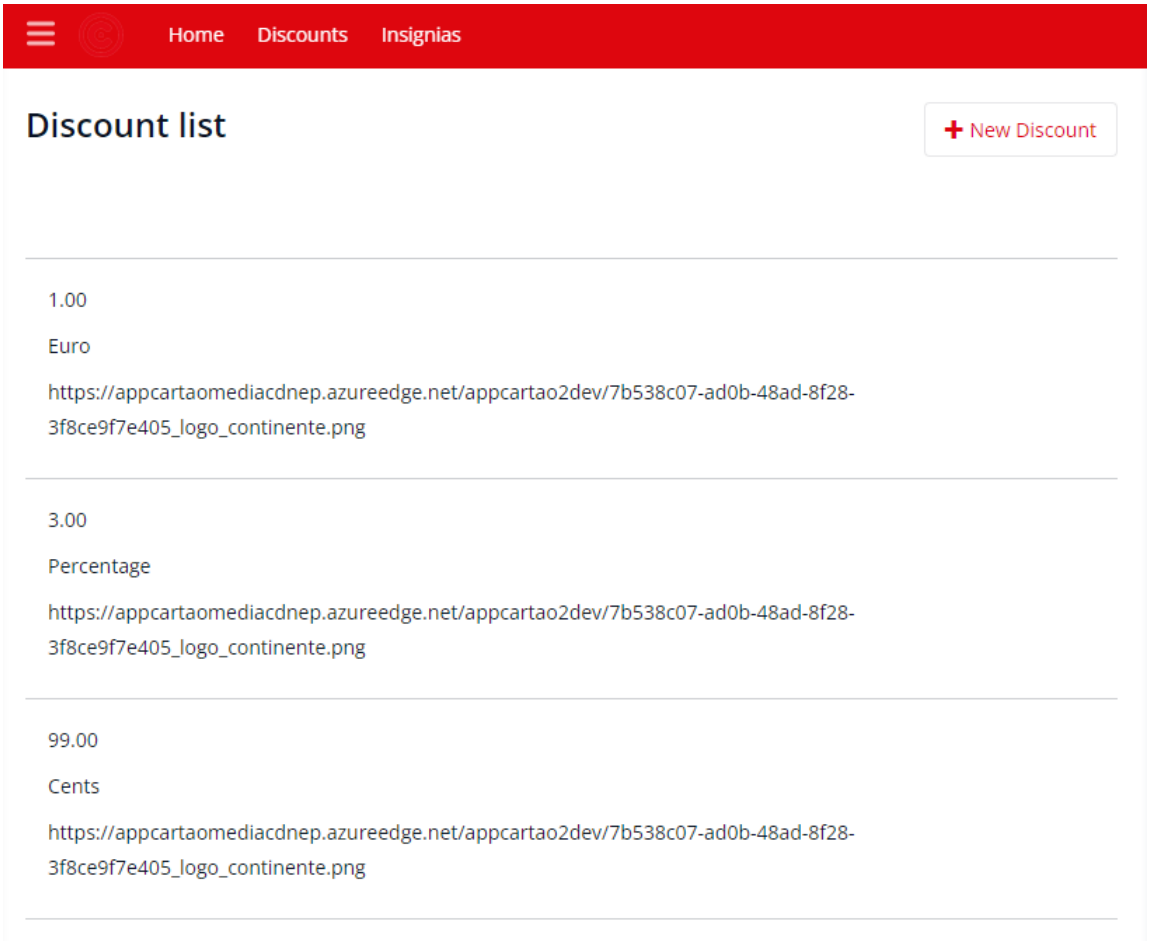

Figura 5.21: Página de listagem de descontos - Backoffice

Quando ao ecrã de detalhe (figura 5.22), o processo de *scaffolding* foi mais completo. A criação da página foi manual mas, com a adição do componente de formulário oferecido por Mendix, a plataforma foi capaz de criar um *input* adequado ao tipo de dados de cada coluna da entidade, criando também as validações adequadas. Neste caso, a plataforma já teve a capacidade de perceber a relação entre os conceitos desconto e insígnia e colocar um campo de opção múltipla de modo a criar esta associação na base de dados.

Um detalhe interessante foi o *scaffoling* optar pela criação de três *radio buttons* para a escolha do tipo de desconto, uma vez que estes descontos foram definidos como uma enumeração com poucos registos, o que resultou num experiência de utilizador mais amigável do que a geração automática de OutSystems.

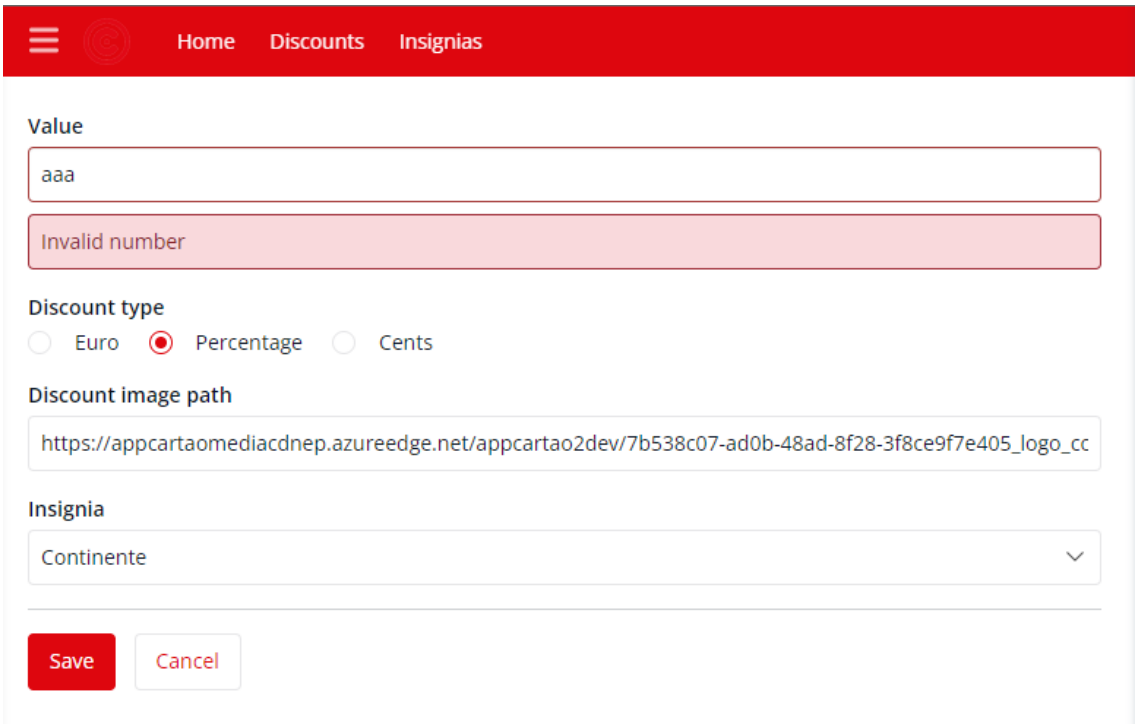

Figura 5.22: Página de detalhe de desconto - Backoffice

Quanto à lógica de negócio de validação dos dados inseridos, esta foi associada a cada campo e entidade de base de dados, através das suas propriedades. Em Mendix, opções para dar resposta a vários requisitos comuns (tamanho máximo permitido, visibilidade, permissões, condições de validação, entre outros) estão disponíveis nas propriedades dos elementos, permitindo que grande parte das validações sejam feitas por configuração. Esta abordagem é principalmente benéfica a programadores menos experientes, uma vez que simplifica o processo, mas pode trazer dificuldades na expressividade do código e na extensão do mesmo.

Na figura 5.23 é mostrada a configuração para o campo decimal 'Value' de um desconto. Este deve respeitar o comprimento máximo definido no atributo na base de dados, ter uma precisão decimal de duas casas, somente ser editável para utilizadores autorizados, não ser vazio e cumprir outras regras de negócio como o facto de uma percentagem de desconto não poder ser superior a cem porcento (comportamento especificado no campo 'Expression' da secção de validação).

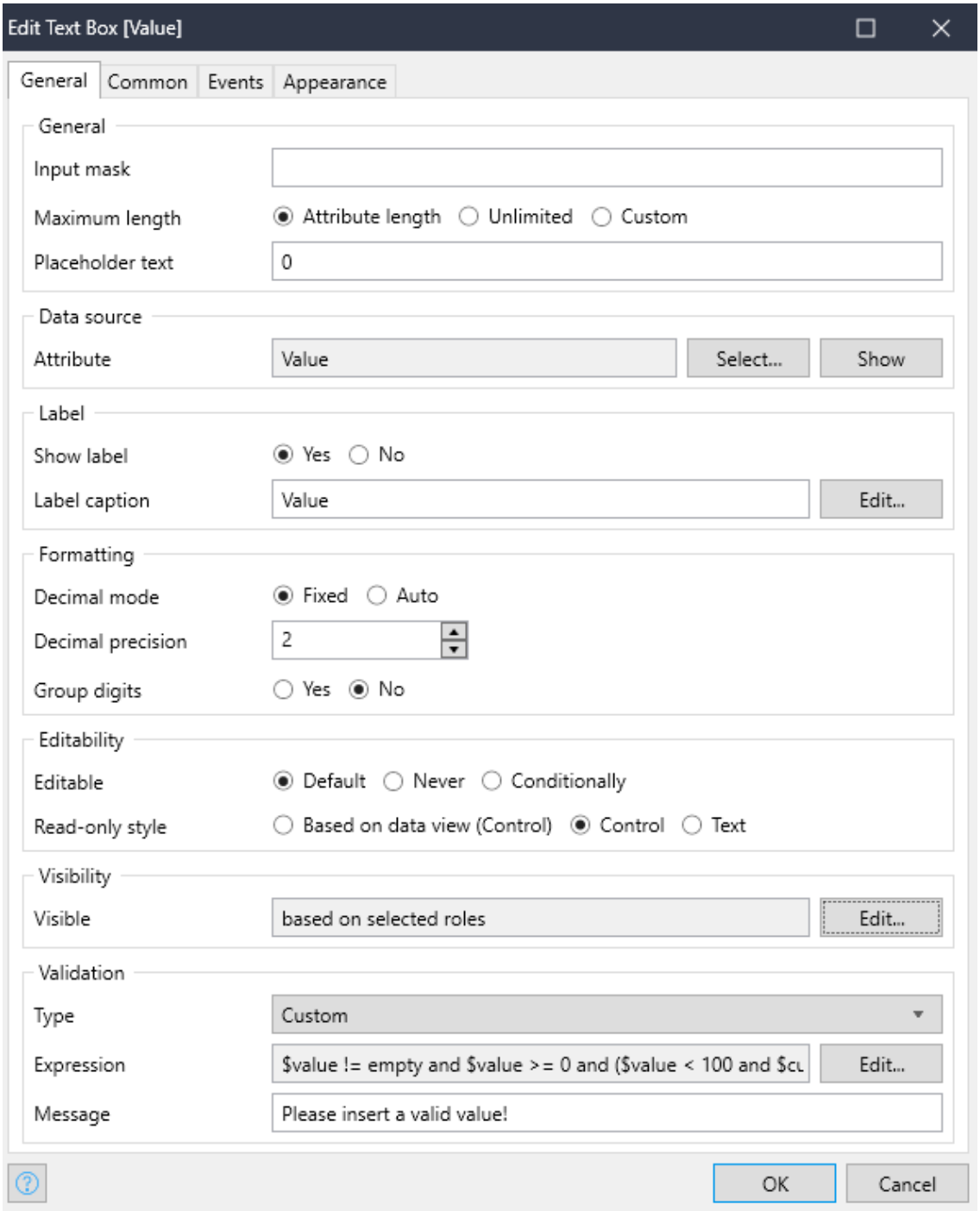

Figura 5.23: Detalhe da configuração de um campo

5.3.4 Limitações

A principal limitação encontrada no desenvolvimento da solução em Mendix foi a falta de profundidade da documentação. Ainda que abrangente, a documentação oficial aborda os tópicos de uma forma vaga, com poucos exemplos práticos (e muito simplistas), o que influenciou negativamente o processo de aprendizagem da plataforma e, por conseguinte, a capacidade de implementação. Algumas falhas na qualidade técnicas da solução podem ser derivadas da falta de conhecimento de uma melhor alternativa de desenho e implementação.

No próprio processo de desenvolvimento, especialmente um modular, foi sentida a necessidade da plataforma abstrair a comunicação entre diferentes aplicações. O facto de ter que expor e consumir manualmente API's REST foi uma tarefa morosa e propensa a erros que noutras ofertas como OutSystems está altamente simplificada.

A abordagem mais amigável ao programador inexperiente que Mendix assume, ao abstrair muita lógica para propriedades e configurações pode, a médio e longo prazo, trazer grandes desafios na manutenção e extensibilidade do código, na medida em que o código não é tão explícito como se fosse todo desenvolvido num fluxograma.

Os padrões de UI e os métodos de *scaffolding* oferecidos por defeito não responderam a vários casos de uso necessários para esta implementação, o que obrigou à procura de alternativas manuais ou disponibilizadas pela comunidade, o que também atrasou significativamente o desenvolvimento.

As publicações das aplicações foram extremamente lentas - mais de cinco minutos em média. Este problema foi mitigado pela solução de 'hot reload', que permite ver as alterações de UI rapidamente sem a necessidade de uma publicação completa. No entanto, ao trabalhar em várias aplicações simultaneamente, este comportamento não é possível.

Na parte de integração com sistemas externos foi sentida, como em OutSystems, a necessidade da plataforma ajudar no tratamento de erros, especialmente relacionados com a indisponibilidade do serviço. Políticas de *retry* têm que ser implementadas manualmente pelo programador e não existe forma automática de aplicar a mesma política a diferentes *endpoints*.

Capítulo 6

Avaliação da solução

Neste capítulo é descrito o processo de avaliação das soluções implementadas nas diferentes tecnologias. Este processo segue uma metodologia que parte de algumas pré-condições para garantir ao máximo um ponto de partida similar entre as várias implementações, de forma a levantar dados precisos e relevantes.

De acordo com os objetivos definidos para a avaliação da solução, serão comparadas as três implementações principais: a versão legada do Portal em *Traditional Web OutSystems*, a versão nova do Portal em *Reactive Web OutSystems* e a versão nova implementada numa SPA Mendix. Adicionalmente, o algoritmo utilizado na versão legada do Portal foi portada para a versão nova, com o intuito de a avaliar de forma independente o algoritmo e a tecnologia.

De notar que não foi possível obter informações concretas sobre a infraestrutura em que a aplicação desenvolvida em Mendix corre, pelo que é um fator que não está contemplado nesta análise. Considerando que foi usada uma licença empresarial de OutSystems e licença gratuita de Mendix, é provável que o servidor e base de dados de OutSystems tenham maior desempenho. É expectável, portanto, que em algumas métricas como o tempo de processamento dos cupões (que depende principalmente da capacidade do servidor e base de dados) sejam influenciadas. Ainda assim, considerou-se que não compromete os resultados ao nível de desqualificar a análise, uma vez que existem várias métricas que não dependem deste fator.

6.1 Metodologia

As soluções serão avaliadas individualmente segundo vários critérios, com uma componente de comparação de resultados no final.

Numa primeira análise será tido em conta os critérios de qualidade das próprias plataformas. Tanto OutSystems como Mendix têm no seu Integrated Development Environment (IDE) um mecanismo de avaliação de código produzido, relatando erros e avisos. Uma vez que os erros impedem os módulos de serem publicados, nenhuma das soluções tem erros. Os avisos, por outro lado, não impedem a publicação das soluções e estão relacionados com possíveis problemas de desempenho, segurança, coerência, entre outros.

Depois, será feita uma análise automática recorrendo à ferramenta *Lighthouse* - ferramenta de código aberto criada pela Google e incluída em *browsers* como o Google Chrome e Microsoft Edge, que permite a geração de relatórios automáticos com o foco em auditorias de desempenho, acessibilidade, Search Engine Optimization (SEO) e melhores práticas (Google 2020).

Os principais indicadores do *Lighthouse* são (Google 2020):

- First Contentful Paint (FCP) - tempo até o primeiro conteúdo relevante é mostrado no ecrã (texto ou imagens)
- Time To Interactive (TTI) - tempo que uma página demora até ficar interativa
- Speed Index - index que avalia quantitativamente a rapidez da página relativamente a outros *websites*
- Total Blocking Time (TBT) - soma de todos os tempos de períodos entre o FCP e TTI onde uma determinada tarefa demorou mais de 50ms a ser completada, podendo deixar a página não interativa
- Largest Contentful Paint (LCP) - o tempo até a maior porção de texto ou imagem é mostrada no ecrã
- Cumulative Layout Shift (CLS) - indicador que mede o reajuste de posicionamento de elementos visíveis (por exemplo uma imagem no meio de dois parágrafos que é carregada posteriormente e obriga a segunda secção de texto a ser empurrada para baixo)

Com base nas ferramentas de programador disponibilizadas pelos *browsers*, será feita uma última análise manual incidindo nos recursos carregados e na forma de carregamento dos mesmos. Esta exploração foca-se nos principais ficheiros necessários ao funcionamento das aplicações, nomeadamente ficheiros HTML, CSS e Javascript.

No final, e partindo das análises anteriores, serão feitas algumas sugestões de melhorias na ótica do desenvolvimento dentro das plataformas e na tradução desses desenvolvimentos em código funcional.

Em suma, a avaliação da solução terá as seguintes etapas:

- Análise de avisos
- Análise automática recorrendo a *Lighthouse*
- Análise manual (número e tamanho de pedidos, tempos, recursos)
- Levantamento de melhorias

Para garantir a fiabilidade dos resultados, algumas pré-condições foram estabelecidas, de modo a garantir o mesmo ponto de partida para as duas plataformas, removendo o máximo de variância possível entre testes (Stackify 2021). Neste sentido, as duas base de dados foram populadas com a mesma informação de forma a não afetar tempos de procura, a chamada ao serviço externo de cupões foi substituída por um resultado estático e os testes foram realizados à mesma hora.

Os testes que dependem do desempenho do servidor e base de dados serão realizados vinte vezes (dez vezes, dois dias). Considera-se para análise a média destes resultados.

Estes testes serão repetidos para diferentes números de cupões (cinco, vinte e quinhentos) de forma a simular diferentes situações de clientes. O caso extremo - quinhentos cupões - pretende sobretudo analisar as diferenças de desempenho do algoritmo, que podem ser mínimas num conjunto menor de dados.

6.2 Resultados

Nesta secção são apresentados os resultados obtidos ao aplicar a metodologia de avaliação proposta em 6.1.

6.2.1 OutSystems

Nesta subsecção é feita a análise relativa à plataforma OutSystems, tanto para o sistema legado do Portal (*Traditional Web*) como para a nova versão (*Reactive Web*). De notar que os avisos encontrados no sistema legado são referentes a todo o projeto e não só à implementação de alguns casos de uso como no caso da nova implementação.

Traditional Web

Em termos de arquitetura, o sistema apresenta dois avisos: módulo de UI monolítico e módulo *enduser* a expor serviços. No primeiro caso, este aviso é válido uma vez que grande parte da lógica de apresentação está inserida no mesmo módulo, quando poderia ser dividida em vários, por área funcional. Quanto ao segundo aviso, este foi considerado um falso positivo, uma vez que resulta de uma classificação errada do módulo 'CartaoContinente_Widgets.

Quanto à segurança, foram encontrados cinco avisos da mesma categoria - botões desabilitados mas visíveis. Estes botões podem ser habilitados do lado do cliente, permitindo fazer ações possivelmente não entendidas. Neste caso o recomendado é revalidar as condições de habilitação do botão no servidor, de modo a garantir que a segurança não está apenas do lado do cliente.

Na manutenção, foram encontrados mais de cento e cinquenta avisos, desde excertos de código longos sem documentação, falta de descrições em elementos públicos e código duplicado.

Quando ao desempenho, foram encontrados avisos relativos à grande dimensão de recursos (por exemplo imagens que podem ser otimizadas para redução de tamanho) e uso excessivo de recursos da base de dados (por exemplo não definir um máximo de linhas a retornar ou fazer várias *queries* quando podia ser feita só uma).

Na figura 6.1 são apresentados os resultados globais da avaliação por parte do *Lighthouse* da solução em *Traditional Web*.

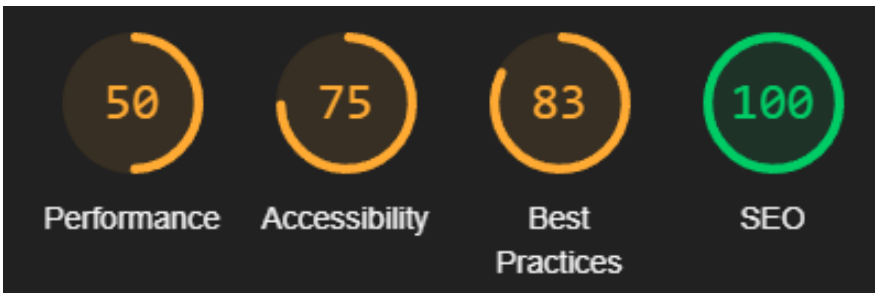

Figura 6.1: Pontuação Lighthouse - Traditional Web

O resultado mais importante para uma boa experiência de utilização, o relativo ao desempenho, está no limiar da média (50), o que corresponde à perceção dos utilizadores.

Relativamente aos indicadores de desempenho do *Lighthouse*, os resultados são apresentados na figura 6.2.

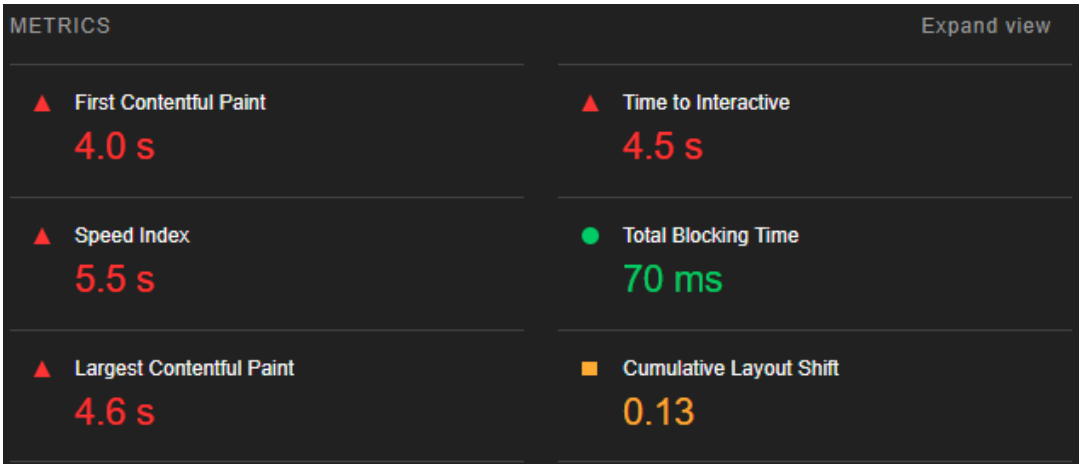

Figura 6.2: Indicadores Lighthouse - Traditional Web

Pela análise destes resultados (especialmente FCP, TTI e LCP), é possível concluir que o principal problema de desempenho está na quantidade de tempo que um utilizador tem que esperar até conseguir ver e interagir com a página.

No que toca ao algoritmo de processamento de cupões, os resultados são apresentados na tabela seguinte (tabela 6.1). A diferença entre os tempos de primeiro processamento e a média são praticamente negligíveis (considerando a ordem de grandeza), pelo que pode ser atribuída a algum mecanismo interno de otimização da plataforma ou base de dados, mas não ao algoritmo. Este resultado vai de encontro ao esperado, uma vez que nesta versão do algoritmo não existe *cache* entre as iterações.

Teste	**Estado**	**Duração (Milissegundos)**	**Tamanho (Kilobytes)**
5 cupões	Primeiro processamento	127 ms	47.3 KB
5 cupões	Média	103 ms	47.3 KB
20 cupões	Primeiro processamento	178 ms	54.8 KB
20 cupões	Média	153 ms	54.8 KB
500 cupões	Primeiro processamento	2302 ms	176 KB
500 cupões	Média	2153 ms	176 KB

Tabela 6.1: Resultados do algoritmo de processamento de cupões - Traditional Web

Pela análise das ferramentas de programador, esta página faz 101 pedidos (excluindo imagens) entre os quais 24 para ficheiros Javascript e 66 para ficheiros CSS. Foram encontrados vários ficheiros que poderiam ser carregados noutros momentos, nomeadamente ficheiros de

CSS que não são referentes a esta página. Existem, também, vários ficheiros de pequena dimensão que servem a mesma área da página que podiam ser agregados num só para reduzir o tráfego. Com o mesmo intuito, pode ser adotado um processo para minimizar o tamanho dos ficheiros.

Reactive Web

Quanto aos avisos gerados pela plataforma OutSystems relativos à nova implementação do Portal em Reactive Web, existem apenas seis avisos de segurança. Estes avisos estão relacionados com a existência de páginas e API's anónimas, que permitem ser acedidas por qualquer utilizador. Foram considerados falsos positivos, uma vez que fazem parte do fluxo de autenticação onde um cliente ainda não iniciou sessão. De qualquer forma, a partir do primeiro ecrã deste fluxo (onde o utilizador é obrigado a inserir o número de telemóvel e a validar um *token* recebido), qualquer ação até ao final deste processo está associada a uma identidade.

Na figura 6.3 são apresentados os resultados globais da avaliação por parte do *Lighthouse* da solução em Reactive Web. O valor de SEO poderia ser facilmente aumentado mas não foi tido em conta nesta implementação.

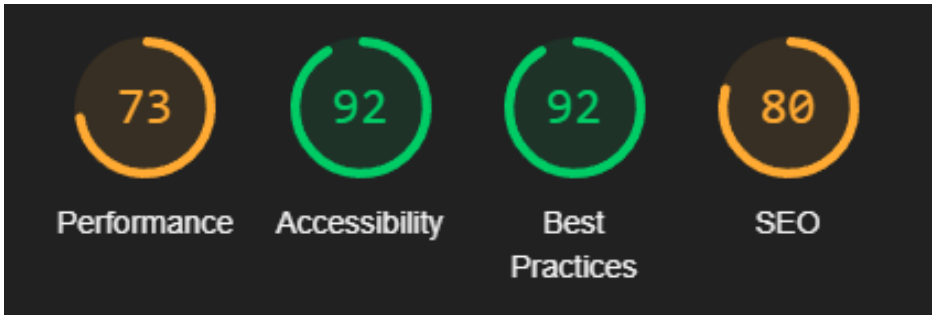

Figura 6.3: Pontuação Lighthouse - Reactive Web

Relativamente aos indicadores de desempenho do *Lighthouse*, os resultados são apresentados na figura 6.4.

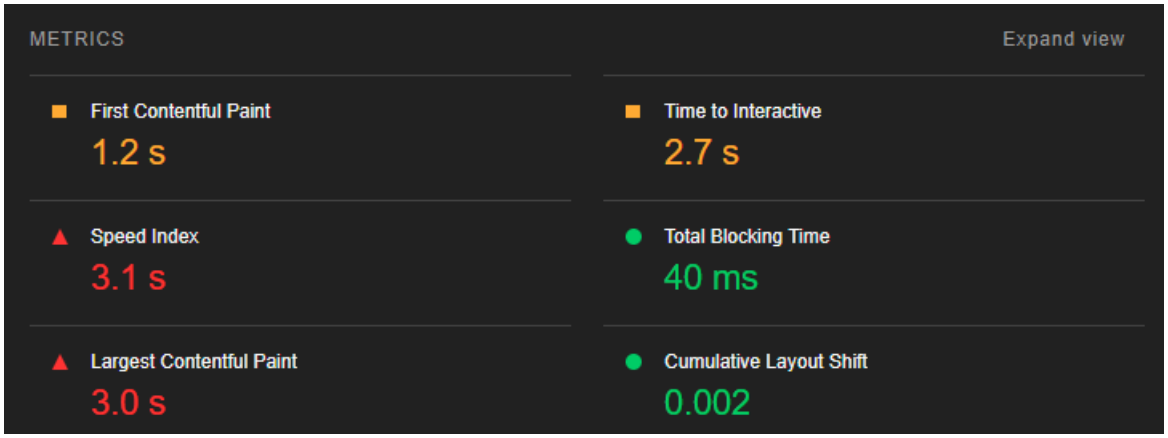

Figura 6.4: Indicadores Lighthouse - Reactive Web

Pela análise destes resultados é possível concluir que a implementação coaduna com o objetivo de melhorar o desempenho percecionado pelo utilizador, uma vez que o valor de

1.2 segundos de FCP é relativamente baixo, o que se traduz numa rápida apresentação de conteúdo visual (na maioria dos casos apenas o esqueleto do conteúdo) ao cliente.

O valor de 3.0 segundos de LCP é alto, o que afeta negativamente o *Speed Index*. Pela análise manual dos pedidos que a SPA faz, foram encontrados vários pedidos a ficheiros relacionados com o funcionamento da plataforma que atrasam o pedido aos cupões e, por conseguinte, a apresentação do conteúdo ao utilizador. Alguns destes ficheiros são: Basic.css (80 ms), OutSystemsManifestLoader.js (160 ms), OutSystems.js (300 ms), cordova.js (160 ms), Debugger.js (200 ms), OutSystemsReactWidgets.js (120 ms), OutSystemsUI.controller.js (287 ms) e OutSystemsUI.model.js (177 ms).

Uma vez que estes ficheiros são controlados pela plataforma, não existe maneira do programador melhorar este aspeto. De qualquer forma, deve ser notado que o carregamento destes ficheiros poderia ser muito otimizado por deixar de incluir o ficheiro 'cordova.js' que é apenas necessário para aplicações móveis nativas e reduzir o tamanho dos outros ficheiros que totaliza cerca de 3 MB.

O próprio modo de funcionamento da ferramenta *Lighthouse* exacerba o problema, uma vez que obriga a aplicação a fazer um *hard refresh*, carregando todos os recursos novamente. Numa utilização normal, todos os recursos acima mencionados seriam apenas carregados uma primeira vez (na primeira página que o utilizador visitasse), o que se traduziria numa melhoria de desempenho muito significativa.

Outro fator a considerar é o facto da avaliação ser feita num ambiente de desenvolvimento, com um servidor mais lento e sem políticas de otimização - notório, por exemplo, pela inclusão do ficheiro 'Debbuger.js'. Promover a aplicação a produção para efetuar testes não foi possível no âmbito desta investigação.

No que toca ao algoritmo de processamento de cupões, os resultados são apresentados na tabela seguinte (tabela 6.2). A diferença entre os tempos de primeiro processamento e a média são notórios, uma vez que existem políticas de *cache*, o que tornam pedidos subsequentes mais céleres.

Teste	**Estado**	**Duração (Milissegundos)**	**Tamanho (Kilobytes)**
5 cupões	Primeiro processamento	99 ms	0.86 KB
5 cupões	Média	64 ms	0.86 KB
20 cupões	Primeiro processamento	147 ms	1.3 KB
20 cupões	Média	70 ms	1.3 KB
500 cupões	Primeiro processamento	262 ms	5.9 KB
500 cupões	Média	124 ms	5.9 KB

Tabela 6.2: Resultados do algoritmo de processamento de cupões - Reactive Web

Uma vez que com o aumento dos cupões a probabilidade das insígnias e descontos se repetirem é maior, a utilização de *cache* entre iterações permite que a evolução dos tempos não sejam diretamente proporcionais.

Os tamanhos de resposta são bastante reduzidos, o que se traduz numa melhoria de experiência de utilização, especialmente para dispositivos que dependam de uma rede mais fraca (por exemplo uma rede móvel).

Quanto ao número de pedidos total, a página de listagem dos cupões fez 48 (excluindo imagens), dos quais 35 são ficheiros Javascript e 7 são ficheiros CSS. A transferência destes ficheiros não foi otimizada nem em tamanho nem em quantidade, o que seria um ponto de melhoria para a plataforma OutSystems.

6.2.2 Mendix

O Service Studio Pro, IDE utilizado para o desenvolvimento, apresenta 13 avisos de duas naturezas. Existem 7 avisos de segurança relativos a *microflows* não utilizados mas que podem ser acedidos publicamente. Estes *microflows* foram gerados automaticamente aquando da criação da aplicação e estão relacionados com funcionalidades Single Sign-On (SSO) de Mendix que não foram utilizadas. Os restantes 6 avisos são relativos a configurações por fazer, que no âmbito desta implementação também não foram necessárias.

Na figura 6.5 são apresentados os resultados globais da avaliação por parte do *Lighthouse* da solução desenvolvida em Mendix. Acessibilidade e SEO não foram focos desta implementação, pelo que o resultado é atribuído exclusivamente às otimizações inerentes à plataforma.

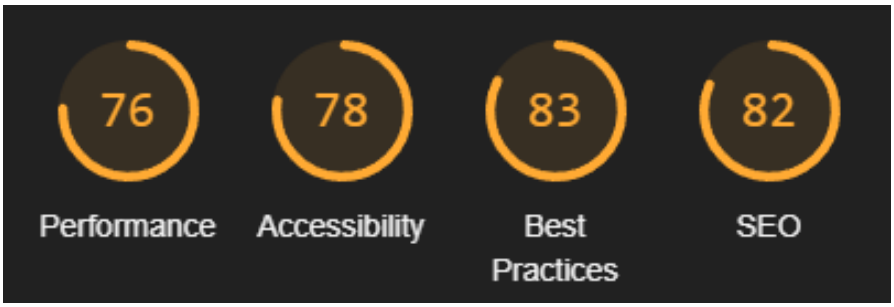

Figura 6.5: Pontuação Lighthouse - Mendix SPA

Relativamente aos indicadores de desempenho do *Lighthouse*, os resultados são apresentados na figura 6.6.

Pelos resultados é possível perceber que o utilizador terá uma boa perceção de desempenho, uma vez que algum conteúdo é apresentado rapidamente (FCP) e a página fica interativa em menos de dois segundos (TTI). Mais uma vez, assim como em Reactive Web, estes resultados representam o pior caso - caso em que todos os recursos da aplicação são carregados no momento de acesso a esta página. Numa utilização normal da aplicação estas métricas não são totalmente representativas da experiência do utilizador.

Mesmo num *hard refresh*, a aplicação faz apenas 15 pedidos (ignorando imagens). Isto deve-se a uma maior concentração de código em alguns ficheiros, nomeadamente o 'theme.compiled.css' e o 'mxui.js'. Estes dois pedidos, ainda que conceptualmente independentes, são feitos em sequência e não em paralelo, o que representa a maior perda de desempenho encontrada,

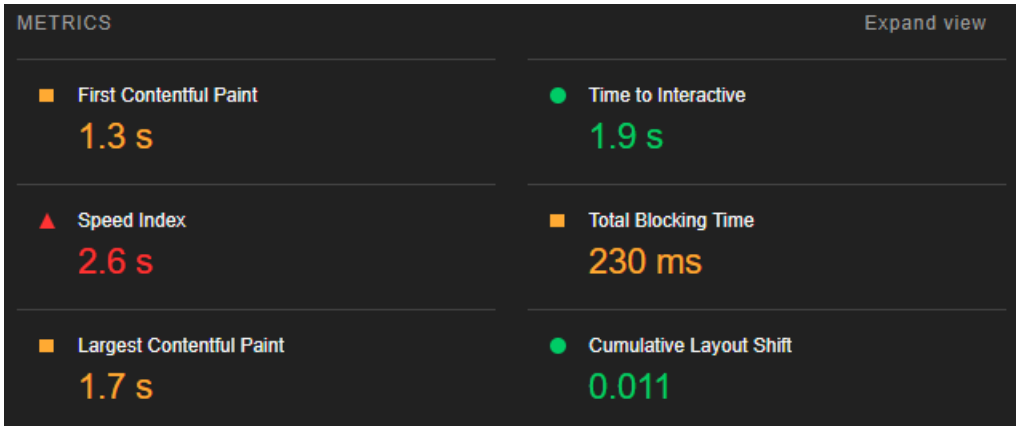

Figura 6.6: Indicadores Lighthouse - Mendix SPA

uma vez que são também os ficheiros de maior tamanho. A paralelização destes dois pedidos podia afetar positivamente todas as métricas, excluindo o CLS, em até 200 ms (tempo médio do pedido mais rápido).

Teste	Estado	Duração (Milissegundos)	Tamanho (Kilobytes)
5 cupões	Primeiro processamento	92 ms	1.6 KB
5 cupões	Média	78 ms	1.6 KB
20 cupões	Primeiro processamento	203 ms	2.9 KB
20 cupões	Média	81 ms	2.9 KB
500 cupões	Primeiro processamento	305 ms	40.0 KB
500 cupões	Média	133 ms	40.0 KB

Tabela 6.3: Resultados do algoritmo de processamento de cupões - Mendix SPA

Pode-se observar que a implementação de *cache* no algoritmo em Mendix tem uma relevância especialmente grande, uma vez que tanto no teste de 20 cupões como no de 500 cupões, o tempo médio de processamento é cerca de duas vezes mais rápido comparando com o primeiro processamento. O facto de se tratar de uma licença gratuita, com uma infraestrutura (servidor e base de dados) possivelmente mais fraca, pode explicar estes resultados, uma vez que o acesso à base de dados será mais lenta, pelo que o uso de *cache* tem um efeito mais significativo.

Quanto ao tamanho da resposta, verifica-se que esta cresce bastante no teste de 500 cupões, o que pode ser explicado pelos campos adicionados automaticamente pela plataforma. Em cada item da resposta, Mendix adiciona alguns campos como, por exemplo, os campos 'guid', 'hash' e 'objectType'. Todos os campos que representam um valor são encapsulados em objetos JSON, o que também, obrigatoriamente, aumenta o tamanho da resposta.

6.3 Comparação de resultados

Nesta secção são comparados os resultados anteriormente apresentados. São comparados tanto os resultados provenientes da ferramenta *lighthouse*, como os resultados relativos a algoritmo de processamento dos cupões (tabela 6.4).

Os valores são apresentados em segundos (s) para as métricas do *lighthouse*, em milissegundos (ms) para o tempo de processamento dos cupões e em kilobytes (KB) para o tamanho da resposta que contém a informação necessária para apresentar na interface gráfica.

Além de uma coluna relativa a cada implementação, existe também uma coluna intitulada 'Diferença' que corresponde ao delta entre o melhor resultado, apresentado a verde, e o pior, apresentado a vermelho.

	Traditional	**Reactive**	**Mendix**	**Diferença**
Time to Interactive	4.5 s	2.7 s	1.9 s	-2.6 s
First Contentful Paint	4.0 s	1.2 s	1.3 s	-2.8 s
Largest Contentful Paint	4.6 s	3.0 s	1.7 s	-2.9 s
Speed Index	5.5 s	3.1 s	2.6 s	-2.9 s
5 cupões				
Duração	103 ms	64 ms	78 ms	-39 ms
Tamanho	47.3 KB	0.86 KB	1.6 KB	-46.44 KB
20 cupões				
Duração	153 ms	70 ms	81 ms	-83 ms
Tamanho	54.8 KB	1.3 KB	2.9 KB	-53.5 KB
500 cupões				
Duração	2153 ms	124 ms	133 ms	-2029 ms
Tamanho	176 KB	5.9 KB	40.0 KB	-170.1 KB

Tabela 6.4: Tabela de comparação de resultados entre implementações

Pela comparação dos resultados observa-se que a implementação original (em OutSystems Traditional Web) pontua pior em todas as métricas. Isto deve-se a principalmente dois fatores: assentar numa tecnologia obsoleta que já não responde às necessidade atuais e ter um algoritmo menos eficiente.

Nas métricas de *lighthouse*, a implementação em Mendix fica à frente, conseguindo mais do dobro do desempenho em todos os testes face à implementação original.

Além de usar numa tecnologia mais moderna, Mendix tem um modo de publicação otimizado que, ao contrário do modo de *debug*, envia apenas os recursos estritamente necessários ao funcionamento da aplicação, o que na prática se traduz em menos HTML, CSS e Javascript para o *browser* processar, o que resulta numa melhoria substancial do desempenho. Em contraste, no ambiente de desenvolvimento de OutSystems não é possível fazer uma publicação otimizada em termos de recursos, pelo que os resultados sofreram por esta limitação. Apenas não incluir os ficheiros 'Debugger.js' e 'cordova.js' resultaria numa melhoria de 360 ms, pelo que é esperado que estas duas soluções convergissem em termos de desempenho se a solução de OutSystems fosse compilada no modo de produção.

Como foi explicado anteriormente em 6.2.1, a ferramenta *Lighthouse* faz a avaliação das páginas num cenário de *hard refresh*, o que obriga ao carregamento completo de todos os recursos. Numa utilização normal, a diferença das métricas de TTI e LCP não seriam tão grandes, o que foi possível perceber de forma empírica.

Quanto ao desempenho do algoritmo, este foi sempre superior em OutSystems tanto em termos de duração de processamento como no tamanho da resposta. No que diz respeito ao tempo de processamento, as diferenças entre OutSystems e Mendix são negligíveis, tendo um máximo de 14 ms entre resultados. Considerando que o tempo de resposta apresentado é composto pelo tempo de processamento do servidor mais o tempo de *download* do *payload*, a diferença de tamanhos das respostas poderia também influenciar este resultado, uma vez que um *payload* maior exige, naturalmente, um tempo de *download* maior. No entanto, mesmo no teste de 500 cupões, onde a diferença entre tamanhos de resposta é maior, não se notam grandes divergências, possivelmente por se tratar de uma unidade relativamente pequena (Kilobyte).

A diferença entre a qualidade do algoritmo é notória em todos os casos de teste, alcançando uma redução de mais de 50% num caso de teste correspondente a uma utilização normal (cliente com 20 cupões). Como era esperado, a diferença vai aumentando com a quantidade de cupões, uma vez que o mecanismo de otimização (cache) é invocado mais vezes.

Em relação aos tamanhos das respostas, Traditional Web apresenta um *payload* cerca de 30 vezes maior do que a nova solução em Reactive Web. Isto deve-se à inclusão de todos os campos (mesmo os não necessários) na resposta. No que toca à diferença entre Reactive Web e Mendix, esta não era expectável, uma vez que a estrutura de resposta é a mesma. No entanto, isto é explicado pela descoberta apresentada em 6.2.2 relativa aos campos extra adicionados automaticamente pela plataforma, aumentando o tamanho da resposta.

Ao cruzar os resultados do algoritmo com os resultados das métricas do *lighthouse*, é possível perceber que a otimização do algoritmo não é o principal aspeto para melhoria, uma vez que Reactive Web tem os melhores resultados no algoritmo mas fica consideravelmente atrás de Mendix nas outras métricas. Conclui-se, portanto, que a otimização dos recursos HTML, CSS e Javascript enviados para o *browser* são de maior relevância para estes resultados.

Capítulo 7

Conclusão

Neste trabalho foi realizado um estudo sobre o desenvolvimento rápido de aplicações, começando pelas metodologias normalmente associadas a este tema, como RAD e Scrum. Foi introduzido o universo das ferramentas no-code e low-code como tecnologias que podem auxiliar as empresas no desenvolvimento rápido, apresentando as principais plataformas low-code hoje presentes no mercado. Destas plataformas, foram escolhidas a OutSystems e Mendix para realizar um projeto de exploração sobre as mesmas, com o intuito de perceber a sua aplicabilidade a um problema real, relacionado com o Portal Cartão Continente atual. Para isso, foram definidos requisitos que detalham as funcionalidades escolhidas para a prova de conceito a desenvolver. Foi, também, definida uma arquitetura independente da plataforma, não tendo em conta as características específicas de cada ferramenta utilizada para o desenvolvimento, no sentido de perceber como estas davam resposta. De seguida, foi descrita a implementação em cada uma das plataformas, baseando-se nos principais conceitos relacionados com o desenvolvimento nas mesmas. Por fim, foi feita uma avaliação das soluções implementadas de modo a apurar a sua qualidade, com base em métricas obtidas pela própria plataforma e por ferramentas externas.

O principal objetivo prendia-se com o desenvolvimento de uma prova de conceito de uma nova versão do Portal Cartão Continente usando Reactive Web Apps de OutSystems. Este objetivo foi alcançado com sucesso, uma vez que a solução desenvolvida permite aferir as vantagens deste novo modo de desenvolvimento oferecido por OutSystems. A nova aplicação tem um desempenho bastante superior a todos os níveis, especialmente nos diretamente relacionados com a experiência do utilizador, como analisado em 6.3.

Numa vertente de investigação foram também atingidos os objetivos de comparar a plataforma OutSystems com outras plataformas low-code e comparar as Traditional e Reactive Web Apps de OutSystems. Para o primeiro objetivo, a secção 2.4 apresenta uma comparação teórica entre OutSystems, Mendix e PowerApps (as principais do mercado). No capítulo de implementação (5) e no capítulo de análise (6), esta comparação é aprofundada entre OutSystems e Mendix, baseada em fatores práticos. Quanto ao segundo objetivo, a comparação entre Traditional e Reactive Web Apps é feita principalmente a nível de resultados, explicando com pouca profundidade as suas diferenças entre as duas tecnologias. Este aspeto poderia ser melhorado através de uma análise teórica entre as divergências destes dois métodos de construir aplicações em OutSystems.

O último objetivo estava relacionado com a análise e levantamento de melhorias das soluções desenvolvidas. Foi dedicado o capítulo 6 a este objetivo avaliando, segundo uma metodologia bem definida, os resultados das soluções atingidas, comparando-as e sugerindo melhorias. Considera-se, portanto, que este objetivo foi cumprido.

As principais limitações encontradas na realização deste trabalho estão relacionadas com a falta de informação sobre as plataformas low-code estudadas. Apesar do crescimento rápido, esta área ainda é relativamente recente e pouco madura, pelo que grande parte da informação é disponibilizada pelas próprias empresas que desenvolvem a plataforma, o que afetou negativamente a diversidade de fontes pretendida. Mesmo a documentação disponibilizada nos websites oficiais é muitas vezes precária, o que dificultou a aprendizagem das plataformas. O facto da comunidade de programadores destas plataformas ser relativamente pequena também impactou negativamente o desenvolvimento, uma vez que o suporte é menor.

Como trabalho futuro sugere-se a avaliação das soluções geradas por OutSystems num ambiente produtivo, de forma a ter em conta as otimizações que a plataforma eventualmente faz a nível de recursos enviados para o browser, uma vez que foi o fator que mais afetou as métricas da solução em Reactive Web Apps.

Bibliografia

Alexander, Forsyth (jan. de 2021a). *Low-code and no-code: What's the difference and when to use what?* url: https://www.outsystems.com/blog/posts/low-code-vs-no-code/.

– (fev. de 2021b). *What can you build with low-code?* url: https://www.outsystems.com/blog/posts/what-can-you-build-with-low-code/.

Alves, Ricardo (out. de 2019). *Reactive web applications - the next generation of web apps*. url: https://www.outsystems.com/blog/posts/reactive-web-applications/.

Ardalis (2022). *Strategies for migrating ASP.NET Web Forms Apps*. url: https://docs.microsoft.com/en-us/dotnet/architecture/porting-existing-aspnet-apps/migrate-web-forms.

Arede, Marco (mar. de 2021). *Architecture naming conventions-for modules in OutSystems*. url: https://marcoarede.medium.com/architecture-naming-conventions-for-modules-in-outsystems-8e2d6ce62f1d.

Beynon-Davies, Paul et al. (set. de 1999). «Rapid application development (RAD): An empirical review». Em: *European Journal of Information Systems* 8. doi: 10.1057/palgrave.ejis.3000325.

BigCommerce (fev. de 2022). *Online shopping statistics*. url: https://www.bigcommerce.com/blog/online-shopping-statistics/#ecommerce-is-growing-every-day.

Caspio (out. de 2021). *The rapid adoption of low-code development platforms*. url: https://blog.caspio.com/rapid-adoption-low-code-platforms/.

Clack, Marilyn B. (abr. de 2020). *Why eCommerce is so important for your business*. url: https://www.digitalmarketingcommunity.com/articles/importance-of-ecommerce/.

coMakeIt (abr. de 2020). url: https://www.comakeit.com/blog/guide-to-evaluate-low-code-platforms/.

Despa, Mihai Liviu (2014). «Comparative study on software development methodologies». Em: *Database Systems Journal* 3, pp. 37–56. issn: 2069-3230.

Ferreira, Luís Filipe Rocha Maia (jan. de 2009). *FV-Rad : A practical framework for rapid application development*. url: https://hdl.handle.net/10216/66702.

Google (2020). *Lighthouse overview*. url: https://developer.chrome.com/docs/lighthouse/overview/.

Hosking, Ben (out. de 2021). *Developers have seen low-code development before and it failed, why it will be different this time*. url: https://medium.com/geekculture/developers-have-seen-low-code-development-before-and-it-failed-why-it-will-be-different-this-time-32d8d2dbdf43.

Jednaszewski, Marek (fev. de 2022). *Low-code vs. no-code: Differences, Similarities and Use Cases*. url: https://www.mendix.com/blog/understand-no-code-vs-low-code-development-tools/#no-code.

Lamsweerde, Axel van (2000). «Requirements Engineering in the Year 00: A Research Perspective». Em: *Proceedings of the 22nd International Conference on Software Engineering*. ICSE '00. Limerick, Ireland: Association for Computing Machinery, pp. 5–19. isbn:

1581132069. doi: 10.1145/337180.337184. url: https://doi.org/10.1145/337180.337184.

LucidChart (ago. de 2018). *4 phases of Rapid Application Development Methodology*. url: https://www.lucidchart.com/blog/rapid-application-development-methodology.

Malan, Ruth, Dana Bredemeyer et al. (2001). «Functional requirements and use cases». Em: *Bredemeyer Consulting*.

Martin, James (1991). *Rapid Application Development*. USA: Macmillan Publishing Co., Inc. isbn: 0023767758.

Mendix (2021a). url: https://docs.mendix.com/refguide/modules/.

– (out. de 2021b). *Building Hybrid Mobile Applications: Mendix evaluation guide*. url: https://www.mendix.com/evaluation-guide/app-capabilities/hybrid-mobile-apps/.
– (out. de 2021c). *Building progressive web applications in Mendix: Mendix Evaluation Guide*. url: https://www.mendix.com/evaluation-guide/app-capabilities/native-mobile-apps/.
– (nov. de 2021d). *Building responsive web applications: Mendix evaluation guide*. url: https://www.mendix.com/evaluation-guide/app-capabilities/web-apps/.
– (abr. de 2021e). *General info - studio 9 guide*. url: https://docs.mendix.com/studio/general.
– (nov. de 2021f). *Mendix component reuse*. url: https://www.mendix.com/evaluation-guide/app-lifecycle/reuse/.
– (ago. de 2022a). url: https://docs.mendix.com/refguide/consume-add-on-modules-and-solutions/.
– (set. de 2022b). *Implement mendix best practices for development*. url: https://docs.mendix.com/howto/general/dev-best-practices/.
– (fev. de 2022c). *Low-code application development platform evaluation guide*. url: https://www.mendix.com/evaluation-guide/.
– (mai. de 2022d). *Mendix Multi-channel apps*. url: https://www.mendix.com/evaluation-guide/app-capabilities/ux-multi-channel-apps/.
– (mai. de 2022e). *Mendix pricing*. url: https://www.mendix.com/pricing/.

Microsoft (2021a). *Overview of building a model-driven app with power apps - power apps*. url: https://docs.microsoft.com/en-us/powerapps/maker/model-driven-apps/model-driven-app-overview.
– (2021b). *Overview of building canvas apps - power apps*. url: https://docs.microsoft.com/en-us/powerapps/maker/canvas-apps/getting-started.
– (2021c). *What are power apps portals? - power apps*. url: https://docs.microsoft.com/en-us/powerapps/maker/portals/overview.

O'Shea, Dan (abr. de 2017). *Study: Shoppers more willing to consult mobile phones than associates while in stores*. url: https://www.retaildive.com/news/study-shoppers-more-willing-to-consult-mobile-phones-than-associates-while/439970.

OutSystems (2019). url: https://rb.gy/c7imbm.

– (2021a). *Architecture: Evaluation Guide*. url: https://www.outsystems.com/evaluation-guide/architecture/.
– (2021b). *OutSystems development and Management Tools*. url: https://www.outsystems.com/evaluation-guide/development-and-management-tools/.
– (2021c). *Translating business concepts into application modules*. url: https://success.outsystems.com/Documentation/Best_Practices/Architecture/Designing_the_Architecture_of_Your_OutSystems_Applications/Translating_business_concepts_into_application_modules.

– (2021d). *What is rapid application development*. url: https://www.outsystems.com/glossary/what-is-rapid-application-development/.
– (2022a). *Choose the right app for your project*. url: https://success.outsystems.com/Documentation/11/Getting_started/Choose_the_right_app_for_your_project.
– (2022b). *Exposing Functionality*. url: https://success.outsystems.com/Documentation/11/Developing_an_Application/Reuse_and_Refactor/Use_Services_to_Expose_Functionality#Exposing_Service_Actions.
– (2022c). *From architecture to development*. url: https://success.outsystems.com/Documentation/Best_Practices/Architecture/From_architecture_to_development.
– (2022d). *The Architecture Canvas*. url: https://success.outsystems.com/Documentation/Best_Practices/Architecture/Designing_the_Architecture_of_Your_OutSystems_Applications/The_Architecture_Canvas.
– (2022e). *Validating your application architecture*. url: https://success.outsystems.com/Documentation/Best_Practices/Architecture/Designing_the_Architecture_of_Your_OutSystems_Applications/Validating_your_application_architecture.

Patel, Unnati A. e Niky K. Jain (2013). «New Idea In Waterfall Model For Real Time Software Development». Em: *International journal of engineering research and technology* 2.

Rick-Anderson (2021). *ASP.NET web forms*. url: https://docs.microsoft.com/en-us/aspnet/web-forms/.

Roe, David (nov. de 2021). *How low-code and no-code apps fuel digital transformation*. url: https://www.reworked.co/information-management/whats-behind-the-explosion-of-low-code-and-no-code-applications/.

Royce, W. W. (1987). «Managing the Development of Large Software Systems: Concepts and Techniques». Em: *Proceedings of the 9th International Conference on Software Engineering*. ICSE '87. Monterey, California, USA: IEEE Computer Society Press, pp. 328–338. isbn: 0897912160.

Sahay, Apurvanand et al. (ago. de 2020). «Supporting the understanding and comparison of low-code development platforms». Em: doi: 10.1109/SEAA51224.2020.00036.

Sanchis, Raquel et al. (2020). «Low-Code as Enabler of Digital Transformation in Manufacturing Industry». Em: *Applied Sciences* 10.1. issn: 2076-3417. doi: 10.3390/app10010012. url: https://www.mdpi.com/2076-3417/10/1/12.

Setrag Khoshafian, Khosh Consulting (fev. de 2021). *No-code/low-code: Why you should be paying attention*. url: https://venturebeat.com/2021/02/14/no-code-low-code-why-you-should-be-paying-attention/.

SonaeMC (2021). *SonaeMC*. url: https://sonaemc.com/sobre-nos/.

Stackify, Stackify (abr. de 2021). *Performance testing types, steps, best practices, and metrics*. url: https://stackify.com/ultimate-guide-performance-testing-and-software-testing/.

Sutherland, Jeff e Ken Schwaber (2020). «The scrum guide». Em: *The definitive guide to scrum: The rules of the game. Scrum. org* 268, p. 19.

Veiga, Gonçalo (fev. de 2018). *The evolution of low-code*. url: https://www.outsystems.com/blog/posts/the-evolution-of-low-code_welcome-to-the-machine.

Vivek, Kumar (2021). *What is power apps? - power apps*. url: https://docs.microsoft.com/en-us/powerapps/powerapps-overview.

Waheed, Ayesha et al. (jan. de 2018). «A Review of Popular Agile Software Development Technologies». Em: *Journal of Information Technology Software Engineering* 08. doi: 10.4172/2165-7866.1000245.

Wong, Jason (2021). *Magic Quadrant for Enterprise Low-Code Application Platforms*. url: https://www.gartner.com/doc/reprints?id=1-27I04ZJT&ct=210921&st=sb.