INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Internet Banking - Desenvolvimento de aplicação em .NET

ANDREIA ISABEL CAMPOS REIS
Outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Internet Banking

Desenvolvimento de aplicação em .NET

Andreia Isabel Campos Reis

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

Orientador: Alberto Sampaio

Coorientadora: Isabel Sampaio

Júri:

Presidente:

Vogais:

Porto, fevereiro 2022

Resumo

Este documento descreve o trabalho de desenvolvimento de funcionalidades para o projeto *Installments*. *Installments* é um projeto de *Internet Banking* que tem como objetivo fornecer funcionalidades de pagamentos fracionados aos utilizadores das aplicações móveis do banco. A solução de *software* na qual se baseia este projeto é composta por quatro camadas. As camadas são a aplicação móvel, o serviço *Installments Experience*, o serviço *Installments Process*, e o Sistema Central.

O trabalho apresentado neste documento consiste no desenvolvimento de algumas funcionalidades nos microsserviços desenvolvidos em .NET *Installments* Experiece e *Installments Process*. As funcionalidades em causa são o desenvolvimento do novo caso de uso Pedir Contacto, e a realização de correções ao caso de uso Consultar Planos Correntes. Também devem ser feitos testes unitários a todos os casos de uso que integram os dois serviços mencionados. Para o design de todas as funcionalidades referidas, foi usada a metodologia iterativa *Attribute-Driven Design*, e o design foi dividido em duas iterações.

A primeira iteração consiste no desenho e implementação dos casos de uso acima referidos. Para o desenvolvimento da primeira iteração foi tido em conta que que deveria ser utilizada cache para reduzir a sobrecarga do sistema central. Redis foi o armazenamento de cache utilizado. Também foi tido em conta que as três camadas do lado *backend* aplicam a abordagem API-led Connectivity, que estabelece responsabilidades bem definidas para as suas APIs que devem ser respeitadas de modo a aumentar a agilidade no desenvolvimento.

A segunda iteração tem como objetivo aplicar os atributos de qualidade manutenibilidade e testabilidade definidos nos requisitos do projeto para todos os casos de uso. Para que o requisito de testabilidade fosse cumprido era necessário que o testes unitários tivessem uma determinada cobertura. E para que o requisito de manutenibilidade fosse cumprido era necessário diminuir os *code smells* até um determinado índice de dívida técnica. Tanto para monitorizar a cobertura como a testabilidade, foi utilizada a ferramenta SonarQube. E com a ajuda dessa ferramenta, os requisitos de testabilidade e manutenibilidade foram cumpridos em ambos os serviços. Com estes requisitos cumpridos, a aplicação pode passar aos testes de qualidade efetuados por outros *stakeholders* do projeto *Installments*.

Palavras-chave: *Internet Banking*, Atributo de qualidade, Estilo arquitetural, API-led Connectivity, *Attribute-Driven Design*, SonarQube

Abstract

This document describes feature development work for the *Installments* project. *Installments* is an *Internet Banking* project that aims to provide *Installments* payment features to users of the bank's mobile applications. The *software* solution on which this project is based is composed of four layers. The layers are the mobile application, the *Installments Experience* service, the *Installments Process* service, and the Central System.

The work presented in this document is the development of some functionalities in the microservices developed in .NET *Installments Experience* and *Installments Process*. The functionalities in question are the development of the new Request Contact use case, and corrections to the Consult Current Plans use case. Unit tests must also be carried out on all use cases that integrate the two services mentioned. For the design of all the functionalities, the iterative *Attribute-Driven Design* methodology was used, and the design was divided into two iterations.

The first iteration consists of designing and implementing the use cases mentioned above. For the development of the first iteration, it was considered that cache should be used to reduce the overhead of the central system. Redis was the cache store used. It was also considered that the three layers of the *backend* side apply the API-led Connectivity approach, which establishes well-defined responsibilities for their APIs that must be respected in order to increase development agility.

The second iteration aims to apply the maintainability and testability quality attributes defined in the project requirements to all use cases. For the testability requirement to be fulfilled, it was necessary that the unit tests had a certain coverage. And for the maintainability requirement to be fulfilled, it was necessary to reduce *code smells* to a certain technical debt index. To monitor both coverage and testability, the SonarQube tool was used. And with the help of this tool, testability and maintainability requirements were fulfilled in both services. With these requirements fulfilled, the application can pass the quality tests carried out by other *stakeholders* of the *Installments* project.

Keywords: *Internet Banking*, Quality Attribute, Architectural Style, API-led Connectivity, *Attribute-Driven Design*, SonarQube

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

ADD	*Attribute-Driven Design*
ATM	*Automated Teller Machine*
API	*Application Programing Interface*
CORBA	*Common Object Request Broker Architecture*
DTO	*Data Transfer Object*
EJB	*Enterprise JavaBeans*
J2EE	*Java 2 Platform, Enterprise Edition*
NCD	*New Concept Development*
NPD	*New Product Development*
REST	*Representational State Transfer*
SOA	*Service Oriented Architecture*
SQL	*Structured Query Language*
TIC	Tecnologias de Informação e Comunicação
URL	*Uniform Resource Locator*

Lista de Símbolos

λ	Valor próprio

1 Introdução

1.1 Contexto

O rápido desenvolvimento das tecnologias de informação e comunicação tiveram um grande impacto na sociedade e na economia mundial (Isac & Drigă, 2015). Desde os anos 90, o número de casas com acesso à Internet aumentou consideravelmente, oferecendo novas possibilidades de mercados para serviços baseados na Internet (Couto, Tiago, & Tiago, 2013). Atualmente, as estatísticas mostram um desenvolvimento rápido e exponencial do uso da Internet (Isac & Drigă, 2015). O notável aumento da utilização da Internet também se verificou em Portugal. Por exemplo, dados de (Roser, Ritchie, & Ortiz-Ospina, 2015) indicam que no ano 2000, 16,43% da população portuguesa usava Internet, e no ano de 2005, a percentagem aumentou para 34,99%.

O forte desenvolvimento das tecnologias de informação aliado a uma economia baseada no conhecimento contribuiu para grandes mudanças no setor bancário. Assim, se os serviços bancários pretenderem ter sucesso nas novas condições, estes devem reorganizar-se e diversificar-se de modo a trazerem satisfação aos clientes. Neste contexto, a *Internet Banking* é um dos serviços necessários e indispensáveis do sistema financeiro (Isac & Drigă, 2015).

A *Internet Banking* é definida como a utilização dos serviços bancários via Internet, por meio de um computador pessoal ou outros dispositivos com capacidade de acesso à Internet (Gkoutzinis, 2006) (Martins, Oliveira, & Popovič, 2014). Os serviços de *Internet Banking* superaram as restrições espaciais e temporais dos serviços bancários tradicionais, uma vez que oferecem uma cobertura de 24 horas por dia (Couto, Tiago, & Tiago, 2013). A *Internet Banking* oferece mais benefícios às instituições financeiras devido à maior acessibilidade dos serviços bancários (Martins, Oliveira, & Popovič, 2014). Através deste canal de distribuição, os clientes podem desempenhar uma grande variedade de transações eletrónicas (Martins, Oliveira, & Popovič, 2014). Essas transações são por exemplo emitir cheques, pagar contas, transferir fundos, imprimir extratos e consultar saldos de contas (Martins, Oliveira, & Popovič, 2014). A literatura sugere várias designações para identificar *Internet Banking* tais como *e-banking*, *electronic banking*, ou *online banking* (Martins, Oliveira, & Popovič, 2014).

Existem outros conceitos relacionados com *Internet Banking* mas que não têm o mesmo significado, que são *Mobile Banking* e *Home Banking*. *Mobile Banking* permite que os clientes do banco acedam aos serviços bancários através de um dispositivo móvel. *Mobile Banking* foi adotado em larga escala devido à crescente popularidade dos *smartphones* (Nawaz, Motiwalla, & Deokar, 2018). *Home Banking* abrange qualquer canal de entrega de serviços bancários de acesso remoto (Gkoutzinis, 2006). Segundo (Couto, Tiago, & Tiago, 2013) a história de *Home Banking* começou com processos de vendas por telefone, seguindo-se o acesso a ATM e computadores de linha discada. Atualmente, combina os canais referidos com *Internet Banking* e *Mobile Banking*.

1.2 Problema

O trabalho realizado nesta dissertação de mestrado, a decorrer em estágio curricular, integra o projeto designado "*Installments*" da ITSector, que é um dos projetos de *Internet Banking* da empresa. Este projeto foi requerido pelo Grupo Millennium BCP para adicionar uma nova operação bancária às suas aplicações móveis, que a operação de fracionamento de pagamentos. A área de negócio é a área de cartões, mais especificamente na vertente de fracionamentos de transações efetuadas via cartão de crédito.

Um fracionamento de pagamentos é um tipo de plano de pagamento que permite dividir um pagamento em prestações. O objetivo dos fracionamentos é dividir um pagamento de um grande montante em vários pagamentos de montantes mais baixos. Os pagamentos fracionados são acordados entre o cliente e o comerciante. As aplicações móveis do Millennium BCP ainda não contemplam este tipo de pagamento, deste modo, se os clientes pretenderem fracionar o seu pagamento terão de se deslocar às sucursais.

O Grupo Millennium BCP já tem aplicações móveis para os bancos Millennium BCP e ActivoBank. Estas são aplicações de *Mobile Banking* por serem aplicações móveis com acesso a serviços bancários. Mas também são aplicações de *Internet Banking*, porque todos os seus serviços só podem ser executados com acesso à Internet. Os bancos Millennium BCP e ActivoBank pertencem ao Grupo Millennium BCP, que é designado de banco cliente ao longo deste documento.

O projeto *Installments* é composto por várias fases, que são as funcionalidades bancárias que podem ser utilizadas cada vez que se pretenda fazer um pagamento fracionado. Estas funcionalidades devem ser implementadas no sistema de *Internet Banking* e são: a criação de planos de fracionamento sobre movimentos de cartão de crédito acima de um certo valor (neste momento o valor é de 150€); a liquidação total dos planos de fracionamento; a liquidação parcial de planos de fracionamento existentes; outros tipos de fracionamento (sem ser sobre cartões de crédito).

1.3 Objetivos

O trabalho a ser desenvolvido neste documento foca-se na primeira fase do projeto *Installments*. A primeira fase é a criação de planos de fracionamento sobre movimentos de cartão de crédito acima de um certo valor (neste momento o valor é de 150€).

Os objetivos deste trabalho que integra o projeto *Installments* é o desenvolvimento de funcionalidades novas nos microsserviços que suportam a operação bancária de fracionamento de pagamentos. E também a realização de testes unitários às funcionalidades já existentes nesse microsserviços.

Os microsserviços que se designam *Installments Process* e *Installments Experience* fornecem interfaces de comunicação para as aplicações móveis do Millenium BCP e do ActivoBank. Estes serviços foram criados para o *backend* do projeto *Installments*, e são desenvolvidos em .NET. Para auxiliar no desenvolvimento das funcionalidades é obrigatória a utilização de uma *framework* fornecida pelo banco cliente, que será descrita mais à frente.

As funcionalidades que já compõem a operação bancária de fracionamento de pagamentos são: a consulta de transações disponíveis para fracionamento; a consulta de planos de fracionamento correntes; a consulta dos detalhes de um dos planos; a criação de um novo plano de fracionamento; a simulação de planos de fracionamento para uma dada compra e/ou montante; a alteração de um plano de fracionamento existente; a simulação do cancelamento de um plano; e o cancelamento de um plano de fracionamento.

A funcionalidade nova que deve ser desenvolvida no projeto é o pedido de contacto ao banco cliente, onde é fornecido ao sistema as informações de contacto do cliente. Também deve ser refinada uma funcionalidade já implementada nos serviços, que é a consulta de planos correntes. Essa funcionalidade não passou nos testes de qualidade efetuados por *stakeholders* do projeto, e por esse motivo foi criado um bug. Esse bug descreve os planos correntes que devem ser disponibilizados ao cliente quando este não atribui filtro nos dados de entrada. Esse bug será descrito de forma mais detalhada mais à frente neste documento.

Após a implementação das novas funcionalidades, devem ser feitos testes unitários a todas as funcionalidades de ambos os serviços que cubram uma determinada percentagem do código. Também é requerida a remoção de *code smells* do código para aumentar a sua manutenibilidade.

A ordem de execução de cada uma das funcionalidades referidas deve obedecer a um determinado fluxo fornecido pelo banco cliente. Esse fluxo é composto por vários esboços de ecrã possíveis da aplicação de acordo com a tarefa que o utilizador está a realizar.

1.4 Estrutura

Este documento encontra-se dividido em vários capítulos. O capítulo 1 apresenta o contexto, o problema e os objetivos do presente trabalho. O capítulo 2 introduz conceitos sobre atributos de qualidade e estilos arquiteturais. Este capítulo também sintetiza o conhecimento sobre o projeto *Installments*, nomeadamente as decisões tecnológicas e arquiteturais. E finalmente, apresenta o estado de arte, que é uma recolha de estudos e projetos de *Internet Banking* onde sejam apresentadas as decisões arquiteturais para os sistemas desenvolvidos. O capítulo 3 apresenta a análise de valor realizada para o presente projeto. O capítulo 4 faz uma análise de requisitos ao trabalho realizado. O capítulo 5 faz o design da solução aplicado duas iterações da metodologia *Attribute-Driven Design*. O capítulo 6 apresenta a implementação da solução em dois casos de uso. O capítulo 7 apresenta os testes unitários feitos a um dos serviços. E finalmente, o capítulo 8 reúne as principais conclusões.

2 Estado de Arte

O presente capítulo introduz conceitos de arquitetura de *software* que são abordados ao longo deste documento, nomeadamente os atributos de qualidade e estilos arquiteturais. De seguida, apresenta alguns projetos de *Internet Banking* que aplicam os conceitos referidos. Esses projetos são o projeto *Installments* onde se insere este trabalho, e mais quatro mencionados na literatura sob o tema *Internet Banking*. Para todos os projetos mencionadas, é fornecido uma visão geral das tecnologias e estilos arquiteturais dos sistemas de *software* em estudo.

2.1 Atributos de Qualidade

Os atributos de qualidade são "propriedades mensuráveis ou testáveis de um sistema que são usadas para indicar o quão bem o sistema satisfaz as necessidades das suas partes interessadas" [1] (Cervantes & Kazman, 2016). Os atributos de qualidade podem estar relacionados com o tempo de execução, o *design* do sistema, ou com a experiência de utilização (Microsoft Patterns & Practices Team, 2009). Exemplos de atributos de qualidade são a usabilidade, o desempenho, a segurança, e a fiabilidade (Microsoft Patterns & Practices Team, 2009).

No processo de desenvolvimento de *software*, para além dos requisitos funcionais são necessários os requisitos não funcionais em sistemas de maior complexidade. Os requisitos de atributos de qualidade podem fazer parte do conjunto de requisitos não funcionais definidas para o sistema de *software* (Gorton, 2006). Estes requisitos devem ser específicos sobre como o sistema deve atingir uma determinada necessidade (Gorton, 2006). Por exemplo, não chega referir como requisito não funcional que o sistema deve ser testável, mas sim referir que os testes devem cobrir 70% do código. Isto que torna o requisito mais preciso e mensurável.

Segundo a norma IEEE 1061 a qualidade de *software* é definida como o grau no qual o *software* possui uma desejada combinação de atributos (Dobrica & Niemela, 2002). Portanto, uma determinada combinação dos atributos de qualidade pode contribuir para o sucesso geral do projeto (Microsoft Patterns & Practices Team, 2009).

O desenvolvimento de aplicações para determinados atributos de qualidade pode ter impacto em outros atributos de qualidade (Microsoft Patterns & Practices Team, 2009). A prioridade que é dada a cada atributo de qualidade depende das características do sistema que se

[1] Tradução livre dos autores. No original "[...] quality attributes are defined as being measurable or testable properties of a system that are used to indicate how well the system satisfies the needs of its *stakeholders*"

pretende desenvolver (Microsoft Patterns & Practices Team, 2009). Alguns exemplos de atributos de qualidade vão ser descritos nas secções que se seguem.

2.1.1 Testabilidade

A testabilidade é uma medida de quão bem o sistema de *software* ou os componentes permitem a criação de critérios de teste e execução de testes para determinar se os critérios são atendidos. Simplificando, mede o quão fácil é de testar o sistema. A testabilidade permite que as falhas do sistema sejam isoladas de forma oportuna e eficaz. Para se obter um sistema testável é necessário planear os testes, e que os testes devem ter uma boa cobertura (Microsoft Patterns & Practices Team, 2009).

Um dos problemas que pode afetar a testabilidade é a alta complexidade dos projetos (Microsoft Patterns & Practices Team, 2009). Como regra geral, quanto mais complexo for um projeto, mais difícil será testá-lo minuciosamente, por isso a simplicidade tende a facilitar o teste (Gorton, 2006). Da mesma forma, escrever menos código próprio incorporando componentes pré testados reduz o esforço de teste (Gorton, 2006). Outros problemas para a testabilidade são a falta de planeamento dos testes, a baixa cobertura dos testes, e as inconsistências entre entradas e saídas que dificultam a construção de casos de teste (Microsoft Patterns & Practices Team, 2009).

2.1.2 Suportabilidade

A capacidade de suporte é uma medida de quão fácil é o suporte de um sistema de *software* depois de implantado (Gorton, 2006). Sistemas com suporte fornecem informações úteis para diagnosticar e corrigir problemas quando este não funciona corretamente (Microsoft Patterns & Practices Team, 2009) (Gorton, 2006). A capacidade de suporte pode ser fornecida por ferramentas de rastreamento que façam uma monitorização do desempenho e atividade do sistema (Microsoft Patterns & Practices Team, 2009). Por exemplo, os *logs* dos erros das aplicações registam as causas das falhas e podem ser usados como recursos de suporte (Gorton, 2006). Os sistemas com suporte também são construídos de forma modular, para que as correções de código possam ser implantadas sem prejudicar muito a utilização (Gorton, 2006).

2.1.3 Disponibilidade

A disponibilidade define a proporção de tempo em que o sistema está funcional e em funcionamento. Este pode ser medido como uma percentagem do tempo de inatividade total do sistema durante um período predefinido. A disponibilidade pode ser afetada por erros do sistema, problemas de infraestrutura, ataques maliciosos ou pela carga do sistema (Microsoft Patterns & Practices Team, 2009).

Se a aplicação não estiver disponível quando esta é necessária então esta não poderá cumprir os seus requisitos funcionais. Em termos de especificação, normalmente as aplicações necessitam de disponibilidade de 24 horas por dia porque o horário comercial *online* não é regular (Gorton, 2006).

Os principais problemas que podem afetar a disponibilidade são a falha de um servidor que pode afetar todo o sistema, os ataques de negação de serviço, falhas na rede, *bug*s no sistema de *software*, ou o uso inadequado de recursos (Microsoft Patterns & Practices Team, 2009). As falhas no sistema para além de fazer com que as suas aplicações fiquem indisponíveis, afetam a sua confiabilidade do sistema (Gorton, 2006). A duração de qualquer período de indisponibilidade é determinada pelo tempo necessário para detetar a falha e reiniciar o sistema (Gorton, 2006).

2.1.4 Interoperabilidade

Interoperabilidade é a capacidade do sistema para comunicar com sistemas externos desenvolvidos e executados por terceiros e que utilizem diferentes formatos de dados. Um sistema interoperável facilita a troca e a reutilização de informações tanto interna quanto externamente. Alguns dos aspetos a serem considerados ao projetar um sistema interoperável são os protocolos de comunicação, interfaces, e os formatos de dados bem padronizados (Microsoft Patterns & Practices Team, 2009)

2.1.5 Capacidadade de gestão

A capacidade de gestão define a facilidade de gerir a aplicação. Normalmente essa gestão é feita utilizando instrumentação necessária em sistemas de monitorização e para depuração e ajuste de desempenho. A capacidade de gestão podes ser afetada pela falta de rastreabilidade e de informações de diagnóstico, pela falta de configurabilidade em tempo de execução, ou pela falta de ferramentas de solução de problemas (Microsoft Patterns & Practices Team, 2009)

2.1.6 Desempenho

O desempenho indica a capacidade de resposta de um sistema para executar qualquer ação dentro de um determinado intervalo de tempo. Pode ser medido pela latência ou pela taxa de transferência. A latência é o tempo necessário para responder a qualquer evento. A taxa de transferência é o número de eventos que ocorrem durante uma unidade de tempo. A melhoria do desempenho de um sistema de *software* geralmente melhora sua escalabilidade, reduzindo a probabilidade de contenção de recursos partilhados. E a falta de escalabilidade pode afetar o desempenho (Microsoft Patterns & Practices Team, 2009).

Um exemplo de requisito de taxa de transferência é garantir que uma aplicação de *Internet Banking* consegue executar mil transações por segundo dos seus clientes. É importante também distinguir se requisito de taxa de transferência refere-se exatamente à taxa de transferência média para um determinado período de tempo ou à taxa de transferência máxima (Gorton, 2006).

O tempo de resposta é muitas vezes associado ao tempo que a aplicação demora a responder a uma determinada entrada. Um rápido tempo de resposta é bom para o negócio porque permite os utilizadores trabalharem mais efetivamente. E também é importante distinguir entre o tempo de resposta máximo e o tempo de resposta médio (Gorton, 2006).

Há ainda uma terceira medida que se pode considerar para definir o desempenho que é o prazo de entrega. Qualquer aplicação que tenha tempo limitado necessita de ter um requisito de prazo (Gorton, 2006).

O aumento do consumo de largura de banda, o aumento do consumo de memória, e o aumento do processamento nas bases de dados são exemplos de problemas que podem reduzir o desempenho (Microsoft Patterns & Practices Team, 2009).

2.1.7 Fiabilidade

A fiabilidade ou confiabilidade é a capacidade de um sistema para continuar a operar da maneira esperada ao longo do tempo. A confiabilidade é medida como a probabilidade de um sistema não falhar no desempenho da sua função pretendida por um intervalo de tempo específico. Os problemas que podem afetar a confiabilidade de um sistema são a ausência de resposta, saídas inconsistentes, falta de desempenho, a indisponibilidade de sistemas externos (Microsoft Patterns & Practices Team, 2009).

2.1.8 Escalabilidade

A escalabilidade é a capacidade de um sistema de lidar com aumentos de carga sem impacto no desempenho do sistema ou a capacidade de ser facilmente ampliado (Microsoft Patterns & Practices Team, 2009). Uma solução escalável permitirá que uma capacidade de processamento adicional seja implantada para aumentar o rendimento e diminuir o tempo de resposta (Gorton, 2006). À medida que a carga aumenta, os sistemas de *software* terão uma diminuição na taxa de transferência e um aumento consequente do tempo de resposta (Gorton, 2006). Isso deve-se ao aumento do consumo de recursos como CPU e memória (Gorton, 2006).

A escalabilidade pode ser melhorada através de dois métodos, que são o dimensionamento vertical e o dimensionamento horizontal (Microsoft Patterns & Practices Team, 2009). No dimensionamento vertical são adicionados mais recursos como CPU, memória e disco a um único sistema (Microsoft Patterns & Practices Team, 2009). No dimensionamento horizontal

são adicionadas mais máquinas para distribuir a carga (Microsoft Patterns & Practices Team, 2009). A distribuição da carga entre várias máquinas é conhecida como balanceamento da carga (Gorton, 2006).

2.1.9 Segurança

Segurança é a capacidade de um sistema para reduzir a probabilidade de sofrer ações maliciosas imprevistas. Essas ações maliciosas podem afetar a disponibilidade do sistema e/ou permitir a divulgação ou perda de informações importantes (Microsoft Patterns & Practices Team, 2009).

Exemplos de requisitos de segurança são relacionados com a autenticação, a autorização, a encriptação, a integridade, o não repúdio. A autenticação refere-se à verificação da identidade dos utilizadores, a autorização refere-se aos direitos de acesso dos mesmos. A integridade garante que o conteúdo da mensagem não é alterado em transito, e o não repudio garante que o emissor e o recetor não possam refutar a sua participação na troca da mensagem (Gorton, 2006).

A melhoria da segurança também pode aumentar a confiabilidade do sistema, reduzindo a probabilidade de sucesso de um ataque que prejudique a operação do sistema. Os fatores que afetam a segurança do sistema são a confidencialidade, a integridade e a disponibilidade (Microsoft Patterns & Practices Team, 2009).

2.1.10 Integridade conceitual

A integridade conceitual define a consistência e coerência na maneira como os componentes ou módulos são projetados, no estilo de codificação e na nomenclatura de variáveis de um projeto. Isto para dar consistência ao projeto em geral. Um sistema coerente é mais fácil de manter porque é consistente com o projeto geral. Por outro lado, um sistema sem integridade conceitual será mais frequentemente afetado por mudanças de interfaces, módulos obsoletos e pela falta de consistência na execução de tarefas (Microsoft Patterns & Practices Team, 2009).

Alguns dos fatores que afetam a integridade são a falta de padrões de *design*, falta de colaboração entre a equipa de desenvolvimento e os processos de desenvolvimento mal geridos. A mistura de diferentes áreas de interesse no projeto, como a lógica e os dados, e a integração com sistemas existentes são outros exemplos de problemas da integridade conceitual (Microsoft Patterns & Practices Team, 2009).

2.1.11 Manutenibilidade

A manutenibilidade é a capacidade de um sistema para sofrer alterações com facilidade. As alterações podem ter o objetivo de adicionar ou alterar a funcionalidade, corrigir erros ou atender a novos requisitos de negócios. Essas alterações podem afetar os seus componentes, serviços, recursos e interfaces, conforme necessário. Melhorar a capacidade de manutenção do sistema pode aumentar a disponibilidade e reduzir os efeitos de defeitos em tempo de execução (Microsoft Patterns & Practices Team, 2009).

Alguns problemas que podem afetar a manutenibilidade são a dependência de implementações personalizadas, a falta de documentação, e a ausência de testes automatizados (Microsoft Patterns & Practices Team, 2009). A baixa coesão e alto acoplamento entre os componentes, camadas e classes do sistema também afetam negativamente a manutenibilidade.

2.1.12 Reutilização

Reutilização é a probabilidade de um componente de um *software* ser usado em outros componentes ou cenários para adicionar novas funcionalidades com pouca ou nenhuma modificação (Microsoft Patterns & Practices Team, 2009). A reutilização minimiza a duplicação de componentes e o tempo de implementação (Microsoft Patterns & Practices Team, 2009). A identificação de atributos comuns entre os vários componentes permite a construção de componentes reutilizáveis (Microsoft Patterns & Practices Team, 2009). A reutilização pode facilitar a manutenibilidade.

2.1.13 Usabilidade

A usabilidade é a capacidade de as interfaces da aplicação fornecerem uma boa experiência geral dos utilizadores. As interfaces devem ter em mente o perfil dos utilizadores que a vão consumir, incluindo para os utilizadores com deficiência (Microsoft Patterns & Practices Team, 2009).

A experiência do utilizador ou usabilidade pode ser afetada pelo excesso de interação ou elevado número de *clicks*. Os fluxos incorretos de etapas nas interfaces e os elementos das interfaces mal agrupados também complicam a utilização. Por fim, a falta de resposta da aplicação resultante de erros e exceções também contribuem para as opiniões negativas dos utilizadores (Microsoft Patterns & Practices Team, 2009).

2.1.14 Modificabilidade

A modificabilidade é a capacidade de reduzir o custo das alterações de qualquer aspeto no sistema. Essas alterações podem ser feitas nas funções que o sistema calcula, na plataforma

onde o sistema executa, o ambiente no qual o sistema opera, as qualidades que este sistema exibe, entre outros. Essas alterações podem ser a adição, a modificação, ou a eliminação de qualquer um dos aspetos referidos. As alterações podem ser feitas na implementação, durante a compilação, durante a configuração, ou durante execução. Uma alteração pode ser feita por um desenvolvedor, um administrador do sistema, ou um utilizador final. Um utilizador final alterar os ecrãs de bloqueio é um exemplo de alteração de um dos aspetos do sistema (Bass, Clements, & Kazman, 2003).

Sempre que uma mudança for especificada, a nova implementação deve ser projetada, implementada, testada e implantada. Todas essas ações levam tempo e dinheiro, e ambos podem ser medidos (Bass, Clements, & Kazman, 2003).

A modificabilidade pode ser afetada pela reutilização, porque a reutilização de componentes permite que as modificações posteriores afetem o menor número de módulos possível. Este atributo de qualidade relaciona-se com a manutenibilidade, mas também abrange a capacidade de permitir que não desenvolvedores possam fazer alterações no sistema.

2.2 Estilos arquiteturais

Um estilo arquitetural, que pode ser designado de padrão arquitetural, é um conjunto de princípios de granularidade grossa que definem a organização estrutural de uma família de sistemas. Um estilo arquitetural é independente da tecnologia, e pode fornecer soluções para problemas recorrentes do projeto (Microsoft Patterns & Practices Team, 2009).

Os estilos podem ser divididos por categorias, dependendo da sua área de atuação. As áreas de foco podem ser a comunicação, o domínio, a implantação e a estrutura. A arquitetura de um sistema de *software* pode ser composta por mais do que um estilo arquitetural. Por exemplo, a SOA pode ser combinada com a arquitetura por camadas e o estilo orientado a objetos (Microsoft Patterns & Practices Team, 2009).

Alguns dos estilos arquiteturais que se considera mais relevantes vão ser apresentados nas secções que se sequem.

2.2.1 Arquitetura cliente/servidor

A arquitetura cliente/servidor é um estilo arquitetural que permite dividir o sistema de *software* em cliente e servidor, separados por uma ligação em rede, onde o cliente faz requisições ao servidor. A forma mais simples de sistema cliente/servidor envolve uma aplicação de servidor que é acedida diretamente por vários clientes (Microsoft Patterns & Practices Team, 2009).

Os principais benefícios do estilo arquitetural cliente/servidor são a segurança porque os dados são armazenados no lado do servidor, que oferece maior segurança do que o lado

cliente. Outro benefício é o acesso centralizado a dados, facilitando o acesso e as atualizações de dados. A facilidade de manutenção é outro benefício, se o lado do servidor for constituído por vários sistemas distribuídos, permitindo que o cliente não seja afetado pelas atualizações (Microsoft Patterns & Practices Team, 2009).

As desvantagens deste estilo incluem a redução da extensibilidade e da escalabilidade que podem impactar negativamente na confiabilidade do sistema de *software*. A utilização da arquitetura das três camadas supera algumas das desvantagens do estilo cliente/servidor e fornece benefícios adicionais (Microsoft Patterns & Practices Team, 2009).

2.2.2 Arquitetura baseada em componentes

O estilo arquitetural baseado em componentes decompõe o *design* do sistema de *software* em componentes funcionais ou lógicos reutilizáveis que expõem interfaces de comunicação bem definidas. Este estilo fornece um nível de abstração alto, sem preocupação com questões tecnológicas como protocolos de comunicação. Cada componente deve ser independente, encapsulado, extensível, substituível, reutilizável e não específico de contexto (Microsoft Patterns & Practices Team, 2009).

Existem plataformas que incluem mecanismos, i.e., arquiteturas de componentes, que fornecem ambientes para a execução de componentes distribuídos. Exemplos dessas arquiteturas no Windows são o Modelo de Objeto de Componente (COM) e o modelo de objeto de componente distribuído (DCOM). Exemplos de outras plataformas são o CORBA e o EJB. As arquiteturas de componentes gerem a mecânica da localização de componentes e das suas interfaces, passando mensagens ou comandos entre componentes (Microsoft Patterns & Practices Team, 2009).

Alguns dos benefícios da arquitetura orientada a componentes são a facilidade de desenvolvimento e de implantação devido à utilização de interfaces bem conhecidas, que permitem a adição e alteração de componentes sem afetar os existentes. O custo reduzido de desenvolvimento e manutenção é outra vantagem devido à possibilidade de utilizar componentes de terceiros e/ou componentes reutilizáveis. A utilização de componentes também mitiga a complexidade técnica (Microsoft Patterns & Practices Team, 2009).

2.2.3 Arquitetura em camadas

A arquitetura em camadas divide os interesses comuns da aplicação em camadas distintas. A comunicação entre as camadas é fracamente acoplada. Esta divisão da aplicação em camadas funcionais oferece suporte à flexibilidade e manutenção (Microsoft Patterns & Practices Team, 2009).

As camadas de uma aplicação podem residir no mesmo computador físico (o mesmo nível) ou podem estar distribuídas ao longo de computadores separados (vários níveis). E os

componentes de cada camada comunicam com componentes de outras camadas através de interfaces bem definidas. Por exemplo, um *design* típico de uma aplicação Web apresenta uma camada de apresentação, que possui funcionalidades relacionada à UI, uma camada de negócios que processa as regras de negócios, e uma camada de dados, para o acesso a dados (Microsoft Patterns & Practices Team, 2009).

Os principais benefícios da arquitetura em camadas são a abstração, o isolamento, a capacidade de gestão, o desempenho, a reutilização e a testabilidade. Um dos benefícios é o isolamento das atualizações tecnológicas de camadas individuais, reduzindo o impacto no sistema em geral. A distribuição das camadas em vários níveis também pode melhorar a escalabilidade, a tolerância a falhas e o desempenho. Outro benefício que se destaca é testabilidade devido às interfaces entre as camadas bem definidas. Isto permite a criação de *mocks* que simulem o funcionamento de uma determinada camada. Este estilo arquitetural também permite a reutilização dos papeis em outras aplicações (Microsoft Patterns & Practices Team, 2009).

O estilo arquitetural em camadas é adequado para a implementação de regras de negócio complexos, permitindo a divisão das equipas em diferentes papeis. Também é apropriado se a aplicação necessitar de suportar diferentes tipos de clientes e dispositivos (Microsoft Patterns & Practices Team, 2009).

2.2.4 Arquitetura *N-Tier/3-Tier*

Os estilos arquiteturais *n-tier* e *3-tier*, tal como o estilo arquitetural em camadas, permitem separar as funcionalidades em camadas. Mas as camadas estão localizadas em servidores físicos separados, podendo ser designadas de níveis. A arquitetura *n-tier* é composta por pelo menos 3 camadas lógicas localizadas cada uma num servidor separado (Microsoft Patterns & Practices Team, 2009).

Cada nível só precisa de comunicar com o nível que está imediatamente acima, do qual recebe pedidos, e o nível que está imediatamente abaixo, se existir, onde reencaminha os pedidos recebidos. A comunicação é normalmente assíncrona para oferecer suporte a uma melhor escalabilidade (Microsoft Patterns & Practices Team, 2009).

Um exemplo de aplicação do estilo arquitetural *N-tier/3-tier* é uma aplicação Web de *Internet Banking*, onde a segurança é importante. Este estilo arquitetural pode ser aplicado em projetos onde os requisitos de segurança nos níveis do sistema diferem. Por exemplo, as camadas de negócio e de dados podem armazenar dados sensíveis, o mesmo pode não acontecer na camada da apresentação (Microsoft Patterns & Practices Team, 2009).

Os principais benefícios deste estilo arquitetural são a manutenibilidade, a escalabilidade, a flexibilidade e a disponibilidade. A flexibilidade e a manutenibilidade devem-se à possibilidade de cada nível ser gerido e dimensionado de forma independente. A escalabilidade e a

disponibilidade são proporcionadas pela implantação distribuída das camadas (Microsoft Patterns & Practices Team, 2009).

2.2.5 Arquitetura de barramento de mensagens

O estilo arquitetural de barramento de mensagens descreve o uso de um sistema de *software* que pode receber e enviar mensagens, que são geralmente assíncronas, através de um ou mais canais de comunicação. Trata-se de um estilo para projetar sistemas em que a interação entre serviços é realizada pela passagem de mensagens por um barramento comum. Deste modo, as aplicações de *software* podem interagir entre si sem precisarem de conhecer detalhes específicos uns dos outros, apenas precisam de saber como comunicar com o barramento. Um barramento de mensagens fornece a capacidade de lidar com comunicações orientadas a mensagens, logicas de processamento complexas, modificações na lógica de processamento e integração com diversos ambientes (Microsoft Patterns & Practices Team, 2009).

Os principais benefícios do estilo de barramento por mensagens são a extensibilidade, flexibilidade, baixo acoplamento, a escalabilidade e a simplicidade de aplicação. A arquitetura do barramento permite anexar aplicações ao processo, ou inserir várias instâncias da mesma aplicação para melhorar a escalabilidade. Embora uma implementação de barramento de mensagens adicione complexidade à infraestrutura, cada aplicação precisa apenas de suportar uma única ligação com o barramento em vez de várias ligações com outras aplicações (Microsoft Patterns & Practices Team, 2009).

2.2.6 Arquitetura Orientada a Serviços (SOA)

O estilo arquitetural SOA estrutura o sistema de *software* para que as suas funcionalidades sejam fornecidas por um conjunto de serviços autónomos, distribuíveis e fracamente acoplados. Os serviços utilizados têm baixo acoplamento porque usam interfaces baseadas em padrões que podem ser invocadas, publicadas e descobertas. Estes serviços interagem entre si partilhando esquema e contrato, mas não partilham classes (Microsoft Patterns & Practices Team, 2009).

O estilo SOA pode empacotar processos de negócio em serviços interoperáveis, usando uma variedade de protocolos e formatos de dados para comunicar informações. Um cliente ou serviço tanto pode interagir com outro serviço local, como pode comunicar com outro serviço remoto por meio da ligação à rede (Microsoft Patterns & Practices Team, 2009).

Os principais benefícios do estilo arquitetural SOA são a alinhamento do domínio, a abstração, a descoberta, a interoperabilidade e a racionalização. O alinhamento de domínio deve-se à reutilização dos serviços comuns com interfaces padrão, permitindo reduzir custos e aumentar oportunidades de negócio e tecnológicas. A abstração deve-se à autonomia dos serviços e ao acesso através de um contrato formal a abstração, determinado o baixo

acoplamento dos serviços. A interoperabilidade permite que os serviços possam ser criados e implantados em diferentes plataformas. A racionalização deve-se à capacidade dos serviços para fornecer funcionalidades específicas, em vez de duplicar a funcionalidade em várias aplicações (Microsoft Patterns & Practices Team, 2009).

Os sistemas SOA podem necessitar de uma camada de integração entre os serviços, denominada de barramento. Caso não exista esse barramento, o cliente terá de comunicar com cada serviço diretamente para cada necessidade (IBM Cloud Education, 2021). Isto aumenta a complexidade, e cria problemas de manutenção no futuro (IBM Cloud Education, 2021).

2.2.7 Arquitetura de Microsserviços

A arquitetura de microsserviços é um estilo arquitetural que estrutura uma aplicação como um conjunto de serviços que são organizados em torno dos recursos de negócios (Richardison, 2014). Os microsserviços são pequenas aplicações que executam uma funcionalidade de uma área de negócios específica (IBM Cloud Team, 2021). Os microsserviços são pequenos, independentes e fracamente acoplados (Microsoft, 2022). A arquitetura de microsserviços permite a entrega rápida, frequente e confiável de aplicações grandes e complexos. (Richardison, 2014).

A arquitetura de microsserviços é um estilo arquitetural semelhante ao SOA. Tal como em SOA, um sistema com esta arquitetura é composto por serviços pouco acoplados, altamente coesos, reutilizáveis, e que podem funcionar de forma independente. Uma das principais diferenças entre estes dois estilos arquiteturais é que as arquiteturas de microsserviços são compostas por serviços mais especializados. Enquanto que as aplicações SOA podem ter serviços de tamanhos superiores. Os sistemas na SOA também operam mais lentamente do que nos microsserviços. A arquitetura de microsserviços pode permitir a duplicação de dados para trazer maior desempenho. Enquanto que a SOA tem maior reutilização de serviços para aumentar a escalabilidade e a eficiência. No caso das arquiteturas de microsserviços, não existe o barramento com o qual os serviços partilhem o seu mecanismo de comunicação. Os protocolos de mensagens dos microsserviços são geralmente mais leves do que os SOA que são heterogéneos. Em suma, a SOA pode ser de âmbito corporativo e a arquitetura de microsserviços é do âmbito de uma aplicação, o que potencia a complementaridade entre estas duas arquiteturas (IBM Cloud Team, 2021).

A Figura 1 estrutura um sistema de *software* fictício criado por (Richardson, 2019) que apresenta a arquitetura de microsserviços.

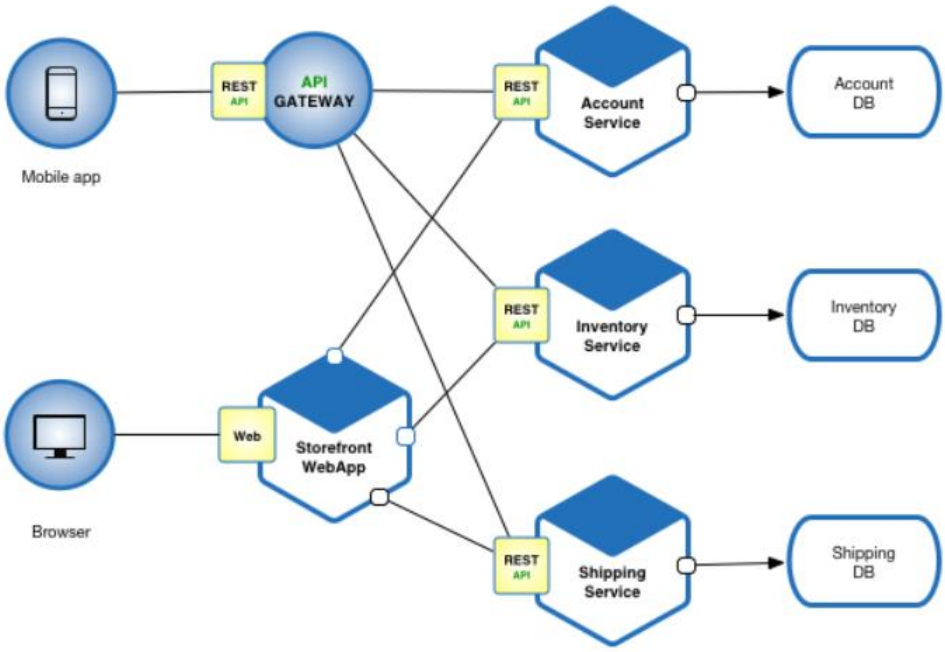

Figura 1 – Exemplo de aplicação de *e-comerce* fictícia (Richardson, 2019)

Tal como se verifica na Figura 1, cada serviço representa uma área funcional, e estes comunicam entre pela interface REST API. Cada serviço funciona de forma independente, contendo a sua própria base de dados. A *API Gateway* é um componente que surge numa arquitetura típica de microsserviços e é o ponto de entrada dos clientes para os microsserviços. O cliente em vez de chamar os serviços diretamente, chama a *API Gateway* que encaminha as chamadas para os serviços adequados no *backend* (Microsoft, 2022). Uma das boas práticas que se deve adotar neste tipo de arquitetura é a utilização de serviços com baixo acoplamento e alta coesão, de forma a evitar que as alterações em um serviço afetem outros serviços (Microsoft, 2022). No caso da Figura 1, o acoplamento pode ser reduzido utilizando apenas uma API Gateway chamada pela aplicação móvel e pelo *browser*.

Os benefícios desta arquitetura incluem a melhor manutenibilidade e a melhor testabilidade devido ao tamanho pequeno de cada serviço (Richardson, 2019). Cada serviço pode ser implantado de forma independente dos outros (Richardson, 2019). Tal fornece maior agilidade na gestão das correções de erros e nos novos lançamentos. O isolamento das falhas permite que se um serviço tiver uma falha, os outros serviços não serão afetados e continuarão a processar os pedidos (Richardson, 2019). Os benefícios referidos permitem a entrega e implantação contínuas de sistemas grandes e complexos (Richardson, 2019). Outro benefício importante é a utilização de equipas pequenas e focadas em criar, testar e implantar cada microsserviço (Microsoft, 2022). Estas equipas são autónomas porque desenvolvem os seus serviços independentemente das outras equipas (Richardson, 2019). Este estilo também pode eliminar qualquer compromisso tecnológico de longo prazo (Richardson, 2019). Cada equipa pode escolher as tecnologias que melhor se adequam ao seu serviço (Microsoft, 2022).

Os microsserviços também tem as suas desvantagens devido à complexidade adicional de criar sistemas distribuídos (Richardson, 2019). Apesar de cada serviço ser mais simples, o sistema como um todo é mais complexo (Microsoft, 2022). A implementação e os testes das comunicações entre os vários serviços são mais complicados (Richardson, 2019). A implantação de um sistema com diferentes serviços tem maior complexidade operacional (Richardson, 2019). A utilização de diferentes linguagens em diferentes serviços também pode dificultar a manutenção (Microsoft, 2022).

2.3 Sistema *Installments*

O projeto *Installments* visa desenvolver uma solução de *software* composta por várias camadas. Esta secção sintetiza o conhecimento sobre a solução *Installments* na qual se insere este trabalho, apresentando a sua estrutura e as tecnologias utilizadas.

O sistema *Installments* faz parte de uma solução de *Internet Banking* já existente do banco cliente que combina o estilo arquitetural *n-tier* com a arquitetura de microsserviços. Um dos estilos é a arquitetura de microsserviços porque cada serviço é responsável por uma área de negócio, onde neste caso apresenta-se apenas os serviços dedicados aos fracionamentos de pagamentos. O estilo *n-tier* é aplicado porque a *framework* do banco cliente permite que cada microsserviço seja implantado num *cluster*.

As camadas físicas da solução *Installments* encontram-se apresentadas no diagrama de componentes da Figura 2.

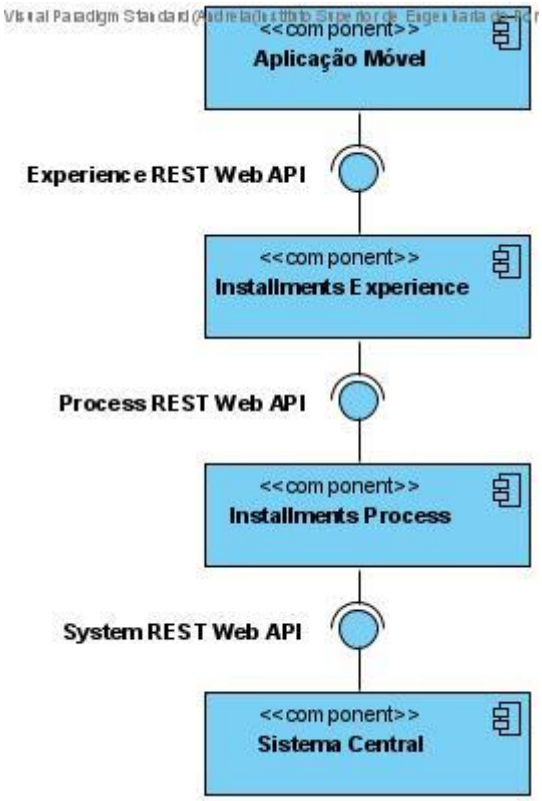

Figura 2 – Diagrama de componentes de *Installments*

O Sistema Central é composto por serviços bancários do banco cliente que não foram desenvolvidos pela equipa da ITSector, e apenas conseguem ser acedidos através de comunicação REST. Este componente fornece os serviços de *Internet Banking* para que possam ser processados pelas camadas seguintes.

Installment Process é a camada da lógica do negócio, portanto é responsável por executar todas as funcionalidades necessárias para o processo de fracionamento se concretize. Essas funcionalidades foram enumeradas na secção 1.3. Este componente irá comunicar diretamente com o sistema central do banco cliente.

Installments Experience é responsável por consumir os dados fornecidos pelo *Process* e extrair apenas aqueles que serão visualizados no ecrã da aplicação móvel. Este serviço comunica diretamente com a camada de *frontend* do sistema, aplicando assim o padrão *bakend-for-fronted*. Como será apenas necessário um tipo de esboços de ecrã para aplicações móveis, esta camada é constituída por apenas um projeto. Se fosse necessária uma versão Web para este projeto onde os esboços dos ecrãs disponibilizariam dados diferentes teria de ser criado um novo serviço *Experience*.

A Aplicação Móvel é o canal de comunicação entre o utilizador e os serviços de *Internet Banking* do banco cliente, que funciona de forma *online*. A aplicação móvel integra pacotes de projetos existentes em React Native para que sejam suportados pelos sistemas operativos IOS e Android. Os projetos de React Native que executam os fluxos para *Installments* são os

projetos *Installments Frontend* e *Installments API*. A aplicação *Installments Frontend* executa todos os fluxos e ecrãs para executar os fracionamentos. E a aplicação *Installments API* funciona como *API gateway*, que redireciona os pedidos da aplicação *Installments Frontend* para o serviço *Installments Experience*. Este projeto não realiza qualquer operação lógica. A sua única função é adaptar os pedidos do *Installments Fronted* à API do serviço *Installments Experience*, de forma a possibilitar a comunicação entre a aplicação móvel e o serviço.

Este trabalho irá focar-se nos componentes *Installments Experience* e *Installments Process*. Estes serviços são antecedidos pelo nome "*Installments*" porque existem mais serviços *Process* e *Experience* referentes a outros projetos do banco cliente. Ambos os componentes são microsserviços desenvolvidos em .NET 5. Cada microsserviço executa um conjunto de funcionalidades necessárias para executar a operação bancária. Essas funcionalidades foram desenvolvidas com o auxílio de uma *framework* criada especificamente para os projetos do banco cliente.

A *framework* denominada 3DF usada no desenvolvimento dos microsserviços *Installments Experience* e *Installments Process* apenas é disponibilizada para a equipa da ITSector dedicada aos projetos do banco cliente. Cada microsserviço 3DF possui um conjunto de abstrações aos recursos comuns das aplicações, por exemplo a autenticação, as metricas, e a cache, que permitem que a implementação se concentre apenas nas funcionalidades do negócio. Assim é reduzido o tempo de desenvolvimento.

As tecnologias que devem ser utilizadas na construção dos microsserviços são .NET 5.0, ASP.NET Web API, Redis e Cosmos DB. .NET é uma *framework* multiplataforma de desenvolvimento de aplicações. ASP.NET Web API é uma *framework* para construir serviços HTTP sobre .NET. Redis será usado para armazenamento de cache distribuída, e Cosmos DB é um serviço de base de dados NoSQL.

2.4 Sistemas de *software* de *Internet Banking*

Esta secção apresenta um estado de arte propriamente dito para sistemas de *software* de *Internet Banking*. São apresentados quatro sistemas de *Internet Banking* documentados em artigos, que adotaram diferentes estilos arquiteturais. Os projetos apresentam diferentes graus de complexidade. Enquanto a secção 2.4.1 sugere um sistema que integra múltiplos canais de distribuição. A secção 2.4.4 apresenta um sistema cujo único canal de distribuição é uma aplicação de Mobile Banking. Em cada solução de *Internet Banking* é apresentado o projeto em que esta foi utilizado, e são abordadas as decisões arquiteturais e tecnológicas.

2.4.1 Projeto de SOA baseada em *Cloud* proposto por (Appandairaj & Murugappan, 2013)

A solução proposta por (Appandairaj & Murugappan, 2013) é a mais complexa de todas as apresentadas neste artigo, porque para além de poder utilizar uma aplicação Web como canal

de distribuição, também pode utilizar outros canais de distribuição para comunicar com os seus clientes. Trata-se por isso de um sistema de *Home Banking*.

O projeto consiste no desenvolvimento de um *software* como serviço (SaaS) para ambiente de *cloud*, utilizando os princípios SOA e destinado ao setor bancário. O modelo SOA proposto pretende ajudar a reduzir a redundância, a inflexibilidade e a ineficiência em processos bancários. Os requisitos do sistema são implementação e integração de aplicações de pagamentos, integração multicanal e abertura de conta (Appandairaj & Murugappan, 2013).

A Figura 3 apresenta os componentes lógicos da arquitetura definida para o protótipo de SaaS para *Home Banking* e implementado por (Appandairaj & Murugappan, 2013).

Figura 3 – Componentes lógicos (Online Payment, Multichannel, New account Opening) (Appandairaj & Murugappan, 2013)

Pela Figura 3 observa-se que o sistema SOA para *cloud banking* tem várias camadas de serviços tais como o Serviço de Infraestrutura, Serviços da Plataforma, as camadas de Segurança, os Serviços de Canais e os Serviços à Empresa que são necessárias na implantação do sistema. Destaca-se os Serviços do Negócio que englobam todos os serviços bancários essenciais e os Serviços de Canais que suportam vários canais, como ATM, agência, *call center*, correio, SMS, serviço *online* e serviço móvel (Appandairaj & Murugappan, 2013).

Os parágrafos abaixo enumeram as vantagens da utilização do estilo SOA não só no projeto desenvolvido por (Appandairaj & Murugappan, 2013), mas também para a implementação de outros sistemas bancários com vários canais.

A implementação de um sistema SOA tem várias vantagens para o requisito dos pagamentos, tais como a possibilidade de integração de novos serviços de pagamento às capacidades de pagamento existentes. Um sistema SOA também permite a reutilização de serviços para a realização de diferentes tipos de transações. Estes serviços são independentes dos canais de pagamento que eles suportam, deste modo o banco pode adicionar facilmente novos canais de pagamento e novos serviços sem perturbar o negócio. Esta flexibilidade posiciona o banco para novos canais de pagamento e novas fontes e alvos de pagamento (Appandairaj & Murugappan, 2013).

A implementação de um sistema SOA, segundo os autores, tem vários benefícios na integração de diferentes canais. As aplicações de canal, tais como por exemplo uma agência de retalho ou um serviço Web, podem consumir várias categorias de serviços, tais como por exemplo as informações do cliente ou os saldos, que podem ser difíceis de integrar. O sistema SOA fornece, assim uma camada de integração dos vários canais de distribuição de forma ter uma visão abrangente da relação com o cliente, fornecendo assim serviços mais personalizados. Isto ajuda a melhorar a flexibilidade para mudanças e a distribuição dos produtos, e a reduzir tempo, custos e riscos (Appandairaj & Murugappan, 2013).

A implementação de um sistema SOA também apresenta benefícios para as aplicações de abertura de nova conta. Uma solução SOA também pode permitir a abertura de uma conta para várias linhas de produtos que são integradas com vários sistemas de *backend*. Os benefícios podem incluir custos mais baixos, aumento de receita e melhores relações com os clientes (Appandairaj & Murugappan, 2013).

Tal como se descreveu nos últimos três parágrafos, os autores consideram que SOA é a arquitetura que melhor cumpre os requisitos para o projeto de SOA baseado em *cloud* para *Home Banking*.

Segundo (Appandairaj & Murugappan, 2013), a SOA oferece reutilização, manutenibilidade, fácil integração dos serviços, interoperabilidade, desempenho, confiabilidade e segurança à solução de *software*.

A SOA permite que o setor bancário tenha agilidade para responder melhor às mudanças nas necessidades dos clientes, nos negócios, no mercado e nas regulamentações (Appandairaj & Murugappan, 2013).

A adoção de SOA também reduz o tempo desde a conceção do produto até ao seu lançamento no mercado. Depois de construção da arquitetura inicial, adições ou mudanças podem ser feitas de forma muito mais rápida e menos dispendiosa (Appandairaj & Murugappan, 2013).

2.4.2 Solução de Wells Fargo

A empresa de serviços financeiros Wells Fargo começou em 1994 a investir em aplicações distribuídas orientadas a objetos usando a primeira versão do CORBA (Kainz, 2004). Entre 2001 e 2003, a solução original baseada em CORBA foi completamente substituída por novas aplicações distribuídas baseados em serviços Web. (Kainz, 2004) retrata as lições aprendidas durante a transição de uma solução SOA baseada em CORBA para uma baseada em Serviços Web (Kainz, 2004).

A arquitetura *3-tier* da solução de *Internet Banking* da Wells Fargo encontra-se ilustrada na Figura 4.

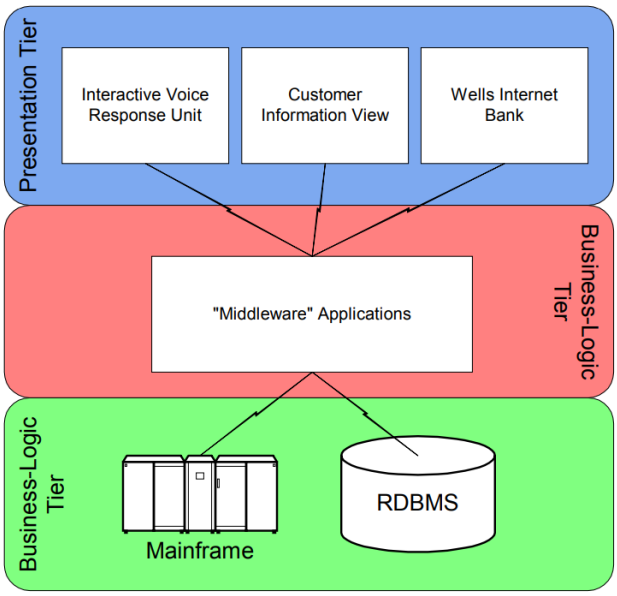

Figura 4 – Arquitetura de *3-tier* de Wells Fargo (Kainz, 2004)

A camada da apresentação é composta por várias aplicações ou canais, entre as quais a Wells Fargo Phone-Bank que fornece serviços bancários por meio de um sistema de Resposta de Voz Interativa. A camada da base de dados é implementada quase inteiramente por mainframes. Existem também alguns sistemas RDBMS muito grandes que implementam aspetos menos significativos do modelo. A lógica do negócio é implementada na camada intermédia e é

partilhada por vários canais. A vantagem deste *design* é que a lógica do negócio não precisa de ser implementada várias vezes para diferentes canais (Kainz, 2004).

A camada intermédia fornece uma visão polimórfica dos dados armazenados por um conjunto diversificado de sistemas de *backend*. As aplicações *frontend* podem representar informações da conta de forma consistente, representando muitos produtos, como empréstimos de contas de poupança, cartões de crédito e carteiras de ações. O recurso ao polimorfismo permite que os clientes do banco visualizem as informações da sua conta na sua página Web personalizada da Wells Internet Fargo, além de realizar vários recursos de *self-service*, como transferências de dinheiro (Kainz, 2004).

Segundo o autor é mais produtivo utilizar bons princípios de *design* do que seguir as tendências do setor. Neste caso, a arquitetura em camadas permite que a implementação isole os aspetos dos canais de distribuição da lógica do negócio. Usar este tipo de *design* é inicialmente mais caro, mas ajuda a isolar muitos tipos de alterações. Normalmente, requer apenas uma pequena subequipa para implementar uma nova camada (Kainz, 2004).

2.4.3 Aplicação Duke's Bank

A aplicação bancária Duke's Bank é utilizada como caso de estudo para os tutoriais como o de Java EE5 em (Oracle, 2010), e também por vários autores como (Bahsoon, Emmerich, & Macke, 2005).

A aplicação bancária Duke's Bank foi selecionada para ser caso de estudo por (Bahsoon, Emmerich, & Macke, 2005), porque representa de forma adequada para os autores uma arquitetura distribuída baseada em componentes de tamanho médio. Foram instanciadas duas versões da arquitetura de *software* do Duke's Bank, cada uma induzida por uma tecnologia de *middleware* distinta, uma com CORBA e outra com J2EE. O objetivo era estudar como arquiteturas de *software* induzidas por *middlewares* distintos podem diferir em lidar com mudanças em requisitos não funcionais. O requisito não funcional estudado foi a escalabilidade. O estudo concluiu que o estilo arquitetural não é o único fator para a estabilidade da arquitetura de *software* quando os requisitos não funcionais evoluem, mas é um fator de extensão (Bahsoon, Emmerich, & Macke, 2005).

O sistema de *software* Duke's Bank apresenta um estilo arquitetural 3-tier, onde na camada da apresentação existem dois clientes. Os clientes são a uma aplicação cliente usada pelos administradores para gerir os clientes do banco e as contas bancárias, e o cliente Web usado pelos clientes para aceder a extratos de contas e realizar transações. Os componentes do lado do servidor executam métodos do negócio para a gestão de clientes, gestão de contas e gestão de transações. As informações de clientes, contas e transações são mantidas em bases de dados (Bahsoon, Emmerich, & Macke, 2005).

A Figura 5 fornece uma visão de alto nível dos 6 componentes EJB do lado servidor do sistema Duke´s Bank.

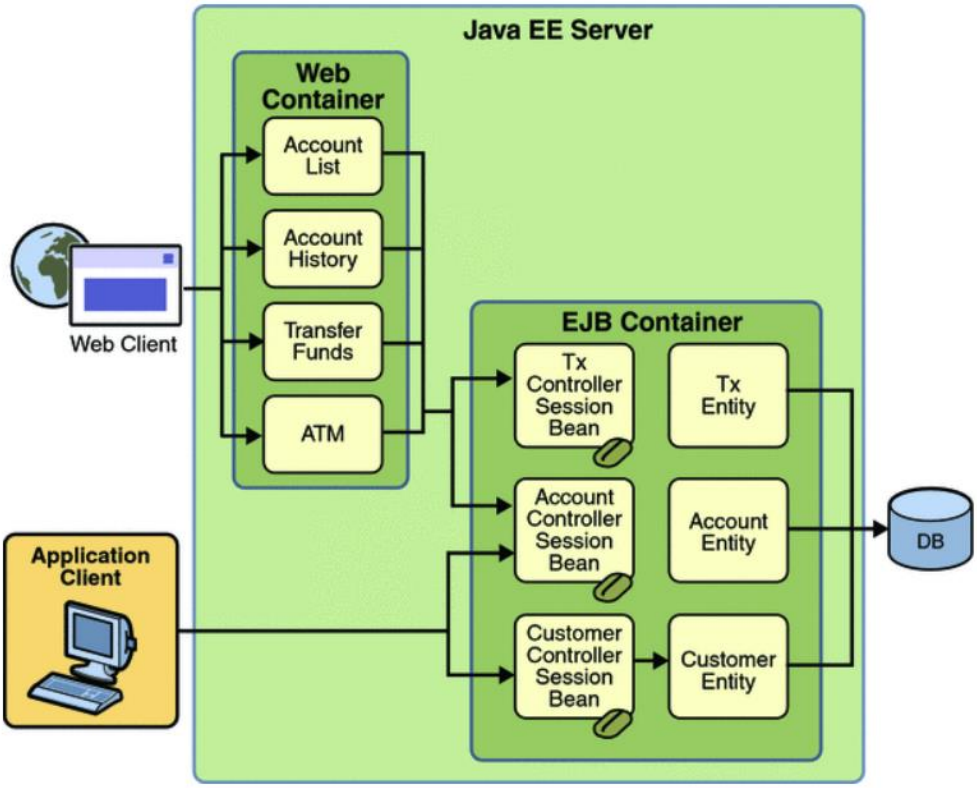

Figura 5 – Componentes do Duke's Bank (Oracle, 2010)

A Aplicação Cliente e o Cliente Web acedem apenas aos *beans* de sessão. O sistema de Duke's Bank tem três beans de sessão que são: *AccountControllerBean*, *CustomerControllerBean*, e *TxControllerBean*, onde Tx significa transação de negócios como a transferência de fundos. Esses *beans* de sessão fornecem a lógica de negócios e comunicam com as respetivas entidades Java Persistence que são: *Account*, *Customer* e *Tx*. Essas entidades, por sua vez, acedem as tabelas da base de dados que armazenam os estados das entidades (Oracle, 2010).

2.4.4 Sistema PERMA

O projeto apresentado por (Nawaz, Motiwalla, & Deokar, 2018) consiste numa aplicação de *Mobile Banking* personalizada que possui uma interface com o utilizador (UI) adaptável às necessidades do utilizador, permitindo-o ter uma interface personalizada. Esta aplicação móvel faz parte do sistema PERMA que foi utilizado como protótipo de pesquisa por (Nawaz, Motiwalla, & Deokar, 2018) para serem feitos testes empíricos ao impacto dessa personalização. A aplicação deve obedecer a um conjunto de requisitos para o estudo. Um dos requisitos é que a aplicação de *Mobile Banking* deve fornecer serviços bancários padrão, que forneçam uma experiência básica aos utilizadores. Outro requisito é que a aplicação deve incluir recursos de personalização que podem ser ativados e desativados pelos administradores do sistema, para realizarem testes empíricos de personalização. E a personalização deve ser alcançada através da realização de análises sobre o comportamento

de uso anterior para apresentar a UI e as mensagens personalizadas (Nawaz, Motiwalla, & Deokar, 2018).

A adaptação da interface da aplicação de *Mobile Banking* é feita usando uma técnica de análise de dados através da análise em tempo real das interações anteriores do utilizador. Conhecendo as interações anteriores, o sistema adapta-se ao comportamento do utilizador para melhorar a experiência do utilizador nos serviços de *Mobile Banking* (Nawaz, Motiwalla, & Deokar, 2018).

O sistema PERMA foi modelado com uma arquitetura cliente-servidor onde o smartphone liga-se a um servidor *backend* em *cloud* privada (Nawaz, Motiwalla, & Deokar, 2018) tal como ilustra a Figura 6.

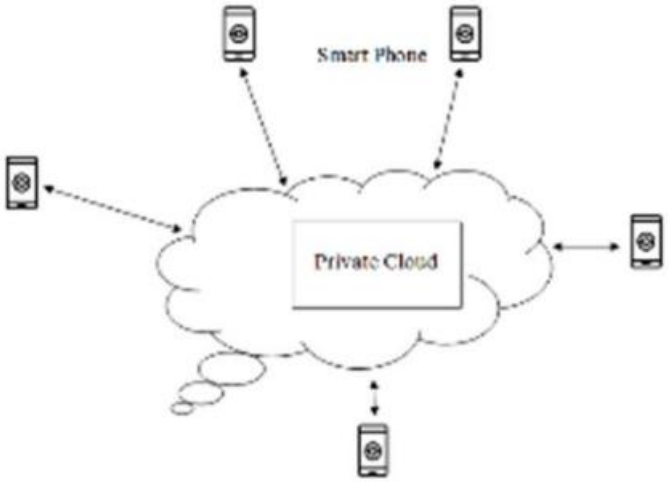

Figura 6 – Sistema PERMA (Nawaz, Motiwalla, & Deokar, 2018)

O sistema PERMA é composto por uma aplicação móvel híbrida, que usa AJAX para enviar e receber pedidos HTTP ao *backend*. O *backend* do sistema PERMA executa num servidor Apache Web Server que contêm a base de dados MySQL e suporta aplicações móveis (Nawaz, Motiwalla, & Deokar, 2018).

(Nawaz, Motiwalla, & Deokar, 2018) atribui vários benefícios a este sistema, entre os quais a melhoria da experiência do utilizador graças à aplicação de análise de dados na geração de uma UI adaptável que pode levar a que mais utilizadores de aplicações móveis acedam a serviços financeiros. A tecnologia híbrida que reduz o tempo de implementação devido à utilização de códigos comuns para Android e IOS, e também a *cloud* privada que aumenta a segurança (Nawaz, Motiwalla, & Deokar, 2018).

3 Análise de Valor

Recapitulando, o objetivo deste trabalho é desenvolver novas funcionalidades de fracionamento de prestações para as aplicações móveis de *Internet Banking* do banco cliente.

Serve este capítulo para aplicar algumas metodologias de análise de valor ao projeto no qual se baseia esta dissertação. São aplicadas as duas atividades iniciais do Fuzzy Front End, e é feita a proposta de valor do projeto. Também é aplicado o método AHP para avaliar os projetos de *Internet Banking* alternativos identificadas no capítulo 2.

3.1 Fuzzy Front End

O processo de inovação pode ser dividido em três áreas tal como ilustra a Figura 7 que são: o Fuzzy Front End (FFE), o desenvolvimento de novo produto (NPD) e a comercialização (Ajamian, et al., 2004).

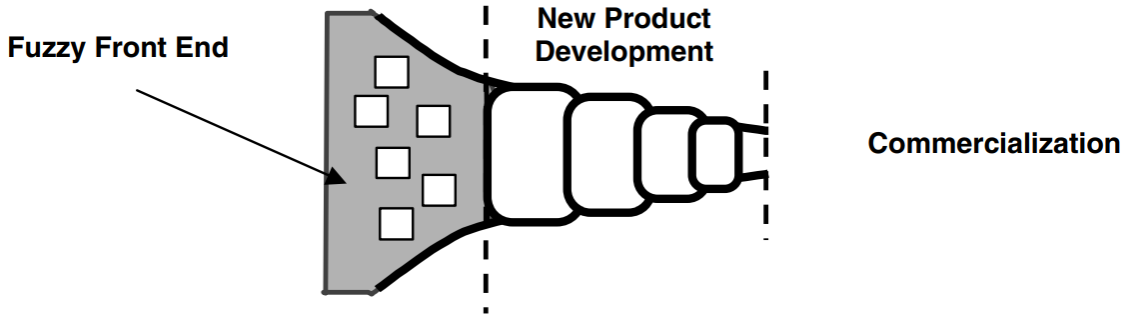

Figura 7 – Processo de inovação (Ajamian, et al., 2004)

O FFE é definido pelas atividades que antecedem o processo de NPD. As atividades no FFE são muitas vezes caóticas, imprevisíveis e não estruturadas. Em comparação o NPD é tipicamente um processo formal e bem estruturado com um conjunto prescrito de atividades e perguntas a serem respondidas (Ajamian, et al., 2004).

Existem muitas pesquisas sobre as melhores práticas de NPD, mas pesquisas semelhantes sobre as melhores praticas de FFE são ausentes (Ajamian, et al., 2004) (Koen, et al., 2001). Isto acontece porque muitas das práticas que auxiliam a parte do NPD não se aplicam ao FFE (Ajamian, et al., 2004). Devido à falta de pesquisas, a parte do FFE é apontada como uma das maiores oportunidades para melhoria do processo geral de inovação (Ajamian, et al., 2004). Assim, uma equipa de várias empresas do Industrial Institute Research tentou determinar coletivamente as melhores práticas do FFE de inovação (Koen, et al., 2001). No entanto esse

processo não trouxe resultados porque não havia uma linguagem comum para os principais elementos (Koen, et al., 2001). Como resultado, o grupo desenvolveu uma construção teórica, definida como o modelo de Desenvolvimento de Novo Conceito (NCD) a fim de fornecer uma linguagem comum (Koen, et al., 2001).

O modelo NCD ilustrado na Figura 8 é composto por três partes principais que são a parte do motor, a área do raio interno e os fatores de influência.

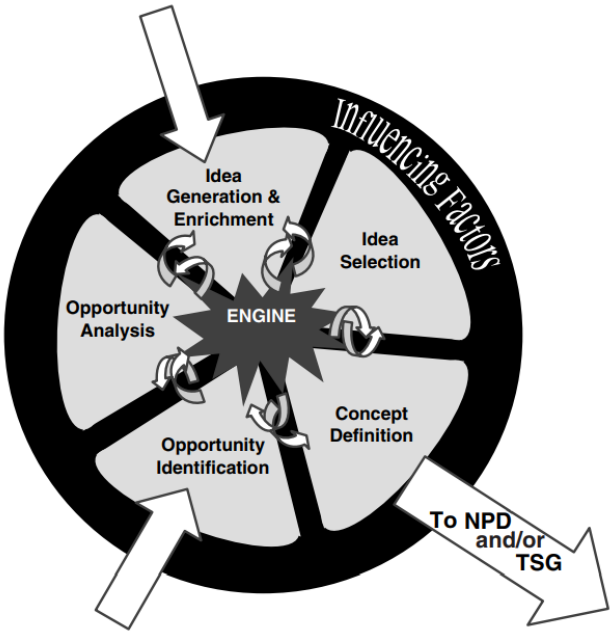

Figura 8 – Modelo NCD (Ajamian, et al., 2004)

A parte do motor (do inglês engine) ou alvo é a liderança, a cultura e a estratégia de negócios da organização que impulsiona os cinco elementos-chave que são controláveis pela organização (Ajamian, et al., 2004).

A área de raio interno define os cinco elementos controláveis do FFE que são a identificação de oportunidade, a análise de oportunidade, a geração e enriquecimento de ideias, a seleção de ideias e a definição de conceito (Ajamian, et al., 2004).

Os fatores de influência consistem em capacidades organizacionais, o mundo exterior (canais de distribuição, clientes, concorrentes) e as ciências capacitadoras que podem estar envolvidas. Esses fatores afetam todo o processo de inovação até a comercialização e são relativamente incontroláveis pela empresa (Ajamian, et al., 2004).

A forma circular do modelo NCD sugere que as ideias e conceitos devem iterar nos cinco elementos. As setas que apontam para o modelo representam pontos de partida e indicam que os projetos começam na identificação da oportunidade ou na geração e enriquecimento de ideias. A seta de saída representa como os conceitos saem do modelo e entram no processo NPD ou no processo *Technology Stage Gate* (TSG) (Ajamian, et al., 2004).

O modelo NCD com a sua terminologia e linguagem comuns deve permitir a melhor otimização das atividades do FFE (Ajamian, et al., 2004). Tal foi conseguido através da identificação de conjunto de métodos, tecnologias e técnicas para cada uma das atividades do FFE. Estas podem ser aplicadas a qualquer projeto, e neste documento foram aplicadas ao desenvolvimento de uma nova aplicação móvel de *Internet Banking* que permite fazer pagamentos em cartão. A análise de mercado e a análise das tendências tecnológicas podem ser aplicadas nos elementos Identificação de Oportunidade e Análise de Oportunidade do FFE.

Não foi possível encontrar dados quantitativos sobre os pagamentos fracionados em Portugal, mas foi possível encontrar informação sobre os pagamentos com cartão, como se apresentará mais à frente. Portanto, a análise de mercado que se fez parte do princípio que não existe aplicação móvel para o banco cliente. As duas próximas secções apresentam a pesquisa necessária para a decisão de desenvolver uma nova aplicação móvel de *Internet Banking*.

3.1.1 Identificação da Oportunidade

Neste elemento, a organização identifica oportunidades que pode querer perseguir (Ajamian, et al., 2004). Algumas das técnicas aplicadas nesta fase do FFE são a pesquisa de mercado e a análise da tendência dos consumidores e das tendências tecnológicas.

No caso de um projeto de *Internet Banking*, o banco antes de decidir investir no fornecimento de serviços bancários através de aplicação móvel teve de fazer determinadas pesquisas. Considera-se que se deve conhecer as estatísticas acerca da utilização de internet e das vendas de *smartphones* para se conhecer a viabilidade do projeto. E também da evolução do número de transações em cartão. O negócio apenas será viável se todas estas variáveis tiverem uma evolução crescente.

3.1.1.1 Evolução da utilização da internet

Tal como referido no capítulo 1, desde os anos 90 a utilização da Internet tem tido uma tendência crescente o que possibilita que as empresas desenvolvam serviços com base em Internet. O gráfico da Figura 9 apresenta a percentagem de população que utiliza a internet em Portugal e na União Europeia (UE) desde 1990 e 2019. No gráfico são considerados utilizadores da Internet pessoas que utilizaram a Internet nos últimos três meses. A Internet pode ser usada através de um computador, telemóvel, assistente pessoal digital, TV digital, entre outros (Roser, Ritchie, & Ortiz-Ospina, 2015).

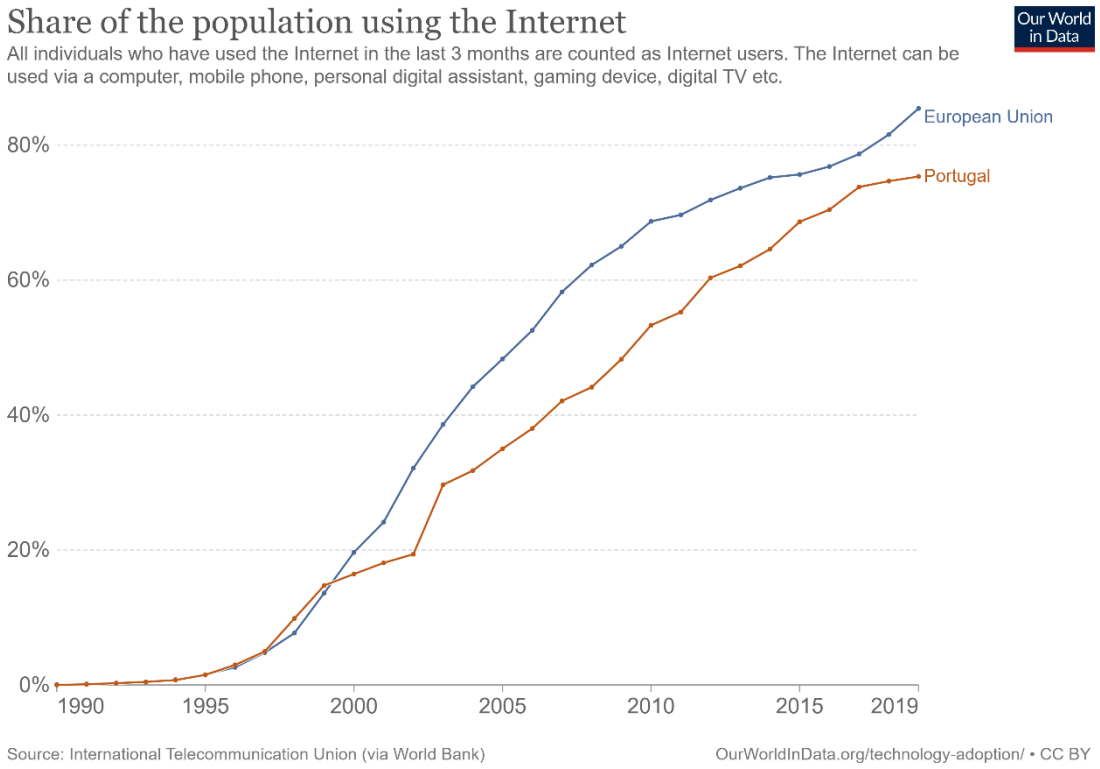

Figura 9 – Percentagem de população que utiliza internet (Roser, Ritchie, & Ortiz-Ospina, 2015)

Como se observa na Figura 9, o crescimento da percentagem de população na UE que usa a Internet é exponencial. Esse crescimento na UE é também acompanhado por Portugal.

Tal como referido na introdução, segundo (Roser, Ritchie, & Ortiz-Ospina, 2015) no ano 2000 16,43% da população portuguesa usava Internet, e no ano de 2005 a percentagem aumentou para 34,99%. E os dados de (Miniwatts Marketing Group, 2021) estimam que, em 31 de Junho de 2020, 78,2% da população portuguesa utilizava a Internet. Os dados de (Miniwatts Marketing Group, 2021) também estimam que, até 31 de março de 2021, 65,6% da população mundial tem acesso à Internet. Assim confirma-se a evolução crescente do uso da Internet não só em Portugal, mas também em todo o mundo.

3.1.1.2 Evolução das vendas de *smartphones*

No que respeita à utilização de *smartphones*, estes também seguiram a mesma tendência de crescimento. O gráfico da Figura 10 apresenta a evolução do número de *smartphones*, em mil milhões de unidades, vendidos a utilizadores finais em todo o mundo de 2007 a 2021.

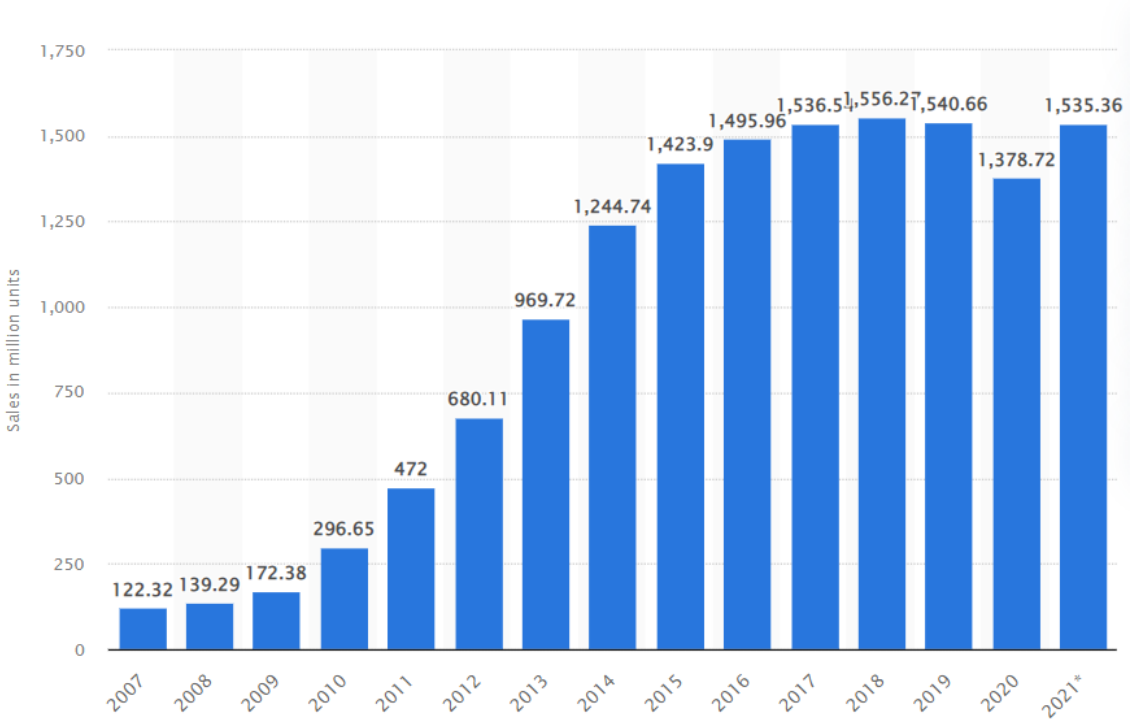

Figura 10 - Número de *smartphones* vendidos a utilizadores finais em todo o mundo de 2007 a 2021 (Statista, 2021)

Menos da metade da população mundial possuía um *smartphone* em 2016, mas a taxa de penetração destes dispositivos continuou a subir chegando a 78,05% em 2020 (Statista, 2021).

Em 2020 os fornecedores de *smartphones* venderam cerca de 1,38 mil milhões de smartphones em todo o mundo (Statista, 2021). As vendas globais destes dispositivos móveis devem aumentar de 2020 a 2021 em todas as principais regiões, devido à recuperação do mercado do impacto inicial da pandemia (Statista, 2021). Em 2021 o número previsto de vendas é de cerca de 1,535 milhões de smartphones. O investimento em aplicações móveis é possível porque existe um grande número de vendas de smartphones que possam instalar essas aplicações.

3.1.1.3 Evolução dos pagamentos em cartão

Como o projeto *Installments* insere-se na área dos pagamentos em cartão, foi pesquisada a evolução de número de pagamentos em cartão na EU. E também se obteve a evolução de vários serviços de pagamento. O número total de pagamentos não em numerário na zona euro aumentou em 2020 em 3,7% para 101,6 mil milhões em comparação com o ano anterior (European Central Bank, 2021). Os pagamentos com cartão representaram 47% das transações (European Central Bank, 2021). Segundo o (European Central Bank, 2021) em Portugal, o número de pagamentos em cartão corresponde a 70% do número de pagamentos não em numerário.

O número de cartões na zona euro com função de pagamento aumentou em 2020 em 6,5% para 609,3 milhões. Com uma população total na zona euro de 343 milhões habitantes, isto representou cerca de 1,8 cartões de pagamento por habitante da zona euro. O gráfico da Figura 11 mostra a evolução da utilização dos principais serviços de pagamento na zona euro de 2000 a 2020 (European Central Bank, 2021).

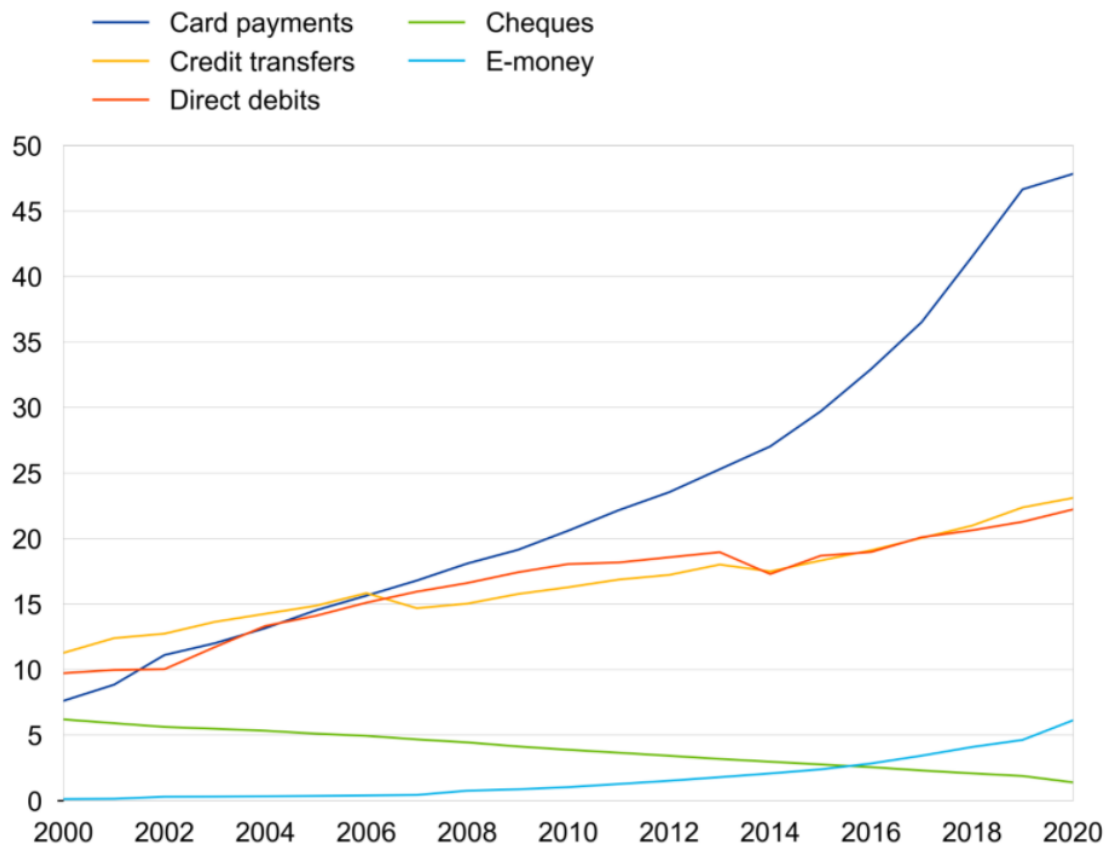

Figura 11 – Utilização dos principais serviços de pagamento na zona euro (European Central Bank, 2021)

Tal como a Figura 11 indica, os pagamentos em cartão foram os que mais aumentaram. Isto permite o investimento dos bancos em novos canais que possibilitem este tipo de pagamento, como por exemplo o canal da *Internet Banking*.

A evolução das tendências dos consumidores impulsionou a evolução tecnológica na área da banca. Por exemplo, o crescimento da utilização Internet desde a década de 90 impulsionou o desenvolvimento da *Internet Banking*, assim como o crescimento do uso dos smartphones desde 2007 gerou o aumento de bancos com serviço de *Mobile Banking*.

Conhecendo as tendências crescentes dos números de utilizadores da Internet, nas vendas de *smartphones* e dos pagamentos em cartão é possível fazer a identificação a oportunidade.

Assim, a oportunidade é o desenvolvimento de uma nova aplicação móvel de *Internet Banking*, que possibilita a realização de pagamentos em cartão.

3.1.2 Análise de Oportunidade

Neste elemento, a oportunidade é avaliada para confirmar que vale a pena segui-la. Isto envolve fazer mais avaliações de mercado e tecnológicas. Muitos dos métodos, ferramentas e técnicas a aplicadas na identificação de oportunidade são idênticas aos utilizados neste elemento (Ajamian, et al., 2004).

Após ser feita a identificação da oportunidade de criar uma aplicação móvel para realizar as operações bancárias, deve ser analisado o mercado para os canais digitais (*Internet Banking* e *Mobile Banking*) dos bancos. Tal como na secção 3.1.1, será feita uma análise das tendências dos consumidores, mas neste caso o foco é a utilização de serviços de *Internet Banking* e *Mobile Banking*.

A *Internet Banking* foi introduzida em Portugal em 1998 pelo Banco Espírito Santo, e desde então tem vindo a crescer. A Figura 12 apresenta a evolução da taxa de penetração da *Internet Banking* em Portugal desde 2003.

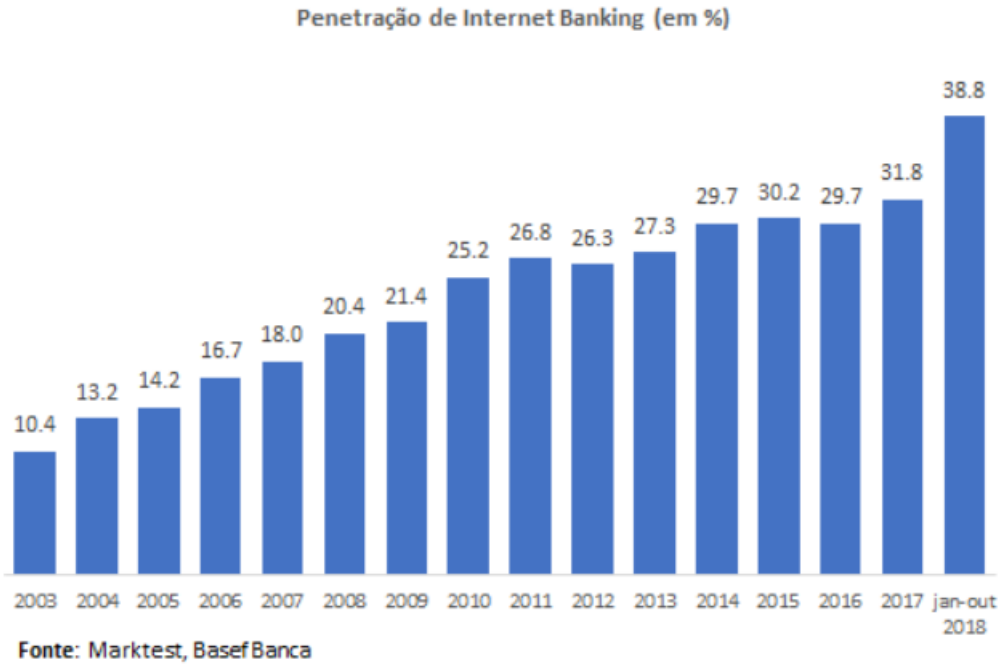

Figura 12 – Taxa de penetração da *Internet Banking* em Portugal (Grupo Marktest, 2019)

A taxa de penetração da *Internet Banking* tem crescido desde 2003 quando 10,4% dos portugueses referia possuir o serviço (Grupo Marktest, 2019). No período de janeiro a outubro de 2018 a percentagem subiu para 38.8% (Grupo Marktest, 2019).

Em 2016, a *Internet Banking* já tinha chegado a 2,5 milhões de portugueses segundo dados da Marktest (Morais, 2017). A Figura 13 apresenta a evolução da *Internet Banking* em Portugal de 2012 a 2016 e o perfil dos utilizadores.

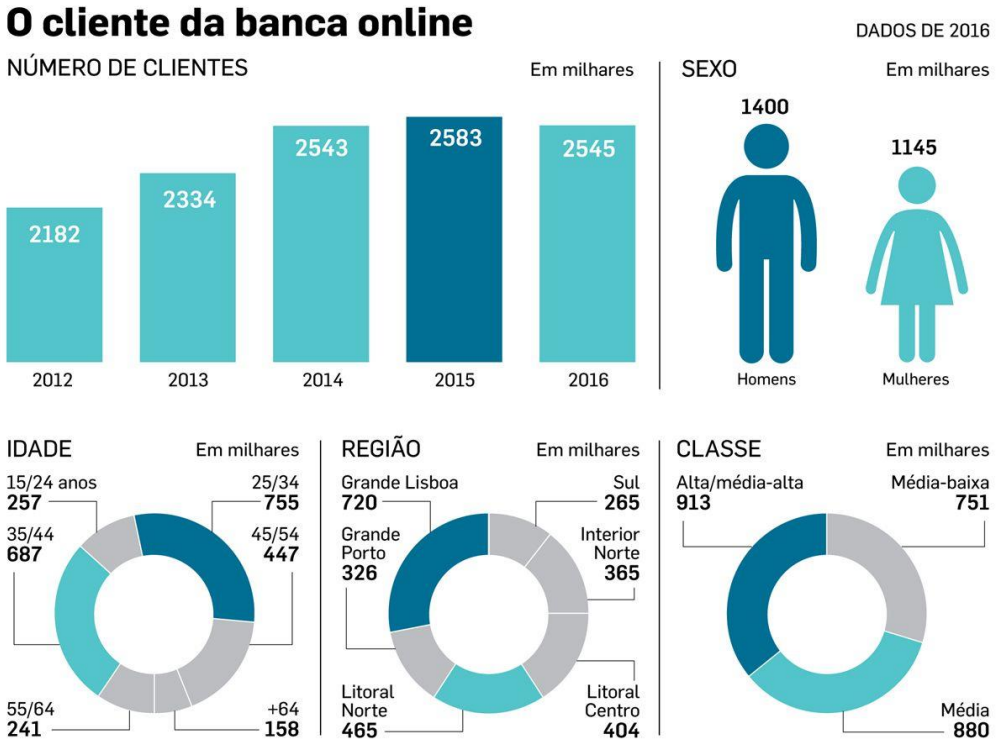

Figura 13 – Número de clientes da *Internet Banking* (Morais, 2017)

O estudo da Marktet para 2016 revela que são mais homens do que mulheres e maioritariamente entre os 25 e os 34 anos que utilizaram mais os serviços de *Internet Banking* em 2016 (Morais, 2017). Os residentes na Grande Lisboa e de classe alta e média-alta também são os que mais usam *Internet Banking* (Morais, 2017).

A tendência de crescimento também se verifica para *Mobile Banking*. Os resultados de um estudo da Marktest para 2019 indicam que 2,675 milhões de portugueses utilizam o serviço de *Mobile Banking* (Grupo Marktest, 2020). Este número corresponde a 33.7% dos indivíduos possuem conta bancária (Grupo Marktest, 2020). Os homens registam uma taxa de penetração deste serviço superior à das mulheres, enquanto os jovens dos 25 aos 34 anos e os indivíduos das classes sociais mais elevadas são os grupos com maior afinidade para a *Mobile Banking* (Grupo Marktest, 2020).

Em 2020 foi feito outro estudo pela Marktest que indica que os canais digitais são usados por 4,71 milhões de portugueses. Este número corresponde a 58,9% dos portugueses que tem conta bancária (Grupo Marktest, 2021). Notando-se aqui um aumento relativamente a 2016. O perfil dos consumidores que mais indicam usar canais digitais é idêntico ao de 2016.

Após esta fase, fica-se a conhecer quantos portugueses utilizam os canais digitais para realizar as suas operações bancárias, e qual o perfil dos portugueses que mais utilizam esses mesmos canais.

3.2 Proposta de Valor

Para realizar a proposta de valor para o projeto *Installments* começa-se por definir valor para o cliente e como este é atribuído a este projeto. Em seguida é utilizada a ferramenta Proposta de Valor Canvas para construir a proposta de valor para este projeto.

3.2.1 Valor para o cliente

O valor pode ter diferentes designações atribuídas por diferentes autores. As designações mais utilizadas são valor do cliente, valor percebido e valor (Woodall, 2003). (Woodall, 2003) designa-o de valor para o cliente (VC). Segundo este autor, VC é qualquer perceção pessoal do cliente acerca da vantagem da oferta de uma organização, e pode ocorrer como redução do sacrifício, presença de benefício ou qualquer combinação do sacrifício e benefício (Woodall, 2003).

Para a solução *Installments*, um exemplo de sacrifício que esta pretende colmatar é a deslocação dos clientes às sucursais para realizar o pagamento em prestações. Um exemplo de benefício é a redução de custos. A redução do sacrifício e o benefício referidos são exemplos de VC, que a solução *Installments* entrega aos utilizadores das aplicações do banco cliente.

(Woodall, 2003) dividiu as diferentes definições de VC encontradas na literatura em noções distintas. A única que se irá abordar aqui é a que define VC como um balanço entre benefícios e sacrifícios (Woodall, 2003). Portanto, quanto maiores forem os benefícios de um determinado produto maior será o seu valor, e quanto maiores forem os seus sacrifícios menores será o valor. Uma aplicação de *Mobile Banking*, poderá ter inconvenientes como ecrãs pequenos, entradas inconvenientes e respostas lentas que dificultam a sua adoção e utilização (Nawaz, Motiwalla, & Deokar, 2018). Mas estes são compensados por benefícios como comodidade, acessibilidade e conveniência que aumentam o VC.

3.2.2 Proposta de Valor Canvas

A proposta de valor é "a definição de como itens de valor, como produtos e serviços, bem como serviços complementares de valor agregado, são empacotados e oferecidos para atender às necessidades do cliente" [2] (Osterwalder & Pigneur, 2003).

A proposta de valor é parte integrante do modelo de negócio (Osterwalder & Pigneur, 2003). Segundo os autores Osterwalder e Pigneur, um modelo de negócio pode ser dividido em diferentes pilares. Esses pilares respondem a questões como: o que a empresa oferece; quem são os seus clientes alvo; como o produto será lançado; e quanto será ganho com isso (Osterwalder & Pigneur, 2003).

Para construir a proposta de valor será usada uma ferramenta denominada de Proposta de Valor Canvas (do inglês *Canvas Value Proposition*). Esta permite visualizar as diferentes ofertas num grafo e visualizar a posição competitiva da empresa (Osterwalder & Pigneur, 2003).

No sentido de garantir que o produto da empresa está posicionado em torno dos valores dos clientes, é criado um ajuste entre o produto e o mercado (Pereira, What is the Value Proposition Canvas?, 2021). Para que isso aconteça são explorados ao detalhe os blocos Segmento de Mercado e Proposta de Valor do Modelo de Negócio Canvas (Pereira, What is the Value Proposition Canvas?, 2021). A Figura 14 ilustra a Proposta de Valor Canvas.

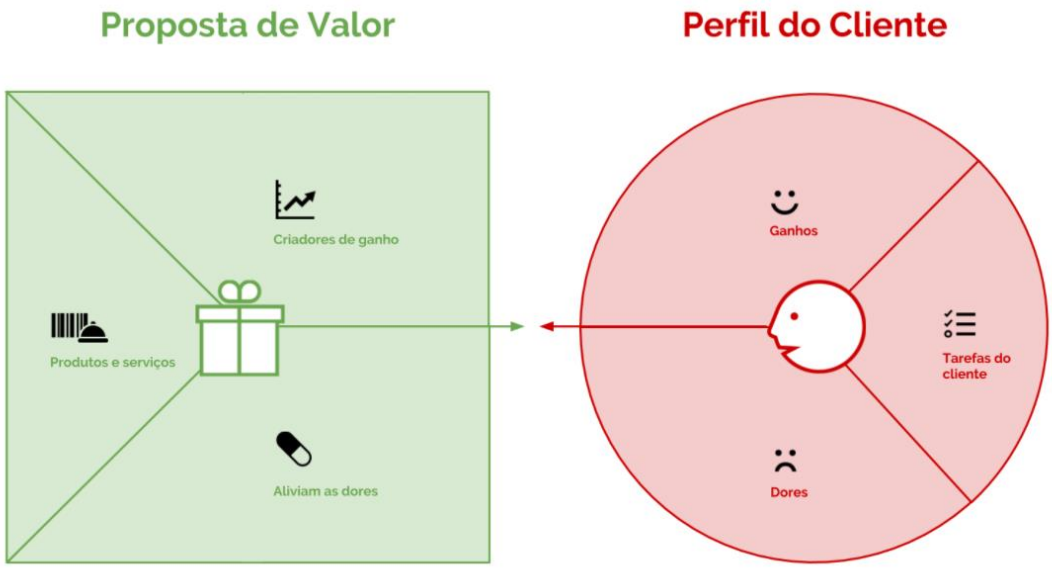

Figura 14 – Proposta de Valor Canvas (Pereira, Canvas da Proposta de Valor, 2019)

[2] Tradução livre dos autores. No original "[...] definition of how items of value, such as products and services as well as complementary value-added services, are packaged and offered to fulfil customer needs."

Como se nota na Figura 14, a Proposta de Valor Canvas está dividida em dois lados, que correspondem aos dois blocos do Modelo de Negócios Canvas referidos (Pereira, 2021).

O lado direito corresponde ao Perfil do Cliente que por sua vez está dividido em tarefas do cliente, ganhos e dores (Pereira, What is the Value Proposition Canvas?, 2021). As tarefas do cliente são as tarefas que o cliente faz, os problemas que resolve ou as necessidades que satisfaz (Pereira, What is the Value Proposition Canvas?, 2021). As dores são as experiências negativas que o cliente experiência quando realiza as tarefas do cliente (Pereira, What is the Value Proposition Canvas?, 2021) (B2B International, 2022). Os ganhos correspondem aos benefícios que o cliente espera ou deseja ao realizar as tarefas do cliente, e que trazem um aumento da sua satisfação (Pereira, What is the Value Proposition Canvas?, 2021).

O lado da Proposta de Valor contem os produtos e serviços, os criadores de ganho e os analgésicos (aliviam as dores) (Pereira, What is the Value Proposition Canvas?, 2021). Os "produtos e serviços" incluem todos os produtos ou serviços que serão entregues ao cliente (Pereira, What is the Value Proposition Canvas?, 2021). Estes aliviam as dores, criam ganhos e criam valor para o cliente (B2B International, 2022). Os "criadores de ganhos" indicam como o produto/serviço oferece valor ao cliente, os benefícios e se as expectativas do cliente são alcançadas (Pereira, What is the Value Proposition Canvas?, 2021). Os analgésicos descrevem como o produto/serviço irá aliviar as dores do cliente (Pereira, What is the Value Proposition Canvas?, 2021) (B2B International, 2022).

No sentido de alcançar um ajuste entre a proposta de valor e o perfil do cliente, os produtos devem corresponder às dores e ganhos do perfil do cliente (B2B International, 2022). Ou seja, os criadores de ganhos da proposta de valor devem ajustar-se aos ganhos definidos no perfil de cliente. Os analgésicos da proposta de valor também devem ajustar-se às dores do perfil do cliente.

A Figura 15 apresenta a Proposta de Valor Canvas para o projeto *Installments*.

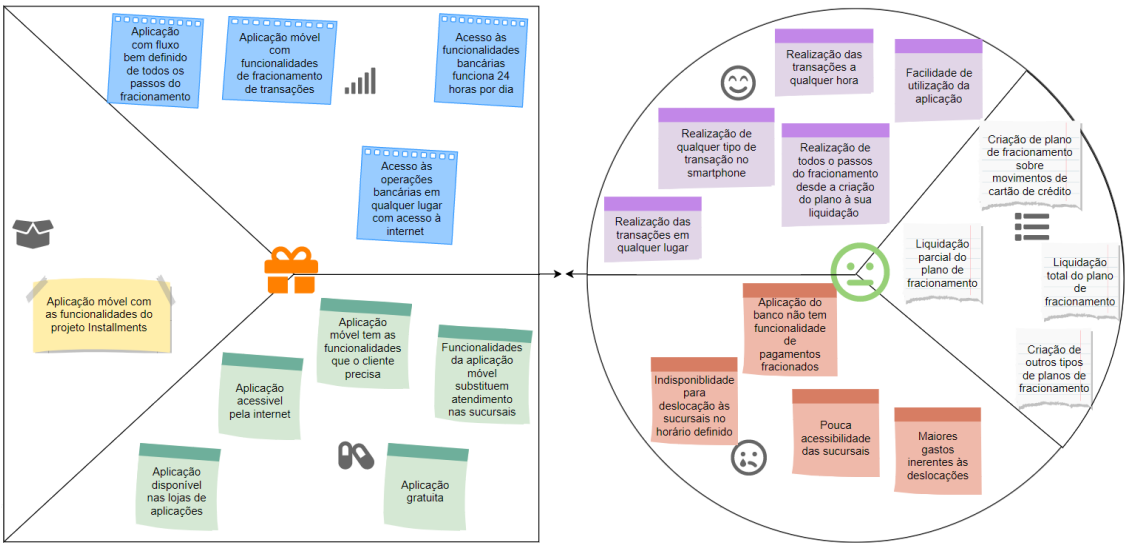

Figura 15 – Proposta de Valor Canvas para o projeto *Installments*

As tarefas do cliente referidas na Figura 15 são as fases do projeto *Intallments* referidas na secção 1.2 que serão necessárias nos fracionamentos de pagamentos. Como é referido na Figura 15, as dores do cliente que este projeto irá colmatar são a ausência das funcionalidades requeridas na aplicação do banco. Outras dificuldades podem ser a indisponibilidade, os maiores gastos nas deslocações, ou a pouca acessibilidade dos centros de atendimento. Alguns dos desejos que o cliente espera ver satisfeitos com as novas funcionalidades são a realização de fracionamentos no seu *smartphone* em qualquer hora e em qualquer lugar. O que o cliente também deseja é que seja possível realizar todas passos do fracionamento na aplicação. A facilidade da utilização da aplicação é também outro requisito importante.

O serviço a oferecer ao cliente é a aplicação móvel bancária com as funcionalidades definidas para projeto *Installments*. Os criadores de ganhos e os analgésicos foram definidos de modo a ajustarem-se aos ganhos e dores, respetivamente. Por motivos de simplicidade, optou-se por não representar os ajustes entre a proposta de valor e o perfil do cliente na Figura 15. Mas estes são descritos nos dois parágrafos que se seguem.

Para a dor “aplicação não tem a funcionalidade de pagamentos fracionados” é atribuído o analgésico de “aplicação tem as funcionalidades que o cliente precisa”, que neste caso é este tipo de plano de pagamento. Isto contribui para a boa relação entre o banco e o cliente, porque este projeto adiciona uma nova funcionalidade às que já existem na aplicação móvel.

Para a dor da “indisponibilidade de deslocação às sucursais no horário definido” é oferecido o analgésico “funcionalidades da aplicação móvel substituem atendimento nas sucursais”. A aplicação móvel também é disponibilizada em lojas de aplicações Android e IOS e é acessível pela internet. A aplicação móvel dá a possibilidade de se realizar as mesmas operações financeiras que se faria presencialmente. Para a dor da “pouca acessibilidade das sucursais” os analgésicos são os mesmos que os aplicados à dor da indisponibilidade das deslocações.

Todas as inconveniências da deslocação a um posto de atendimento presencial são mitigadas pela oferta da aplicação móvel.

No que toca à dor dos "maiores gastos com as deslocações" esta é resolvida com a "aplicação gratuita". A aplicação móvel permite reduzir custos com deslocações e é gratuita.

O ganho da "realização de qualquer tipo de transação no *smartphone*" corresponde ao criador de ganho "aplicação móvel com funcionalidades de fracionamento de transações" para o serviço oferecido. E também é oferecido pelo criador de ganho da "aplicação com fluxo bem definido de todos os passos do fracionamento". Esses fracionamentos serão feitos pelo cliente no seu *smartphone*, tal como realiza as outras operações bancárias.

Para o ganho da "realização das transações em qualquer lugar", o banco oferece o criador de ganhos que é uma aplicação com as "acesso às operações bancárias em qualquer lugar com acesso à internet". O ganho da "realização das transações a qualquer hora" é criado pela aplicação com "acesso às funcionalidades bancárias 24 horas por dia". Estes são dois dos maiores desejos que o cliente pretende com uma aplicação bancária.

O benefício da "realização de todos os passos do fracionamento desde a criação do plano à sua liquidação" é fornecido pela "Aplicação com fluxo bem definido de todos os passos do fracionamento". O ganho referido também é criado pela "aplicação móvel com funcionalidades de fracionamento de transações". Isto permite que o cliente possa acompanhar todos os passos do processo de fracionamento sem sair de casa.

O ganho "facilidade de utilização da aplicação" é concretizado pela "aplicação com fluxo bem definido de todos os passos do fracionamento". Dado que o fluxo de tarefas é organizado e de fácil compreensão, também contribui para a boa experiência de utilização.

3.3 Processo de Hierarquia Analítica (AHP)

O AHP é um método de tomada de decisão multicritério desenvolvido pelo autor Thomas L. Saaty que permite a tomada de decisão com base em critérios e alternativas.

Este processo está dividido em três partes que são: identificar e organizar os objetivos de decisão, critérios, restrições e alternativas em hierarquia; realizar comparações par a par entre os elementos relevantes de cada nível da hierarquia; e a síntese usando o algoritmo que utiliza os resultados das comparações par a par em todos os níveis. Finalmente, obtém-se o resultado do algoritmo que dá a importância relativa de cursos alternativos de ação (Saaty, 1988).

A comparação entre os diferentes critérios e as diferentes alternativas pode ser apresentada numa matriz quadrada cujos elementos representam uma comparação par a par entre dois

critérios ou duas alternativas. Os valores atribuídos a cada um dos elementos obedece à escala fundamental dos valores absolutos e que está representada na Tabela 1.

Tabela 1 – Escala fundamental dos valores absolutos (Saaty R. W., 1987)

Intensidade de importância	**Definição**	**Explicação**
1	Igual importância	Duas atividades contribuem igualmente para o objetivo
3	Importância moderada de um sobre outro	A experiência e o julgamento favorecem moderadamente uma atividade sobre a outra
5	Importância essencial ou forte	A experiência e o julgamento favorecem fortemente uma atividade sobre outra
7	Importância muito forte	Uma atividade é fortemente favorecida e seu domínio demonstrado na prática
9	Extrema importância	A evidência que favorece uma atividade sobre outra é da mais alta ordem de afirmação possível
2,4,6,8	Valores intermédios entre julgamentos adjacentes	
Recíprocos	Se a atividade i tem um dos números acima atribuídos a ela quando comparada com a atividade j, então j tem o valor recíproco quando comparada com i	
Racionais	Proporções decorrentes da escala	Se a consistência for forçada pela obtenção de n valores numéricos para abranger a matriz

A Tabela 1 indica a escala na qual deve ser indicado o número de vezes que um elemento é mais importante do que outro relativamente a um determinado critério ou atributo (Saaty T. L., 2004).

A matriz referida é uma matriz de julgamentos de ordem n que pode ser representada por $A=[a_{ij}]$, sendo a_{ij} a importância da alternativa i sobre a alternativa j. Os valores recíprocos são representados por $a_{ji}=1/a_{ij}$ sendo a_{ji} a importância da alternativa i sobre a alternativa j. As importâncias apresentadas na matriz devem ser consistentes. E para que tal aconteça devem

obedecer à equação $a_{ij}a_{jk}=a_{ik}$, onde i, j e k representam números inteiros compreendidos entre 1 e n. Não é necessário conhecer todos os julgamentos porque alguns dos elementos da matriz de julgamentos podem ser induzidos pela mesma equação.

Para conhecer os pesos de cada critério na matriz de julgamentos é necessário normalizar a matriz. A normalização é feita dividindo todos os elementos da matriz pela soma dos elementos da respetiva coluna. Os pesos de cada critério são calculados pela média da cada linha da matriz normalizada. Assim, é possível obter o vetor das prioridades que identifica a ordem de importância de cada citério.

O vetor de prioridades é o vetor próprio da matriz dos julgamentos. Conhecendo o vetor próprio W da matriz A, é pode-se calcular o seu valor próprio. A equação $AW= \lambda_{max}W$ é a relação entre as matrizes A e W e o valor λ_{max}, que é o maior valor próprio que a matriz A pode ter (Saaty R. W., 1987). A partir dessa equação é possível induzir a equação 1 para calcular o valor λ_{max} recorrendo a valores escalares.

$$\sum_{j=1}^{n} a_{ij}w_j = \lambda_{max}w_i \quad (1)$$

A equação 1 é constituída por a_{ij} que é o valor da importância localizado na linha i e na coluna j da matriz A de ordem n, por w_j que é o valor do elemento da linha j da matriz W, que tem uma coluna, e por w_i é o elemento da linha i da matriz W. É de notar que são calculados tantos valores λ_{max} quanto o número de linhas da matriz A. O valor resultante de λ_{max} é a média desses valores.

Considerando que a matriz dos julgamentos é positiva, o valor próprio máximo λ_{max} da matriz é maior ou igual à sua ordem n, mas é igual se a matriz dos julgamentos for consistente (Saaty T. L., 2004). Devido à igualdade entre λ_{max} e n em caso de consistência da matriz de julgamentos, o valor da ordem da matriz pode ser utilizado no cálculo do índice de consistência (IC) da matriz. O IC da matriz é calculado recorrendo à equação 2.

$$IC = \frac{\lambda_{max} - n}{n-1}, \lambda_{max} \geq n \quad (2)$$

Analisando a equação 2, verifica-se que se o valor λ_{max} for igual à ordem da matriz, o IC é zero, portanto a matriz tem o máximo de consistência.

O índice aleatório é a média de vários cálculos de índices de consistência para matrizes com uma determinada ordem compreendida entre 1 e 10. A Tabela 2 mostra os índices aleatórios IA correspondentes a matrizes de ordem n.

Tabela 2 – Índices aleatórios (Saaty T. L., 2004)

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

IA	0	0	0,52	0,89	1,11	1,25	1,35	1,40	1,45	1,49

O rácio de consistência (RC) é a razão entre o índice de consistência e o índice aleatório correspondente à ordem da matriz dos julgamentos. Uma percentagem de inconsistência de 10% é considerada aceitável (Saaty T. L., 2004). Ou seja, se o RC for inferior a 0,1 a matriz de julgamentos analisada é considerada consistente, se o RC for superior a esse valor a matriz é inconsistente.

Tal como referido na introdução, uma restrição da implementação para este projeto é a utilização da arquitetura de microsserviços. E os serviços devem ser desenvolvidos em .NET com o auxílio de uma *framework* do banco cliente. Mas existem outros projetos de *Internet Banking* documentadas na literatura que apresentam alternativas arquiteturais que poderiam ser adotadas neste projeto.

Este projeto em particular deve obedecer a critérios como manutenibilidade e testabilidade. No entanto o método AHP necessita de mais do que dois critérios. E existem outros atributos de qualidade que também contribuem para a satisfação do utilizador. O objetivo de decisão é selecionar qual dos sistemas de *Internet Banking* apresentados neste documento contribui melhor para satisfação do utilizador.

3.3.1 Justificação dos critérios

Os critérios adotados neste processo de tomada de decisão são a testabilidade, a segurança, a usabilidade e o desempenho. Estes critérios foram identificados na pesquisa sobre os critérios de qualidade que melhor contribuem para a satisfação dos utilizadores de serviços de *Internet Banking*. Na literatura, foram identificados diversos critérios de qualidade, os mais comuns são a facilidade de utilização da aplicação e a privacidade/segurança. E também foram identificados critérios que podem ajustar-se ao atributo de qualidade desempenho.

Selecionou-se o estudo conduzido por (Sakhaei, Afshari, & Esmaili, 2014) que identificou seis medidas de qualidade que podem conduzir à satisfação do cliente. As medidas de qualidade são: a confiabilidade da informação fornecida e o bom funcionamento técnico; a eficiência ou capacidade para o cliente chegar ao seu produto desejado; a responsividade dos retalhistas para fornecer o apoio necessário; o cumprimento da entrega do produto no tempo prometido; a segurança e a privacidade da informação pessoal; e o apelo visual do Website.

A variável confiabilidade tal como é definida pelos autores pode ser influenciada por vários atributos de qualidade tais como o desempenho e a fiabilidade. Mas neste caso associou-se a mesma ao atributo de qualidade testabilidade, porque uma aplicação que é bem testada terá provavelmente um bom funcionamento técnico, e a informação que fornece será mais precisa. A eficiência pode ser influenciada pelo atributo de qualidade usabilidade. Porque, ter um bom fluxo na aplicação/*site* ou um reduzido número de cliques, contribui para a boa experiência do

utilizador. O fator da responsividade, não foi considerado porque não se relaciona com o funcionamento da aplicação. O fator do cumprimento pode ser ajustado ao atributo de qualidade desempenho, porque avalia o tempo de resposta para obter um determinado produto bancário. O *design* do *Website* é avaliado pelo atributo usabilidade, e a segurança é outro atributo de qualidade.

No final do estudo foi obtido a classificação de todos os fatores referidos quanto à sua influência para a satisfação do utilizador. O fator confiabilidade é o que influencia mais a satisfação do consumidor, e o *design* do *Website* é o fator que influencia menos (Sakhaei, Afshari, & Esmaili, 2014).

Os valores das classificações foram utilizados para calcular as importâncias entre os diferentes critérios do AHP, que neste caso são atributos de qualidade que classificam um sistema de *software*. A usabilidade que é influenciada por dois fatores é a soma das classificações desses dois fatores. Os valores dos restantes critérios correspondem às classificações dos respetivos fatores de qualidade. A Tabela 3 apresenta as classificações atribuídas a cada um dos critérios de qualidade tendo em conta os resultados do estudo de (Sakhaei, Afshari, & Esmaili, 2014).

Tabela 3 – Classificação dos critérios

Critério	**Classificação**
Usabilidade	6,62
Testabilidade	4,05
Desempenho	3,75
Segurança	3,48

A comparação para a par das importâncias dos critérios é feita para todos os critérios, através da divisão entre as classificações de todos os pares possíveis de critérios. Esta divisão indica quantas vezes um critério é mais importante do que outro, obedecendo à escala fundamental dos valores absolutos. A Tabela 4 apresenta as comparações para a par dos diferentes critérios.

Tabela 4 – Comparação par a par entre critérios

	Usabilidade	**Testabilidade**	**Desempenho**	**Segurança**
Usabilidade	1	1	2	2
Testabilidade	0,5	1	1	1
Desempenho	0,5	1	1	1
Segurança	0,5	1	1	1

3.3.2 Justificação das alternativas

A soluções de *software* avaliadas pelos critérios são a solução adotada neste projeto e as apresentadas no estado de arte. Cada um dos projetos do estado de arte apresenta uma alternativa arquitetural para o projeto *Installments*. E cada uma das soluções apresenta mais ou menos preocupações quanto à experiência do utilizador.

Alguns autores não fazem referência à forma como contribuíram para a experiência do utilizador nos seus projetos, por isso assume-se que o critério da usabilidade tem uma classificação mínima nestes projetos. Os restantes critérios serão avaliados tendo em conta os estilos arquiteturais adotados para cada um dos projetos. Todos os estilos arquiteturais favorecem os atributos de qualidade do *software* de maneira diferente. E por isso, este tem classificações diferentes nas soluções alternativas adotadas. Não foram tidas em conta as tecnologias aplicadas.

Como a análise AHP é um método quantitativo, considera-se que todas as soluções alternativas têm o mesmo conjunto de funcionalidades, portanto tem o mesmo grau de complexidade. E assume-se que todos os sistemas têm a mesma quantidade de utilizadores, de forma a não ter influência no desempenho. Deste modo, o peso atribuído a cada atributo de qualidade, exceto a usabilidade, em cada solução depende exclusivamente dos estilos arquiteturais aplicados.

As soluções alternativas que se consideraram na análise AHP são a Solução A, a Solução B, a Solução C e a Solução D, que são explicadas de seguida.

A Solução A é o próprio sistema *Installments*. Este sistema apresenta uma arquitetura de microsserviços e um estilo arquitetural *N-tier*, por ter pelo menos quatro níveis. Neste projeto também existe preocupação com o fluxo das várias etapas do fracionamento de forma a melhorar a usabilidade.

A Solução B é a aplicada no projeto proposto por (Appandairaj & Murugappan, 2013). Este projeto integra vários canais de comunicação e combina o estilo dos barramentos de mensagens com o estilo SOA. Os autores deste projeto não apresentam qualquer referência à experiência do utilizador. Se o projeto *Installments* combinasse estes dois estilos arquiteturais, a camada de integração estaria entre a aplicação móvel e um serviço. Essa camada de integração teria mesma função que o projeto *Installments* API, referido em 2.3. Esse serviço combinaria as funcionalidades de *Installments Experience* e *Installments Process*, e comunicaria com o Sistema Central. Esta solução tem quatro níveis/camadas físicas

Para a Solução C decidiu-se juntar dois projetos do estado arte que apresentam estilos arquiteturais semelhantes. Estas são a solução da Wells Fargo e a solução Duke's Bank. Ambos os sistemas têm o estilo arquitetural *3-tier*. Se o projeto *Installments* também tivesse 3 níveis

seria composto pela aplicação móvel, por um serviço que implementaria toda a lógica do negócio para os fracionamentos, e pelo sistema central do banco.

A Solução D é baseada no projeto PERMA. Este projeto possui uma arquitetura cliente/servidor, e possui uma interface personalizada para cada utilizador. Se esta fosse aplicada no projeto *Installments*, a aplicação móvel ligaria diretamente ao sistema bancário do cliente.

Segue-se a avaliação das quatro soluções apresentadas tendo em conta cada um dos quatro critérios definidos anteriormente, no sentido de encontrar a melhor solução.

Quanto ao critério da usabilidade, a matriz de comparação par a par entre todas as alternativas tecnológicas está representada na Tabela 5

Tabela 5 – Comparação par a par entre alternativas para critério usabilidade

	Solução A	**Solução B**	**Solução C**	**Solução D**
Solução A	1	3/2	3/2	1/2
Solução B	2/3	1	1	1/3
Solução C	2/3	1	1	1/3
Solução D	2	3	3	1

Quanto ao critério usabilidade, considera-se que a Solução D fornece a melhor experiência do utilizador por permitir definir um fluxo de interações tendo em conta as preferências do utilizador. A Solução A é a que fornece a segunda melhor usabilidade por ter um fluxo estruturado de tarefas para desempenhar uma operação bancária. Mas esse fluxo não é adaptável às preferências do cada utilizador.

Quanto ao critério da testabilidade, a matriz de comparação par a par entre todas as alternativas tecnológicas está representada na Tabela 6.

Tabela 6 - Comparação par a par entre alternativas para critério testabilidade

	Solução A	**Solução B**	**Solução C**	**Solução D**
Solução A	1	4/3	4/3	2
Solução B	3/4	1	1	3/2
Solução C	3/4	1	1	3/2
Solução D	1/2	2/3	2/3	1

Quantas mais camadas tiver um sistema, mais fácil é de testar cada camada de forma isolada, utilizando *mocks* para simular as interações com outras camadas. Por isso, as importâncias atribuídas para a testabilidade são atribuídas comparando o número de camadas (não físicas) das várias soluções, sem contar com a camada da *API Gateway*. As camadas de integração/API Gateway não têm qualquer função lógica que possa ser testada de forma isolada, por isso não são consideradas para a testabilidade. A Solução A que tem quatro camadas testáveis é 2 vezes mais testável do que a Solução D que tem duas camadas. A solução C por ter 3 camadas é 3/2 vezes mais testável do que a Solução D. A Solução B por ter 3 camadas para além da camada de integração tem a mesma testabilidade que Solução C.

Para o critério do desempenho, a matriz de comparação par a par entre todas as alternativas tecnológicas está representada na Tabela 7.

Tabela 7 - Comparação par a par entre alternativas para critério desempenho

	Solução A	**Solução B**	**Solução C**	**Solução D**
Solução A	1	1	4/3	2
Solução B	1	1	1	2
Solução C	3/4	3/4	1	3/2
Solução D	1/2	1/2	2/3	1

Tal como referido na secção 2.2.3, a distribuição das camadas em vários níveis físicos também pode melhorar a escalabilidade e desempenho. Por esse motivo, para classificar as alternativas, considera-se que o desempenho aumenta com o número de camadas físicas. A Solução A e a solução B que têm quatro níveis tem 2 vezes mais desempenho do que a Solução D que tem dois níveis. A solução C por ter 3 camadas tem 3/2 vezes mais desempenho do que a Solução D.

Quanto ao critério da segurança, a matriz de comparação par a par entre todas as alternativas tecnológicas está representada na Tabela 8.

Tabela 8 - Comparação par a par entre alternativas para critério segurança

	Solução A	**Solução B**	**Solução C**	**Solução D**
Solução A	1	1	3/2	3
Solução B	1	1	3/2	3
Solução C	2/3	2/3	1	2
Solução D	1/3	1/3	1/2	1

Os protocolos de segurança podem variar em cada nível físico dos sistemas. Para a matriz de julgamentos resultante da Tabela 8 considera-se que na camada da apresentação não existem restrições de segurança. Mas nas camadas seguintes, os requisitos de segurança diferem para cada nível. E por isso, neste caso a segurança aumenta com o número de níveis físicos no lado *backend*. Portanto, a soluções A e B que tem três níveis no lado *backend* tem 3 vezes mais segurança do que a solução D. Já a solução C que tem 2 níveis no lado *backend* tem duas vezes mais segurança do que a solução que tem apenas um nível.

3.3.3 Obtenção do resultado do algoritmo

Tal como referido a primeira fase do AHP é a identificação dos objetivos, dos critérios e das alternativas. Para este processo de tomada de decisão o objetivo é a identificação do sistema de *Internet Banking* que melhor contribui para a satisfação do consumidor

Selecionou-se o estudo conduzido por (Sakhaei, Afshari, & Esmaili, 2014) para identificar os critérios para o AHP. Os critérios são a usabilidade, a testabilidade, o desempenho e a segurança. As alternativas são os sistemas de *Internet Banking* com diferentes estilos arquiteturais identificados neste documento, que foram obtidos por meio de artigos, e também o sistema selecionado para o projeto *Installments*. As alternativas foram denominadas como Solução A, Solução B, Solução C e Solução D, e foram descritas na secção 3.3.2.

A segunda fase é realização de matrizes de comparação para a par para os critérios e para as alternativas que foram apresentadas na secção 3.3.1 e na secção 3.3.2, respetivamente. A partir da matriz de comparação par a par dos critérios obteve-se o peso de cada critério, ou seja, as prioridades. O peso calculado do critério usabilidade é de 40% e os pesos dos restantes critérios é de 20%. Essa percentagem indica o quanto cada critério contribui para o objetivo que é a satisfação do utilizador quanto à utilização de serviços de *Internet Banking*. Os pesos dos critérios são agrupados no vetor de prioridade dos critérios. Recorde-se que o vetor da prioridade dos critérios é obtido calculando a média de cada linha da matriz de julgamentos normalizada.

Para as alternativas foi calculada uma matriz de comparação para a par para cada um dos critérios de forma a avaliar e comparar cada solução alternativa em função de cada critério. Por isso foram calculadas quatro matrizes de julgamentos. E pela mesma técnica indicada para os critérios foram calculados os pesos de cada alternativa para cada critério, que indicam o quanto cada alternativa satisfaz cada critério. O resultado das prioridades das alternativas para cada critério é representado na Tabela 9.

Tabela 9 – Prioridades das alternativas para cada critério

	Usabilidade	**Testabilidade**	**Desempenho**	**Segurança**
Solução A	0,27	0,33	0,31	0,33

Solução B	0,09	0,25	0,31	0,33
Solução C	0,09	0,25	0,23	0,22
Solução D	0,54	0,17	0,15	0,11

O RC calculado para a matriz de comparação dos critérios e para as quatro matrizes de comparação das alternativas é zero. O que significa que as matrizes não têm qualquer percentagem de inconsistência.

Conhecendo as prioridades dos critérios e as prioridades de todas as alternativas para cada critério, é possível obter as prioridades gerais das alternativas. Estas prioridades permitem encontrar o sistema que melhor satisfaz os utilizadores dos serviços de *Internet Banking*. As prioridades gerais são calculadas multiplicando matriz das prioridades das alternativas para cada critério, cujos valor estão na Tabela 9, pelo vetor das prioridades dos critérios.

O vetor das prioridades gerais identifica a Solução A como a alternativa que tem mais peso na satisfação do consumidor, que é de 0,287. A solução A tem a arquitetura selecionada para o projeto *Intallments*.

Após a aplicação deste algoritmo conclui-se que a arquitetura da solução *Installment*s é a que fornece melhor desempenho, a melhor segurança e é a mais testável entre todas as soluções de *Internet Banking* alternativas apresentadas neste documento.

4 Análise de requisitos

Neste capítulo é feita e recolha e análise de requisitos para o desenvolvimento de funcionalidades que visam o fracionamento de pagamentos

Este capítulo começa por apresentar de uma forma detalhada as funcionalidades que compõe o fracionamento de pagamentos. É também apresentado o sistema existente onde design das novas funcionalidades deve ser feito. De seguida apresenta o modelo de casos de uso do projeto, os cenários de atributos de qualidade, as restrições, e as preocupações arquiteturais. No final, o fluxo do negócio *Installments* é ilustrado através de diagramas de atividades, e os seus principais conceitos do projeto são ilustrados através de um diagrama de modelo de domínio.

4.1 Requisitos do sistema

A partir da análise das funcionalidades que já existiam nos microsserviços e da recolha de informações junto de *stakeholders* do projeto, é possível descrever de forma mais detalhada os requisitos do projeto. Os requisitos do projeto focam-se na criação de planos de fracionamento para compras de 150€ ou mais feitas com cartão de crédito.

O banco cliente pretende adicionar à sua aplicação móvel de *Internet Banking* a funcionalidade de pagamentos fracionados. Esta forma de pagamento permite que o cliente possa fracionar o pagamento das suas compras.

O único utilizador da aplicação de *Internet Banking* é o cliente do banco que utiliza a aplicação de *Internet Banking* para efetuar as suas operações bancárias. O cliente tem a função de consultar as suas transações e os seus planos, simular planos de fracionamento para posterior criação do seu próprio plano, e gerir os seus planos de fracionamento.

As tarefas iniciam-se com a consulta das transações disponíveis. Estas transações ou movimentos são compras com cartão de crédito de valor igual ou superior a 150 euros. As transações podem ser filtradas pelo cliente em montante, data e/ou cartões associados às transações. Qualquer uma destas transações pode ser fracionada pelo cliente, criando um plano de pagamento. Cada transação é composta por vários dados, entre os quais estão a chave da transação, o montante, a descrição, a data da transação e a moeda.

Antes de escolher o tipo de plano de fracionamento que quer criar, o cliente pode requerer a simulação de planos de fracionamento para cada uma das suas transações. Para que essa simulação seja feita o cliente precisa de fornecer entre outras informações a chave da transação, e o montante que pretende simular. Como resultado para cada simulação são

apresentados os montantes, as taxas aplicadas em cada prestação, o cartão associado, e a duração do fracionamento.

Ao criar um novo plano de fracionamento, o cliente seleciona uma compra e seleciona o número de prestações. O utilizador pode escolher entre 3, 6 ou 9 prestações. Antes da criação do plano é possível simular a criação do mesmo, como referido acima, para que o cliente possa confirmar ou não. A criação do plano de fracionamento só é confirmada inserindo as posições pedidas do código multicanal do cliente. Após a criação do novo plano, o cliente é informado do sucesso da operação e recebe o identificador e os detalhes do plano criado.

A personalização do plano pode ser feita logo após a sua criação, e é feita quando o cliente solicita a alteração do plano, atribuindo-lhe um nome e uma categoria. O cliente toma conhecimento da alteração do plano quando o sistema responde com sucesso. O cliente pode alterar o plano sempre que quiser.

O cliente também pode consultar todos os seus planos. Quando o cliente solicita os seus planos correntes, este recebe uma lista de todos os seus planos, que pode ser filtrada como o cliente quiser. O filtro contém alguns dos dados de entrada inseridos pelo cliente que são os identificadores dos cartões associados ao plano, os estados dos planos (ativo, concluído ou cancelado), data de início dos planos, e nome do plano que o cliente pretender.

Os planos visíveis para o cliente quando este não aplica qualquer filtro são diferentes dos planos visíveis para o cliente quando este aplica filtro. Caso não tenha sido aplicado filtro deve ser apresentado na lista os planos de fracionamento ativos, cancelados e os que que foram concluídos no último extrato do cartão. Caso tenha sido aplicado filtro deve constar na lista todos os planos que correspondam ao filtro aplicado. Por exemplo, se o cliente indicar que quer visualizar apenas os planos concluídos, o sistema apresenta todos os planos concluídos independentemente da sua data de conclusão.

Cada plano da lista de planos correntes apresenta características gerais do plano, como o seu identificador, a data de início, o montante total, o montante pago, o estado, o nome, a categoria, entre outros.

O cliente pode aceder aos detalhes de qualquer um dos seus planos correntes fornecendo o identificador do plano. Os detalhes do plano incluem o montante que já foi pago, a duração do plano (número de prestações), o número de prestações pagas, a data de início do plano, os detalhes de cada prestação, entre outros valores. Cada prestação do plano é identificada por um número de prestação, e contém a data de pagamento, o montante dessa prestação, e o estado de pagamento de cada prestação. Nos detalhes do plano o cliente pode acompanhar a evolução do pagamento das prestações do plano até à sua conclusão ou cancelamento.

O cliente pode cancelar o plano antes da sua conclusão, ou seja, antes de todas as suas prestações serem liquidadas. Para cancelar o plano o cliente terá também de inserir as posições do seu código multicanal, e antes de confirmar o cancelamento pode visualizar os detalhes do cancelamento. Nesses detalhes constam o montante em débito, o número de

prestações que faltam pagar, o montante já pago, entre outros valores. Após a confirmação do cancelamento, o sistema informa do sucesso da operação.

O cliente também pode simular planos de fracionamento sem depender de transações disponíveis. O cliente apenas precisa de fornecer um montante à sua escolha, que seja igual ou superior a 150€. E receberá como resposta uma simulação básica de planos de fracionamento para os números de prestações (3, 6 e 9) e cartões de crédito possíveis do cliente. Cada plano simulado é composto por montantes e taxas cobradas ao cliente.

Todos as etapas do fracionamento referidas têm operações em comum. Em primeiro lugar, antes de qualquer operação bancária, o cliente deve iniciar sessão no sistema. Só com o identificador de sessão do cliente é possível aceder a qualquer operação de fracionamento. Após o início de sessão, todas as operações do fracionamento necessitam da informação pessoal do cliente e da lista de cartões de crédito do cliente. Essa informação concede acesso aos detalhes do cartão de crédito associado a um plano ou transação do cliente. Essa operação de obtenção das informações pessoais do cliente é daqui em diante designada de operação inicial, porque deve ser executada sempre antes de qualquer operação do fracionamento.

Os dados pessoais do cliente são o seu nome, email e número de telefone. E cada cartão de crédito do cliente contem o identificador do cartão, o identificador da conta cartão, o nome da conta cartão, o URL da imagem do cartão e o limite do cartão de crédito.

Durante todo o processo, o cliente pode pedir contacto ao banco para esclarecer dúvidas, fornecendo os seus dados pessoais e a mensagem. A resposta do sistema é o sucesso ou o insucesso da operação. Este requisito funcional ainda não foi implementado, ao contrário dos restantes, e deve ser implementado neste trabalho.

Também o requisito de consultar os planos correntes que apesar de já ter sido implementado apresenta um bug que deve ser resolvido. O que acontece atualmente é que todos os planos concluídos são exibidos para o utilizador, sem aplicar filtros. Mas só devem ser exibidos os planos que tenham sido concluídos no último ciclo de cartão de crédito.

As funcionalidades descritas devem ser desenvolvidas nos microsserviços dedicados ao projeto que são *Installments Experience* e *Installments Process*. O serviço *Installments Process* deve comunicar com o sistema central do banco. As características dos serviços referidos, os componentes do sistema de *software* onde é desenvolvido o projeto *Installments*, e as tecnologias utilizadas foram descritos na secção 2.3.

É aconselhado utilizar uma ferramenta de cache para reduzir as interações com o sistema central às essenciais, evitando a sua sobrecarga. Por exemplo nas funcionalidades já implementadas, os dados obtidos da operação inicial são guardados em cache para serem posteriormente utilizados nas operações de fracionamento seguintes. O armazenamento de cache utilizado neste projeto é Redis.

Deve ser usada uma base de dados externa para armazenar dados customizados para os clientes. Essa base de dados é fornecida pelo serviço Cosmos DB. Esses dados não são relevantes para o sistema central do banco cliente, mas são relevantes para o cliente personalizar a informação dos seus planos. Por exemplo a categoria do plano é apenas relevante para o cliente personalizar o seu plano.

Por fim, durante a implementação, deve-se ter a preocupação de manter atualizadas as versões das tecnologias utilizadas no desenvolvimento, nomeadamente a versão de .NET.

Após a implementação das funcionalidades requeridas, o cliente deve remover a complexidade do código e outros *code smells* com a ajuda da ferramenta SonarQube. Também devem ser realizados testes unitários às funcionalidades implementadas nos dois serviços, que tenham uma cobertura de pelo menos 80% o código.

4.2 Proposta de *Design*

A arquitetura base e os componentes do sistema Instalments foram apresentados no capítulo 2, mais especificamente na secção 2.3. E também já foram implementadas algumas funcionalidades para o fracionamento. Portanto, trata-se de um sistema já existente ao qual devem ser adicionados novas funcionalidades, e devem ser testadas as já existentes. Dado que se trata de um sistema que já foi previamente criado, (Cervantes & Kazman, 2016) denominam-no de sistema *brownfield*.

Mais especificamente, as funcionalidades devem desenvolvidas nos serviços *Installments Experience* e *Installments Process*. Outros *stakeholders* terão a responsabilidade de desenvolver funcionalidades nas restantes camadas do sistema. Cada um dos serviços referidos tem uma determinada responsabilidade.

Na secção 2.3 foi referido que *Installments* possui uma arquitetura de microsserviços combinada com o estilo arquitetural *n-tier*. Mas para além disso, as três camadas do lado *backend* dessa arquitetura aplicam uma abordagem designada API-led Connectivity.

API-led Connectivity é uma evolução de SOA que interliga serviços através de APIs reutilizáveis. Essas APIs podem ser as APIs do sistema, as APIs de processo e as APIs de experiência. As APIs referidas têm a função de desbloquear dados de sistemas, compor dados em processos e entregá-los ao utilizador final, respetivamente (Pearlman, 2017).

As APIs de sistema acedem aos principais sistemas de registo, e isolam da complexidade ou de qualquer alteração de sistemas subjacentes (Pearlman, 2017). As APIs de processo podem ser reutilizadas em vários projetos (Pearlman, 2017). Estas APIs alteram com menos frequência. No sistema Instalments, a camada do Sistema Central fornece APIs para vários projetos do banco cliente.

As APIs de processo encapsulam os processos de negócio subjacentes num único sistema (ou vários sistemas), e são independentes dos sistemas que lhes deram origem e dos sistemas de destino (Pearlman, 2017). No sistema Instalments, a lógica de negócio relativa a este projeto é encapsulada no serviço *Installments Process*.

As APIs de experiência reconfiguram os dados provenientes de uma fonte comum para que possam ser lidos pelo público alvo (Pearlman, 2017). Uma API de experiência é projetada para uma experiência de utilizador específica (Pearlman, 2017). Estas APIs alteram com mais frequência. Relativamente ao sistema *Installments*, o serviço *Installments Experience* fornece API para ser consumida pelas aplicações móveis.

A abordagem API-led Connectivity aumenta a interoperabilidade, porque permite integrar diferentes tecnologias que comunicam através de APIs. Por exemplo, as APIs que permitem a comunicação entre os serviços de *Installments* utilizam o padrão REST. REST é um padrão arquitetural muito conhecido, facilitando assim a reutilização das APIs. Esta forma de integração das APIs aumenta a agilidade na entrega dos projetos.

Esta abordagem também aumenta a flexibilidade para mudanças, de modo a suportar as necessidades crescentes de alterações nos serviços. Por exemplo, enquanto que os esquema da base de dados de um sistema bancário central pode mudar anualmente, as aplicações bancárias online que se ligam a esses sistemas podem mudar semanalmente (Pearlman, 2017).

As alternativas a esta abordagem são as abordagens tradicionais de SOA, onde existe comunicação ponto a ponto (Pearlman, 2017).

4.3 Casos de uso

A funcionalidade é capacidade de o sistema efetuar o trabalho para o qual foi requerido. A funcionalidade primária não tem influência direta na arquitetura do sistema, ao contrário dos atributos de qualidade. Mas alguns cenários de atributos de qualidade podem estar diretamente ligados com algumas ou todas as funcionalidades (Cervantes & Kazman, 2016).

Para o *design* da arquitetura deve ser considerada pelo menos a funcionalidade primária. A funcionalidade primária pode ser definida como a “funcionalidade que é crítica para alcançar os objetivos de negócio que motivam o desenvolvimento do sistema”[3] (Cervantes & Kazman, 2016). Outra definição de funcionalidade primária pode ser que esta requer a utilização e vários elementos do sistema (Cervantes & Kazman, 2016).

[3] Tradução livre dos autores. No original "[...] functionality that is critical to achieve the business goals that motivate the development of the system."

Para *Installments*, os casos de uso que integram a funcionalidade primária são todos aqueles cuja funcionalidade é abrangida pelas camadas *Installments Expereince*, *Installments Process*, e Sistema Central.

Para cada ação que o cliente realiza na aplicação móvel, esta chama determinada operação no serviço *Installments Experience*, utilizando o padrão REST. Como foi referido, o trabalho descrito neste documento foca-se apenas no lado *backend* do sistema. Portanto, cada caso de uso mencionado neste documento representa o conjunto de chamadas HTTP necessárias para realizar a operação *Installments* que esse caso de uso refere. Assim, cada solicitação do cliente é realizada na forma de uma solicitação HTTP e cada resposta do sistema é dada como uma reposta HTTP.

A descrição das funcionalidades de *Installments* descrita na secção 4.1 pode ser estruturada num modelo de casos de uso. Os casos de uso são descritos na Tabela 10.

Tabela 10 – Casos de Uso para *Installments*

ID	**Caso de uso**
UC-1	Consultar Planos Correntes
UC-2	Pedir Contacto
UC-3	Consultar Transações Disponíveis
UC-4	Alterar Plano
UC-5	Criar Plano
UC-6	Consultar Detalhes do Plano
UC-7	Visualizar Simulação
UC-8	Visualizar Simulação Básica
UC-9	Simular Cancelamento do Plano
UC-10	Cancelar Plano
UC-11	Solicitar Operação Inicial

Os casos de uso também podem ser representados em forma de diagrama, o diagrama de casos de uso. Um diagrama de casos de uso identifica os principais comportamentos do sistema. A Figura 16 apresenta o diagrama de casos de uso para *Installments*.

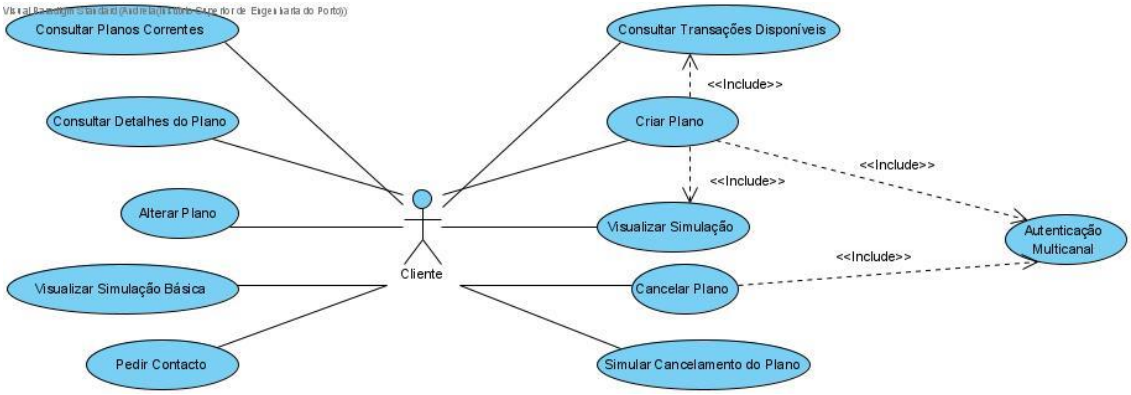

Figura 16 – Diagrama de casos de uso de *Installments*

Como observa na Figura 16 os casos de uso Consultar Transações Disponíveis e Visualizar Simulação estão incluídos em Criar Plano. Para a criação de um novo plano, o sistema deve conhecer qual a transação que o cliente selecionou para fracionar, que é obtida na lista de transações disponíveis. E deve conhecer o plano de pagamento correspondente à duração do plano selecionada pelo cliente, que é obtida na simulação.

Quer a criação, quer o cancelamento de plano necessitam de autenticação multicanal para serem executados. Todos os casos de uso incluem a operação inicial, por isso esse caso de uso não precisa de ser representado no diagrama da Figura 16.

Os casos de uso para *Installments* identificados na Figura 16 são detalhados nas secções que se seguem.

4.3.1 Consultar Planos Correntes

4.3.1.1 Descrição

O cliente quer consultar todos os seus planos de fracionamento e quer poder filtrar os planos por identificador do cartão, por estado (ativo, concluído ou cancelado), data de início do plano, e nome do plano. Caso não aplique nenhum filtro, os planos concluídos visíveis para o cliente devem ter sido concluídos no último ciclo de cobrança de cartão de crédito (1 mês) ou menos.

4.3.1.2 Pré-condições

O cliente está registado no sistema.

4.3.1.3 Pós-condições

Caso não tenha sido aplicado filtro deve ser apresentado na lista: planos de fracionamento que estão no estado ativo; planos de fracionamento que estão no estado cancelado; e planos de fracionamento no estado concluído que foram concluídos no último ciclo de cartão de crédito. Caso tenha sido aplicado filtro deve constar na lista todos os planos que correspondam ao filtro aplicado.

4.3.1.4 Importância

Alta

4.3.1.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial.
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.
5. O cliente solicita a visualização dos planos correntes, sem aplicar filtro.
6. O sistema responde com os dados correspondentes a todos seus planos ativos, todos os seus planos cancelados, e todos os planos concluídos no último ciclo de cartão de crédito.

4.3.1.6 Fluxo alternativo

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial.
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.
5. O cliente solicita a visualização dos planos correntes, aplicando o filtro desejado nos parâmetros de entrada: nome dos planos, identificadores dos cartões de crédito associados, data de início dos planos e/ou estado dos planos.
6. O sistema responde com os dados correspondentes aos seus planos devidamente filtrados.

4.3.2 Pedir contacto

4.3.2.1 Descrição

O cliente quer pedir contacto ao banco para esclarecimento de dúvidas.

4.3.2.2 Pré-condições

O cliente está registado no sistema.

4.3.2.3 Pós-condições

A informação de contacto do cliente é registada no sistema.

4.3.2.4 Importância
Alta

4.3.2.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial.
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.
5. O cliente solicita o pedido de contacto enviando o seu número de telefone, o seu endereço de email, a data inicial e a data final do contacto, a mensagem e outros dados.
6. O sistema responde com sucesso.

4.3.3 Consultar Transações Disponíveis

4.3.3.1 Descrição
O cliente quer ter acesso à lista de movimentos passíveis de fracionamento dos seus cartões de crédito.

4.3.3.2 Pré-condições

O cliente está registado no sistema.
4.3.3.3 Pós-condições

Se existirem movimentos passiveis de fracionamento, o cliente obtém a sua lista de transações.
4.3.3.4 Importância
Alta

4.3.3.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.

5. O cliente solicita a visualização das transações disponíveis, podendo fornecer filtros como os identificadores dos cartões às quais estas estão associadas, ou o intervalo das datas das transações.

6. O sistema responde com a lista de compras passíveis de fracionamento do cliente.

4.3.3.6 Fluxo alternativo

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.

2. O sistema responde com sucesso fornecendo o identificador de sessão.

3. O cliente solicita a visualização das transações disponíveis, podendo fornecer filtros como os identificadores dos cartões às quais estas estão associadas, ou o intervalo das datas das transações.

4. O sistema responde com mensagem de erro e solicita ao cliente a chamada da operação inicial.

4.3.4 Alterar Plano

4.3.4.1 Descrição

O cliente quer personalizar/alterar o seu plano de fracionamento.

4.3.4.2 Pré-condições:

O cliente está registado no sistema; o cliente tem pelo menos 1 plano registado.

4.3.4.3 Pós-condições

O plano selecionado pelo cliente é devidamente alterado.

4.3.4.4 Importância

Alta

4.3.4.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.

2. O sistema responde com sucesso fornecendo o identificador de sessão.

3. O cliente seleciona a operação inicial.

4. O sistema responde com a sua lista de cartões e a sua informação pessoal.

5. O sistema solicita a alteração do plano fornecendo o identificador do plano que pretende alterar, o novo nome e a nova categoria do plano.

6. O sistema responde com a informação do sucesso

4.3.5 Criar Plano

4.3.5.1 Descrição

O cliente quer criar um plano de fracionamento para um movimento da lista de compras do seu cartão de crédito.

4.3.5.2 Pré-condições

O cliente está registado no sistema; O cliente tem compras passíveis de fracionamento disponíveis.

4.3.5.3 Pós-condições

É criado um novo plano de fracionamento no sistema.

4.3.5.4 Importância

Alta

4.3.5.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.
5. O cliente solicita a visualização de transações disponíveis.
6. O sistema responde com a lista de transações disponíveis, cada transação é identificada com a chave de transação.
7. O cliente solicita a simulação do fracionamento de uma dada transação, fornecendo a chave da transação.
8. O sistema responde com a lista das simulações de todos os planos de fracionamento possíveis para a sua transação.
9. O cliente solicita as posições para a autenticação multicanal.
10. O sistema responde com as posições multicanal que o cliente deve preencher para criar plano.
11. O cliente solicita a criação do plano de fracionamento selecionando a duração do plano de fracionamento (3, 6, ou 9), a chave da transação que pretende fracionar e as posições pedidas do seu código multicanal.

12. O sistema responde com sucesso apresentado o identificador do novo plano e os detalhes do novo plano.

4.3.6 Consultar Detalhes do Plano

4.3.6.1 Descrição

O cliente pretende ver os detalhes do seu plano.

4.3.6.2 Pré-condições

O cliente está registado no sistema; o cliente tem pelo menos um plano.

4.3.6.3 Pós condições

O cliente tem acesso aos detalhes do plano que selecionou.

4.3.6.4 Importância

Alta

4.3.6.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente solicita a operação inicial.
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.
5. O cliente solicita os detalhes de um dos planos da lista, fornecendo o identificador desse plano e o identificador da conta cartão correspondente a esse plano.
6. O sistema responde com os detalhes do plano.

4.3.7 Visualizar Simulação

4.3.7.1 Descrição

O cliente quer poder simular o fracionamento da minha compra, recebendo várias opções possíveis de fracionamento.

4.3.7.2 Pré-condição

O cliente está registado no sistema; o cliente tem um movimento passível de fracionamento.

4.3.7.3 Pós-condição

O fracionamento de uma compra é simulado.

4.3.7.4 Importância

Alta

4.3.7.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.
5. O cliente solicita a simulação do fracionamento de uma dada transação, fornecendo a chave da transação, o montante que pretende simular e outros dados.
6. O sistema responde com a lista de todos os planos de fracionamento possíveis para a sua transação.

4.3.8 Visualizar Simulação Básica

4.3.8.1 Descrição

O cliente pretende simular planos de fracionamento de 3, 6 e 9 prestações para os seus cartões disponíveis, fornecendo determinado montante.

4.3.8.2 Pré-condições

O cliente está registado no sistema; o cliente tem transações disponíveis para simular.

4.3.8.3 Pós-condições

A simulação do fracionamento de um determinado montante para os números de prestações e cartões de crédito possíveis é apresentada ao cliente.

4.3.8.4 Importância

Alta

4.3.8.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial.
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.
5. O cliente solicita uma simulação básica de planos de fracionamento, fornecendo um determinado montante.

6. O sistema responde com a simulação básica do fracionamento do montante.

4.3.8.6 Fluxo alternativo

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial.
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.
5. O cliente solicita uma simulação básica de planos de fracionamento, fornecendo um determinado montante.
6. O sistema responde com erro na simulação básica.

4.3.9 Simular Cancelamento do Plano

4.3.9.1 Descrição

O cliente pretende obter os detalhes do cancelamento do plano antes de cancelar o mesmo.

4.3.9.2 Pré-condições

O cliente está registado no sistema; O cliente tem pelo menos um plano de fracionamento ativo.

4.3.9.3 Pós-condições

O cliente tem acesso à simulação do cancelamento do plano.

4.3.9.4 Importância

Alta

4.3.9.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e a palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial
4. O sistema responde com a lista de cartões e a informação pessoal do cliente.
5. O cliente solicita a simulação do cancelamento de um determinado plano fornecendo como entradas o identificador desse mesmo plano e outros valores.
6. O sistema fornece os detalhes para esse cancelamento.

4.3.10 Cancelar Plano

4.3.10.1 Descrição

O cliente quer cancelar um dos seus planos de fracionamento ativos.
4.3.10.2 Pré-condições

O cliente está registado no sistema; O cliente tem pelo menos um plano de fracionamento ativo.
4.3.10.3 Pós-condições

O plano de pagamento selecionado é cancelado.
4.3.10.4 Importância
Alta

4.3.10.5 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial
4. O sistema responde com a lista de cartões e a informação pessoal do cliente.
5. O cliente solicita as posições para a autenticação multicanal.
6. O sistema responde com as posições multicanal que o cliente deve preencher para o cancelamento do plano.
7. O cliente solicita o cancelamento de um determinado plano de fracionamento fornecendo o identificador do plano que pretende cancelar, as suas posições pedidas do seu código multicanal, e outras informações.
8. O sistema responde com sucesso apresentado o identificador do plano cancelado e o estado da transação.

4.3.11 Iniciar Operação

4.3.11.1 Descrição

O cliente quer aceder à sua ter acesso à sua informação pessoal e lista de cartões, para serem utilizadas posteriormente nas operações de fracionamento.
4.3.11.2 Pré-condições

O cliente está registado no sistema.

4.3.11.3 Pós-condições

A lista de cartões e outras informações pessoais são apresentadas ao cliente.

4.3.11.4 Fluxo básico

1. O cliente inicia sessão fornecendo o nome de utilizador e palavra passe.
2. O sistema responde com sucesso fornecendo o identificador de sessão.
3. O cliente seleciona a operação inicial fornecendo entre outros valores, e o seu identificador de sessão.
4. O sistema responde com a sua lista de cartões e a sua informação pessoal.

4.4 Cenários de Atributos de qualidade

De todos os *driver*s, os atributos de qualidade são aqueles que contribuem mais significativamente para a modelação da arquitetura. Dada a sua importância, é necessário quantificar o que é testabilidade, manutenibilidade, entre outros. A melhor forma de quantificar atributos de qualidade é a realização de cenários atributos de qualidade. Um cenário de atributo de qualidade descreve como um sistema é requerido para a responder a determinado estímulo. Um cenário de atributo de qualidade é dividido em cinco partes que são a fonte do estímulo, o estímulo, o ambiente, a resposta, e a medida da resposta. Cada cenário é testável e mensurável, por isso é fácil de avaliar se o sistema de *software* desenvolvido satisfaz o cenário na prática, através de medições e testes (Cervantes & Kazman, 2016).

Os cenários devem ser priorizados tal como os outros atributos de qualidade. Esta priorização pode ser feita em duas dimensões. Essas dimensões são a importância do cenário para o sucesso do sistema. A outra dimensão é o grau de risco técnico ou associado ao cenário. A escala utilizada para cotar cada uma das dimensões pode ser alta, média e baixa (Cervantes & Kazman, 2016).

A obtenção de atributos de qualidade deve ser considerada em qualquer fase do desenvolvimento de *software*. Ou seja, um atributo de qualidade pode não ser exclusivamente dependente do *design*, da implementação ou da implantação (Bass, Clements, & Kazman, 2003).

A partir da lista de requisitos do sistema conseguiu-se identificar vários cenários de atributos de qualidade. A Tabela 11 descreve todos esses cenários, os atributos de qualidade que cada um mede, os casos de uso associados a cada cenário, e as prioridades de cada cenário. As duas últimas colunas indicam as duas dimensões que medem a prioridade de cada cenário, que são a importância e a dificuldade.

Tabela 11 – Cenários de atributos de qualidade

ID	Atributo de Qualidade	Cenário	Casos de uso associados	Importância	Dificuldade
QA-1	Testabilidade	Os testes unitários aos serviços devem cobrir pelo menos 80% do código	Todos	Média	Alta
QA-2	Manutenibilidade	O tempo de correção dos *code smells* deve ser no máximo 5% do tempo de desenvolvimento de cada serviço	Todos	Média	Média

A definição dos cenários dos atributos de qualidade pode ser feita de forma mais detalhada, tendo em conta a divisão em cinco partes referidas.

Para QA-1 a fonte do estímulo é o desenvolvedor. O estímulo é o final do desenvolvimento dos testes unitários. Os artefactos são os serviços *Installments Experience* e *Installments Process*. O ambiente é a fase de desenvolvimento. A resposta é a observabilidade da cobertura dos testes unitários em cada serviço. E a medida de resposta é que os testes unitários devem cobrir pelo menos 80% do código de cada serviço.

Para QA-2 a fonte do estímulo é o desenvolvedor. O estímulo é o final da implementação dos casos de uso. Os artefactos são os serviços *Installments Experience* e *Installments Process*. O ambiente é a fase de desenvolvimento. A resposta é a observabilidade do tempo de correção dos *code smells* em cada serviço. E a medida de resposta é que o tempo de correção *dos code smells* deve ser no máximo 5% do tempo de desenvolvimento de cada serviço.

Se os cenários de atributos de qualidade não forem cumpridos, o projeto pode não passar nos testes de qualidade endereçados por outros *stakeholders* após a fase de desenvolvimento. Os dois cenários serão testados recorrendo a uma ferramenta de qualidade de código que é abordada mais à frente.

4.5 Restrições

As restrições no desenvolvimento devem fazer parte do processo de *design* arquitetural. Estas restrições podem ser tecnologias obrigatórias, outros sistemas com o quais este sistema deve comunicar, normas que devem ser cumpridas, prazos de entrega, entre outras (Cervantes & Kazman, 2016).

Para este projeto como existe uma arquitetura e tecnologias já definidas pelo banco cliente, existem restrições de *design* e implementação que devem ser obedecidas. A Tabela 12 enumera cada uma dessas restrições.

Tabela 12 – Restrições

Indentificador	**Restrição**
CON-1	O desenvolvimento deve ser feito em dois serviços: *Installments Experience* e *Installments Process*, tendo cada serviço responsabilidades definidas.
CON-2	O serviço *Installments Process* deve comunicar com o sistema central do banco cliente.
CON-3	As tecnologias que se devem utilizar na implementação são .NET 5, ASP.NET Web API, a *framework* do banco cliente, o serviço de base de dados NoSQL Cosmos DB, e o armazenamento de estruturas de dados Redis para implementar cache distribuída.

4.6 Preocupações arquiteturais

As preocupações arquiteturais compreendem aspetos adicionais a serem considerados no *design* arquitetural (Cervantes & Kazman, 2016).

A lista de preocupações arquiteturais a ter em conta no *design* de *Installments* são os apresentados na Tabela 13.

Tabela 13 – Preocupações arquiteturais

Indentificador	**Preocupação arquitetural**
CRN-1	Manter atualizada a versão da *framework* .NET
CON-2	Utilizar uma ferramenta de cache para reduzir as interações com o sistema central às essenciais, evitando a sua sobrecarga.

4.7 Modelo de Negócio

4.7.1 Diagrama de Atividades

Algumas operações do sistema *Installment*s dependem de valores de saída de outras operações. As operações de fracionamento obedecem a uma determinada ordem, nomeadamente as relativas à gestão dos planos de fracionamento. O fluxo dessas operações está ilustrado num diagrama de atividades como o da Figura 17. Este diagrama representa as operações que podem ser feitas sobre um determinado plano de fracionamento durante uma sessão de utilizador.

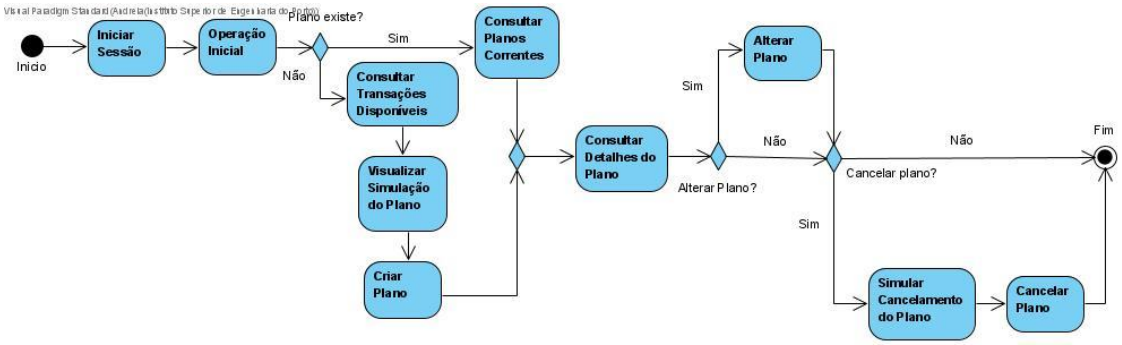

Figura 17 – Diagrama de atividades de *Installments*

Como se observa no diagrama da Figura 17, todas as operações são precedidas pelo início de sessão. De seguida deve ser realizada a operação inicial, porque todos as operações seguintes necessitam da informação do cliente retornada por esta operação.

A criação de um plano de fracionamento depende da consulta de transações disponíveis e da visualização da simulação, como referido acima. A consulta dos detalhes do plano pode ser feita após a criação do mesmo plano, e após a visualização de planos correntes. Estas duas últimas operações fornecem o identificador do plano para a consulta dos seus detalhes. A alteração e o cancelamento do plano são operações que o cliente pode realizar opcionalmente sobre um plano. A alteração do plano pode ser feita após a consulta dos detalhes para que o cliente tenha acesso ao nome e categoria atuais do plano. O cancelamento pode ser a última operação a ser executada sobre determinado plano, se o cliente não quiser pagar mais prestações.

É de salientar que o fluxo pode ser interrompido em qualquer uma das atividades definidas. Por exemplo, o cliente não é obrigado a cancelar um plano depois de visualizar a simulação do cancelamento do mesmo. Mas o cancelamento do plano pode ser precedido pela simulação do cancelamento, para que o cliente possa confirmar o seu pedido. A aplicação móvel

também obriga a que a ordem das tarefas neste fluxo seja obedecida, apesar de no *backend* não ser obrigatório por exemplo simular o cancelamento antes de cancelar o plano.

4.7.2 Modelo de domínio

Apesar do modelo de domínio não ser necessário para o *design* deste sistema, os principais conceitos e verbos obtidos na secção 4.1 permitem criar facilmente um diagrama de modelo de domínio. Um modelo de domínio reúne os principais conceitos de um projeto e as relações entre eles.

Os principais conceitos para este projeto são o cliente, cartão, compra, plano e prestação. O cliente é o único ator deste sistema e é responsável por gerir os seus planos de fracionamento e realizar zero ou mais compras passíveis de serem fracionadas. Um cliente pode ter vários cartões em sua posse. Um cartão podes ser usado para fazer várias compras, mas uma compra só pode ser feita com um cartão. Cada plano de fracionamento criado pelo cliente é relativo a uma compra. Cada plano contém 3, 6 ou 9 prestações, que são várias prestações.

O diagrama de classes que representa o modelo de domínio para *Installments* está ilustrado na Figura 18.

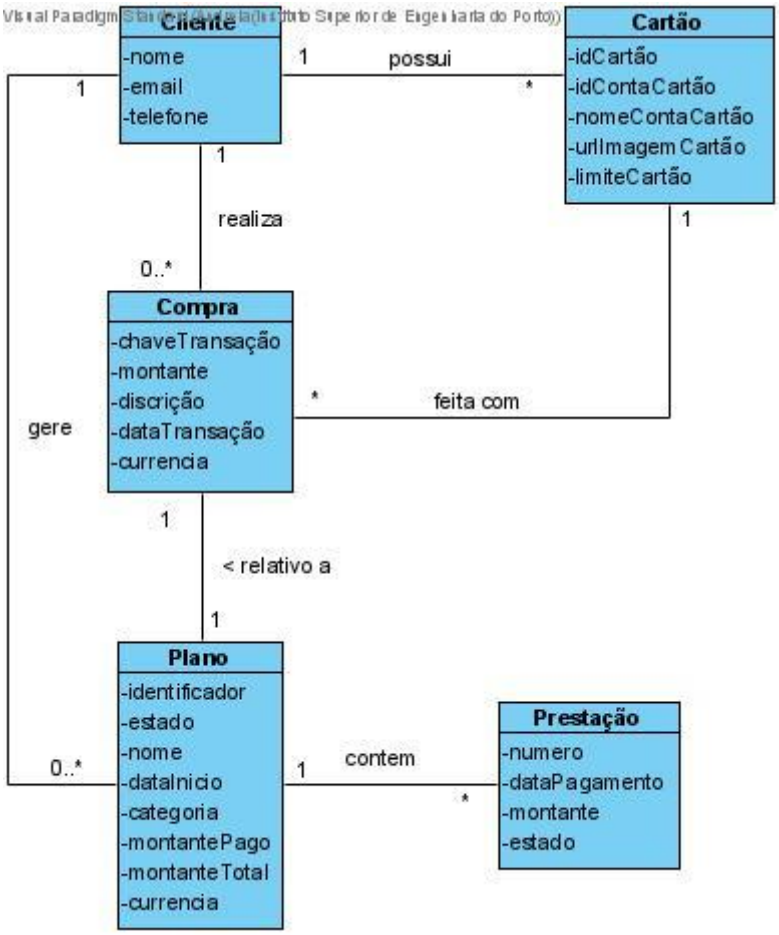

Figura 18 – Diagrama de Modelo de Domínio de *Installments*

Note-se que este diagrama não será usado no *design* da solução, e serve apenas para demonstrar os principais conceitos deste projeto. A equipa de desenvolvimento da ITSector não tem acesso aos objetos de domínio de *Installments*. Apenas tem acesso aos DTOs de cada caso de uso, que têm a função de transferir os dados estritamente necessários entre chamadas. Os atributos representados no diagrama de classes são alguns dos dados obtidos dos DTOs dos projetos.

5 *Design* da Solução

Este capítulo aborda a aplicação de *Attribute-Driven Design* (ADD) no desenvolvimento das funcionalidades indicadas nos requisitos.

O método de design escolhido é um processo iterativo que abrange os novos desenvolvimentos dos casos de uso Pedir Contacto e Consultar Planos Correntes na primeira iteração. A abordagem escolhida para os testes e a manutenção foi aplicada na segunda iteração desta metodologia.

5.1 Introdução a ADD

ADD é um processo iterativo de design de arquiteturas de *software* baseado em atributos de qualidade que o sistema deve atender. Esta abordagem começa com a definição de um conjunto de atributos de qualidade, i. e. requisitos não funcionais que são críticos para o sistema, requisitos funcionais e restrições. Segue-se a seleção da(s) parte(s) do sistema que se pretende desenhar, e finalmente a tomada de decisões arquiteturais que vão ao encontro dos referidos requisitos e restrições (Ali & Solis, 2014).

A Figura 19 ilustra as etapas da versão 3.0 desta metodologia iterativa.

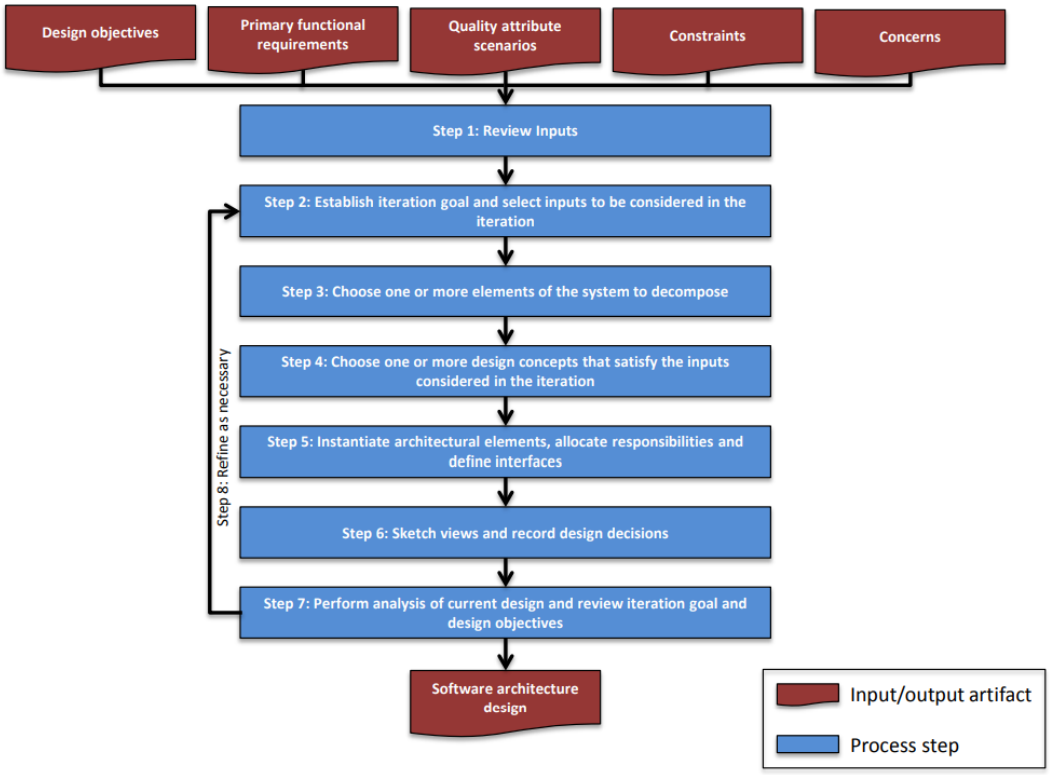

Figura 19 – Passos de ADD (Cervantes & Kazman, ADD 3.0: Rethinking Drivers and Decisions in the Design Process, 2015)

ADD necessita de um conjunto de entradas definidas de acordo com os objetivos do negócio para que o processo iterativo possa ser iniciado. Essas entradas são a proposta de *design*, os requisitos funcionais, as restrições, os atributos de qualidade e as preocupações arquiteturais. Estas entradas são também designadas de *drivers* arquiteturais por (Cervantes & Kazman, 2016).

Os requisitos funcionais podem ser expressos como casos de uso e os requisitos de atributos de qualidade são expressos como cenários de atributos de qualidade (Ali & Solis, 2014).

Os *driver*s arquiteturais são críticos para o sucesso do sistema, dado que estes moldam a arquitetura (Cervantes & Kazman, 2016).

Estas entradas são definidas para a *Installments* nas secções que se seguem. O *design* irá abranger todas as funcionalidades dos microsserviços de *Installments*, porque todas elas foram testadas.

5.2 Primeira iteração de ADD

5.2.1 Primeira etapa

Quando as entradas de ADD já estão disponíveis, a primeira etapa de ADD começa com a revisão das entradas. Nesta fase confirma-se que existem requisitos suficientes e devidamente priorizados de acordo com os objetivos do negócio (Bachmann, et al., 2006).

Esta etapa é só executada na primeira iteração, e foi elaborada nas secções acima.

5.2.2 Segunda etapa

Concluída a revisão das entradas, segue-se a segunda etapa de ADD. A segunda etapa visa estabelecer o objetivo da iteração pela seleção de *drivers* arquiteturais.

Se os serviços para o desenvolvimento deste projeto não existissem, i.e., se o sistema tivesse de ser desenvolvido de raiz, na primeira iteração desta etapa, todo o sistema teria de ser estruturado. Mas como a arquitetura base deste sistema já foi definida, o objetivo é o suporte da funcionalidade primária.

A funcionalidade primária de *Installments* compreende todas as funcionalidades indispensáveis para a criação de planos de fracionamento sobre movimentos de cartão de crédito a partir de 150 €. A maioria destas funcionalidades encontravam-se no sistema existente. No entanto, nem todos os casos de uso para completar a funcionalidade primária foram implementados.

Nesta etapa também são selecionados os *drivers* arquiteturais que tem maior impacto na funcionalidade primária. Esses *drivers* são as entradas às quais foi atribuída maior prioridade. Recorde-se que os casos de uso, os atributos de qualidade e as restrições e as preocupações sãos identificados de acordo com a Tabela 10, a Tabela 11, a Tabela 12 e a Tabela 13, respetivamente. As entradas a serem endereçadas são os casos de uso UC-1, e UC-2, as restrições CON-1, CON-2, CON-3 e as preocupações CRN-1 e CRN-2.

Note-se que nenhum cenário de atributo de qualidade foi selecionado porque não são necessários para o desenvolvimento da funcionalidade primária. Relativamente às restrições e preocupações arquiteturais, foram selecionadas todas, porque todas elas são tidas em conta no desenvolvimento da funcionalidade primária. Existem outros casos de uso que suportam a funcionalidade primária, no entanto ao contrário dos outros, os dois casos de uso referidos ainda não foram implementados.

5.2.3 Terceira etapa

A terceira etapa de ADD é a seleção de um ou mais elementos do sistema para decompor.

Os elementos a serem refinados nesta primeira iteração são os serviços *Installments Experience* e Installment Process. Nestes microsserviços, devem ser criados métodos que suportem a funcionalidade primária, que contêm UC-1 e UC-2, não esquecendo as responsabilidades definidas para cada um destes serviços.

5.2.4 Quarta etapa

A quarta etapa de ADD é a seleção dos conceitos de *design* que satisfazem os *drivers* selecionados.

A *framework* do banco cliente já pré-estabelece a estrutura interna de cada microsserviço. Por isso não existe a necessidade de definir padrões arquiteturais possíveis para aplicar nos serviços.

Todas as restrições CON-1, CON-2 E CON-3 foram selecionadas para esta iteração. Como o desenvolvimento deve ser feito num sistema existente, este já tem camadas com responsabilidades e tecnologias bem definidas. Cada decisão de *design* é aplicada aos casos de uso UC-1 e UC-2. Estes devem satisfazer a preocupação CRN-2 que aconselha a utilização de cache. E também a preocupação CRN-1.

Os conceitos de *design* apresentados nesta secção visam satisfazer os *driver*s selecionados, e são associados à cache e à separação de responsabilidades das diferentes camadas do sistema.

A Tabela 14 apresenta as decisões de *design* para esta iteração.

Tabela 14 – Conceitos de *design* para a primeira iteração

Decisões de *design* e Localização	**Racional**
Integração do armazenamento Redis num projeto de .NET	Redis é um armazenamento de estruturas de dados usado como base de dados, cache, agente de mensagens e mecanismo de *streaming* (redis.io). Este armazenamento é geralmente utilizado como cache distribuída. Cache distribuído na memória é uma forma de armazenamento em cache que permite que o cache abranja vários servidores para aumentar o desempenho e a escalabilidade (Khan, 2015). A alternativa à memória em cache é o armazenamento numa base de dados tradicional, no entanto iria requerer servidores de maior custo (Khan, 2015). Mas ao contrário das bases de dados tradicionais, a cache é um armazenamento temporário e de acesso mais rápido. A cache Redis pode ser usada num serviço ASP.NET Web API para armazenar e ler posteriormente respostas a solicitações à base de dados que se interliga com o serviço. A escrita de dados na cache é feita utilizando o método SetAsync, que armazena um valor para uma dada chave, e a leitura de dados é feita com o método GetAsync para ler o valor correspondente a uma chave fornecida como parâmetro. Os métodos referidos são fornecidos pela interface IDistributedCache que pode ser implementada num projeto em ASP.Net Core. A vantagem da cache distribuído é que reduz o tráfego para base de dados (Khan, 2015). Alternativamente, sem cache todas as solicitações seriam redirecionadas para a base de dados do sistema, mas tal afetaria o desempenho.

Separação de responsabilidades entre os serviços de experiência e os serviços de processo num sistema API-led Connectivity	Os serviços de Processo fornecem APIs de processo. As APIs de processo fornecem a lógica de negócio à camada de experiência, e podem ser reutilizadas por diferentes serviços de experiência. O serviço de processo pode orquestrar chamadas a várias APIs de sistema, que contêm as várias vertentes do negócio, ou outras APIs de processo. As APIs de experiência são projetadas para um determinado canal de comunicação do cliente, e fornecem apenas os dados necessários para o cliente nesse canal. Estas podem centralizar informação proveniente de diferentes APIs de processo para as fornecer à camada de *frontend*. Se esta divisão de responsabilidades não for bem feita os dois serviços não serão completamente independentes. Tal dificultará a manutenção dos dois serviços, e também a reutilização das suas APIs em futuros desenvolvimentos.

Os conceitos de *design* descritos na Tabela 14 podiam ser aplicadas em todos os casos de uso deste projeto definidos para a funcionalidade primária. Mas neste caso, são aplicados nos dois casos de uso selecionados para esta iteração.

5.2.5 Quinta etapa

A etapa 5 instância os elementos arquiteturais, aloca responsabilidades, e define interfaces.

O foco desta etapa é a definição das responsabilidades de cada microsserviço no desenvolvimento de cada caso de uso e as interações com recursos e serviços externos no *design* de cada caso de uso.

A Tabela 15 instância as decisões de *design* tomadas acima para os casos de uso selecionados para esta iteração.

Tabela 15 – Decisões arquiteturais para a primeira iteração

Decisões de *design* e Localização	**Racional**

Utilização de cache Redis no caso de uso Pedir Contacto	O caso de uso Pedir Contacto necessita das informações pessoais do cliente e da lista de cartões, ou seja, da operação inicial, tal como as outras operações de *Installments*. Assim, para a evitar a repetição de solicitações ao mesmo recurso foi utilizada o armazenamento de cache Redis. Redis armazena os dados da reposta à solicitação da operação inicial, para depois serem reutilizados nas operações seguintes. A operação Pedir Contacto usa as informações da operação inicial para ter informação para as solicitações ao Sistema Central. Uma dessas solicitações é o pedido dos detalhes do cartão de crédito associado ao identificador da conta cartão. Essas informações foram retiradas da classe InitOperation e guardadas em cache. Esses dados são posteriormente lidos da cache na classe RequestContactOperation para constar no pedido de contacto ao Sistema Central.
Separação de responsabilidades entre *Installments Process* e *Installments Experience* no *design* e implementação do algoritmo para caso de uso Pedir Contacto	Os únicos dados que o utilizador tem de fornecer ao serviço *Installments Experience* para o caso de uso Pedir Contacto são o seu número de telefone, o seu endereço de email, a data inicial e a data final do contacto, e a mensagem. Neste serviço não é executada nenhuma lógica de negócio porque este preocupa-se apenas com a experiência do utilizador. O serviço *Installments Process* recebe os dados do utilizador, e da cache e deve fazer duas solicitações ao sistema central. Uma das solicitações é para obter os detalhes de um dos seus cartões. A outra é para solicitar o pedido de contacto ao sistema central, fornecendo alguns dados da cache, os detalhes do cartão, e os dados inseridos pelo utilizador. O serviço responde com sucesso ou insucesso. Portanto, toda a lógica de negócio deste caso de uso está concentrada neste serviço.

Separação de responsabilidades entre *Installments Process* e *Installments Experience* no *design* e implementação do algoritmo pedido para o caso de uso Consultar Planos Correntes	Para consultar planos correntes, o serviço *Installments Experience* recebe como dados de entrada todas as variáveis que podem ser usadas como filtro inserido pelo cliente (nome do plano, identificadores dos cartões de crédito associados, data de início do plano e estado do plano). Neste serviço deve ser verificado se as entradas inseridas foram filtradas. Ou seja, é verificado se os valores da cada variável de entrada foram inseridos pelo cliente, ou se são os seus valores por defeito. Esta condição poderia ser verificada no serviço *Installments Process*, no entanto neste serviço já não é possível distinguir entre valores por defeito e valores inseridos pelo utilizador. Por esse motivo, é verificado em *Installments Experience* se os campos de entrada foram filtrados pelo cliente ou mantém o seu valor por defeito. O resultado dessa condição é guardado na variável booleana isFiltered, que juntamente com os outros parâmetros entra na solicitação a *Installments Process*. Neste serviço, podem existir mais campos de entrada do que aqueles que o utilizador pode alterar, e o valor isFiltered não é editado pelo utilizador. A maneira como planos concluídos são exibidos depende do valor de isFiltered. Se o valor isFiltered for *true*, então todos os planos concluídos obtidos do sistema central são atribuídos a uma variável para posterior processamento do filtro pedido pelo cliente. Se o valor isFiltered for false então são guardados na variável apenas os planos concluídos desde o último mês. Os restantes são descartados. A variável referida é dedicada à lista de planos em estado concluído do cliente. Esta é agrupada com restantes planos que devem constar na resposta do serviço *Installments Process*. No serviço *Installments Experience*, a lista de planos correntes é organizada para ser exibida na resposta ao cliente.

A diferença entre as descrições de UC-1 e UC-2 é que o algoritmo do primeiro é mais complexo, além disso já existia implementação prévia de UC-1. No entanto, a implementação necessitava de ajustes, porque disponibilizava todos os planos concluídos quando não era aplicado filtro. Mas a correção a disponibilizar nesta iteração resolve esse problema.

O caso de uso UC-1 também já tem a implementação do fornecedor de cache. Assim, apenas UC-2 necessita desta funcionalidade.

5.2.6 Sexta etapa

Na etapa 6, é feito um registo das decisões de *design* mais relevantes, e também são esboçadas vistas que representam essas mesmas decisões.

Nesta fase são retratadas em forma de esboço as decisões arquiteturais tomadas, por exemplo através da representação de diagramas. Em cada decisão de *design* são

documentados os elementos que compõe essa decisão de *design* e as responsabilidades (Software Engineering Institute | Carnegie Mellon University, 2015).

5.2.6.1 Caso de Uso Pedir Contacto

Para representar as decisões de *design* tomadas para o caso de uso Pedir Contacto, são apresentadas duas vistas de processos. As vistas de processos podem ser representadas em forma de diagramas de sequência segundo o modelo 4+1.

A primeira vista apresenta as interações entre o cliente, os dois serviços e o sistema central necessárias para o caso de uso UC-2. O que significa que este possui o nível de 2 de abstração segundo o modelo C4. O diagrama de sequência que representa a vista referida está ilustrado na Figura 20 - Digrama de sequência de nível 2 para Figura 20.

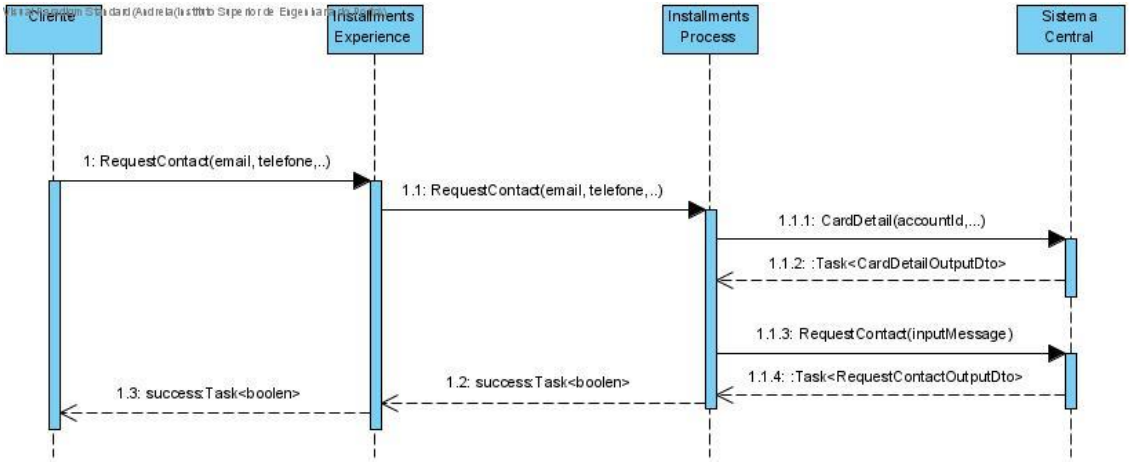

Figura 20 - Digrama de sequência de nível 2 para UC-2

Cada um dos métodos apresentado na Figura 20 são responsáveis por processar as comunicações HTTP entre os diferentes serviços. Note-se que nos métodos estão apresentados alguns dos campos de entrada necessários para as solicitações aos serviços.

O diagrama da Figura 20 esboça a decisão de *design* de separação das responsabilidades entre os serviços *Installments Experience* e *Installments Process*, para facilitar a reutilização das suas APIs por outros serviços.

A segunda vista de processo apresenta as interações entre o serviço *Installments Experience* e os principais elementos do serviço *Installments Process*, incluído o fornecedor de cache, para UC-2. Esta vista possui o nível de 3 de abstração segundo o modelo C4. O diagrama de sequência que representa esta vista é apresentado na Figura 21.

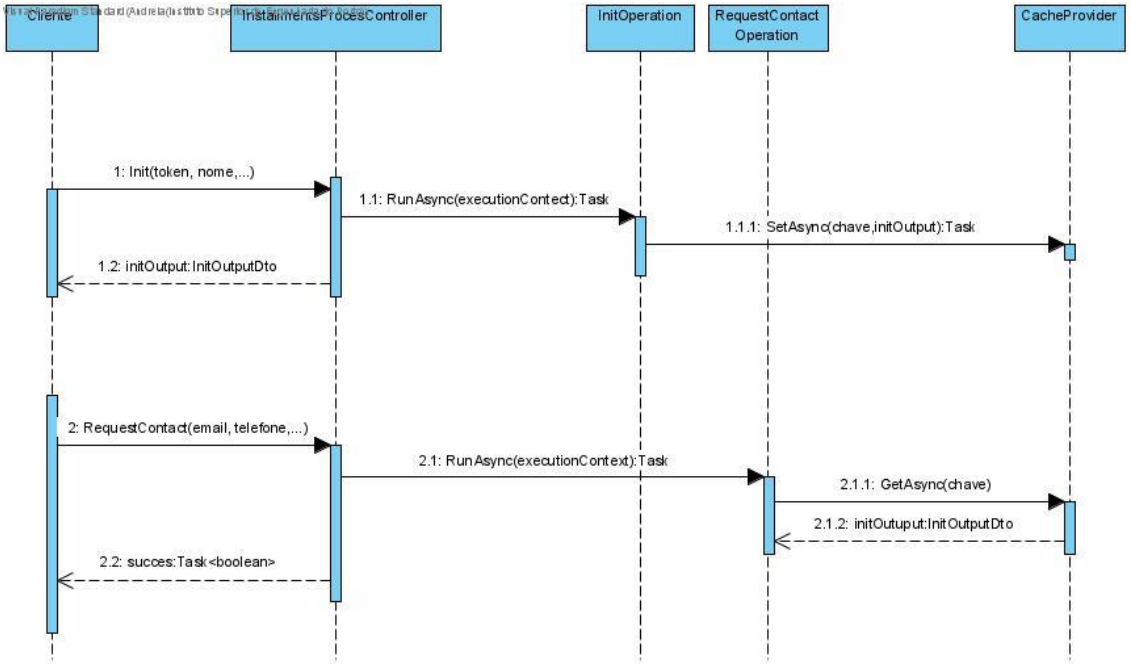

Figura 21 - Diagrama de sequência de nível 2 do serviço *Installments Process* para UC-2

O diagrama da Figura 21 esboça a decisão de *design* de utilização de cache Redis no caso de uso UC-2.

Note-se que a integração de Redis nos microsserviços *Installments* é encapsulada pela *framework* do banco cliente. Por isso, a implementação dos métodos GetAsync e SetAsync é um pouco diferente da implementação fornecida pela interface IDistributedCache.

O diagrama de sequência da Figura 21 é composto por elementos e interfaces. Os elementos e as respetivas responsabilidade relacionados com UC-2 que compõem o serviço *Installments Process* são apresentados na Tabela 16.

Tabela 16 – Responsabilidades dos elementos de *Installments Process* relacionados com UC-2

Elemento	**Responsabilidade**
*Installments*ProcessController	Gere todos os recursos do serviço expostos via Web API.
InitOperation	Representa as operações da lógica de negócio para a operação inicial, recebendo uma mensagem de entrada e produzindo uma mensagem de saída.
RequestContactOperation	Representa as operações da lógica de negócio para o caso de uso Pedir Contacto, recebendo uma mensagem de entrada e produzindo uma mensagem de saída.

CacheProvider	Fornece as funcionalidades da cache distribuída. Esta implementa métodos de guardar valores na cache chamando o método SetAsync, e também métodos de receber valores da cache chamando o método GetAsync.

As interfaces ou métodos que permitem a comunicação entre os diferentes elementos referidos na Tabela 16 acima estão apresentados na Tabela 17.

Tabela 17 – Métodos que compõem os elementos de *Installments Process* relacionados com UC-2

Nome do método	**Descrição**
	Elemento: *Installments*ProcessController
Task<RequestContactOutputDto>: RequestContact (email, telefone, …)	Gere todos os recursos do serviço expostos via Web API.
Task<InitOutputDto >: Init(identificador, nome,…)	Representa as operações da lógica de negócio para a operação inicial, recebendo uma mensagem de entrada e produzindo uma mensagem de saída.
	Elemento: RequestContactOperation
Task:RunAsync (executionContext)	Representa as operações da lógica de negócio para UC-11, recebendo uma mensagem de entrada e produzindo uma mensagem de saída.
	Elemento: InitOperation
Task:RunAsync (executionContext)	Executa todas as operações do negócio do serviço, inclusive solicitações a serviços externos, para UC-2
	Elemento: CacheProvider
Task: SetAsync(chave, initOutputDto)	Guarda valores na cache, onde cada valor guardado na cache é identificado por uma chave única. A variável initOutputDto representa o objeto guardado em cache.
Task< InitOutputDto>: GetAsync(chave)	Obtém um valor guardado na cache identificado por uma determinada chave. Esse valor é do tipo InitOutputDto.

É de salientar que o identificador de sessão e o nome do utilizador são campos que constam em todas as solicitações HTTP a ambos os serviços. Por isso estes valores são conhecidos por todos os casos de uso. O identificador de sessão tem um prazo de validade, o que significa que

se for ultrapassada essa validade não é possível fazer mais solicitações aos serviços com esse identificador.

A classe RequestContactOperation conhece o valor da chave correspondente ao valor "initOutputDto" guardado em cache na classe InitOperation, porque essa chave contém o identificador de sessão do cliente. Assim, é possível à classe RequestContactOperation receber os dados da resposta do método Init. Tal como mostra a Figura 21 a esses dados são guardados na variável "initOperationOutput".

Ambos os serviços *Installments* tem na sua estrutura interna fornecedores de serviços externos, como por exemplo cache, e uma classe Operation para cada caso de uso. Cada classe Operation implementa vários métodos, entre os quais o método RunAsync que é responsável pela lógica de negócio desse caso de uso. Portanto é este método que recebe entradas do caso de uso, excuta um algoritmo, realiza solicitações a outros serviços Web ou serviços externos como cache ou base de dados, e fornece as saídas resultantes do algoritmo.

5.2.6.2 Caso de uso Consultar Planos Correntes

A decisão de *design* tomada para UC-1 é a separação de responsabilidades entre *Installments Experience* e *Installments Process*. Para representar essa decisão de *design*, optou-se por criar um diagrama de atividades que descreve o algoritmo do caso de uso. Devido à sua complexidade, considera-se que um diagrama de atividades é mais adequado para esboçar o algoritmo do que um diagrama de sequência. Esse diagrama de atividades está apresentado na Figura 22.

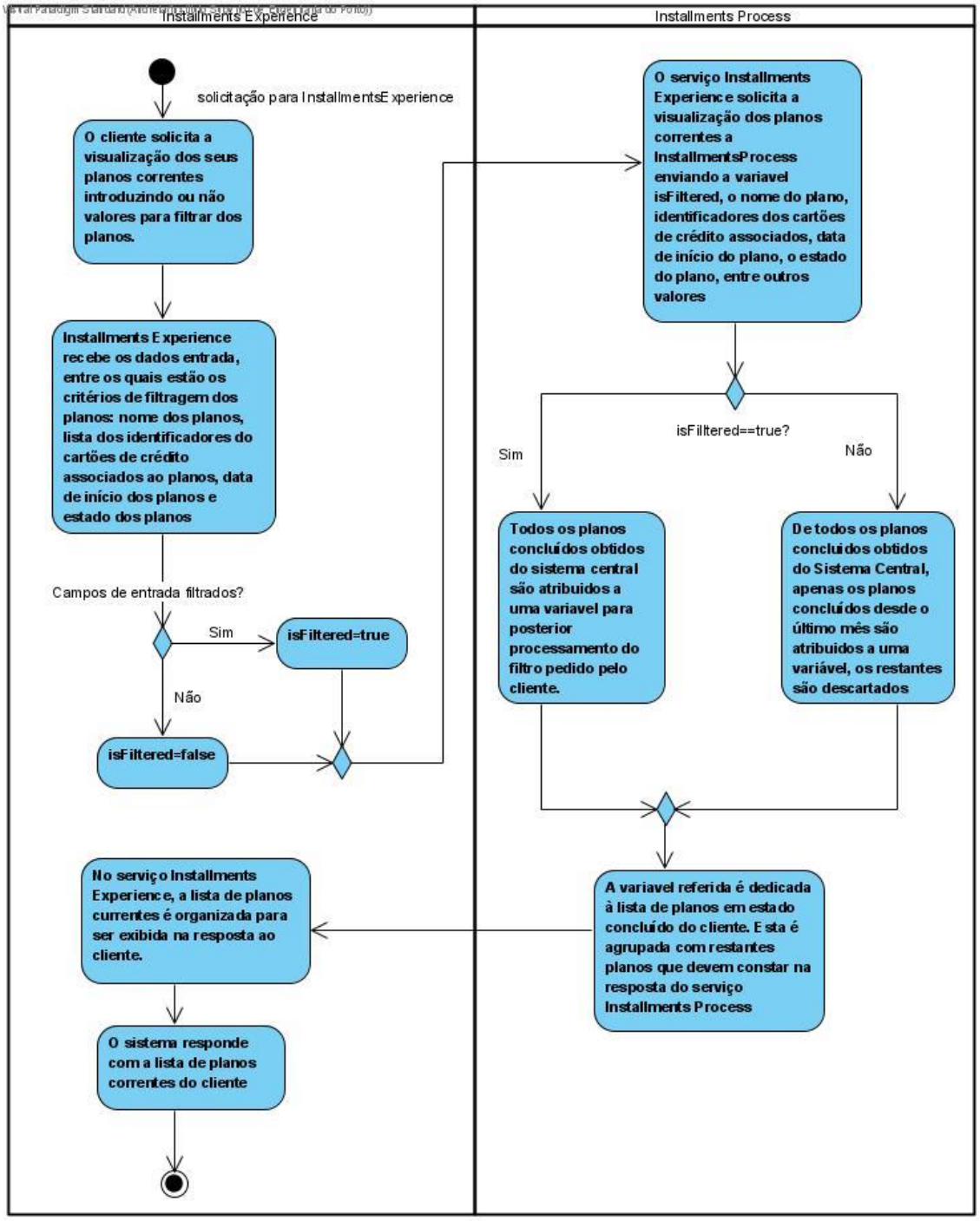

Figura 22 – Diagrama de atividades para UC-1

Cada atividade do diagrama da Figura 22 descreve uma tarefa a ser executada desde a solicitação do cliente até à resposta do sistema. Neste conjunto de tarefas não constam as interações com a cache porque já existiam. Note-se que o diagrama da Figura 22 divide as tarefas entre aquelas que são executadas no serviço *Installments Experience* e as que são executadas no serviço *Installments Process*.

5.2.7 Sétima etapa

A etapa 7, é feita uma análise do *design* feito até ao momento, é feita uma revisão dos objetivos da iteração e dos objetivos de *design*.

Nesta etapa, são analisadas as decisões que foram tomadas, e analisa-se a possibilidade de continuar o processo ADD com uma nova iteração, ou de se concluir o *design* (Software Engineering Institute | Carnegie Mellon University, 2015).

O objetivo de cada iteração consiste em selecionar um conjunto de *driver*s que constam na lista de entradas de ADD. Para cada um dos *driver*s é definida uma ou mais decisões de *design* que satisfaçam esses mesmos *driver*s. No final de uma iteração as decisões tomadas durante essa iteração são avaliadas. Para registar esse processo é utilizado um *Kamban Board* que acompanha o avanço do processo de *design*. Este Indica a influencia que as decisões de *design* tiveram nos *drivers* (Software Engineering Institute | Carnegie Mellon University, 2015).

O *Kamban Board* da primeira interação encontra-se apresentado no Anexo 1. Os *driver*s podem ser não endereçados, parcialmente endereçados ou completamente endereçados por cada uma das decisões de *design* tomadas durante esta iteração.

O que foi a feito nesta iteração foi o suporte à implementação dos casos de uso UC-1 e UC-2 nos serviços. Todas as funcionalidades (primárias ou não) devem ser desenhadas e implementadas para que seja possível aplicar os cenários de atributos de qualidade definidos.

Durante o desenvolvimento de cada caso de uso foi necessário aplicar todas as restrições de *design* e implementação. Porque os casos de uso UC-1 e UC-2 necessitavam dessas restrições para serem implementados. A preocupação de usar cache em UC-2 também foi completamente endereçada. E também a preocupação de manter atualizada a versão de .NET foi completamente endereçada, porque teve de ser atualizada para .NET 6 durante a implementação dos referidos casos de uso. E esta é a versão mais recente, que não voltou a ser atualizada.

Os atributos de qualidade não foram endereçados nesta iteração, bem como os casos de uso cuja implementação já tinha sido concluída, mas restam os testes unitários às respetivas implementações.

Apenas se considera os casos de uso completamente endereçados quando as suas funcionalidades forem testadas nos serviços. Assim, o processo deve continuar para endereçar os atributos de qualidade referidos e todos os casos de uso.

5.3 Segunda interação de ADD

5.3.1 Segunda etapa

Após a implementação de todas as funcionalidades primárias, o objetivo desta segunda iteração é o suporte aos cenários de atributos de qualidade e casos de uso adicionais.

As entradas selecionadas nesta iteração são os atributos que qualidade QA-1 e QA-2, e todos os casos de uso.

5.3.2 Terceira etapa

Os elementos escolhidos são todas as classes Operation presentes em todas as implementações dos casos de uso. Estas são responsáveis pela execução da lógica de negócio do caso de uso. Portanto, os testes devem ser feitos sobre esta classes. E a remoção de *code smells* também deve ser feita maioritariamente sobre estas classes.

5.3.3 Quarta etapa

Na presente secção é apresentada quarta etapa de ADD da 2ª iteração.

5.3.3.1 Táticas para Testabilidade

Os autores (Cervantes & Kazman, Designing software architectures: a practical approach, 2016) apresentam diferentes categorias de conceitos de *design* como por exemplo os padrões de *design* e arquiteturais, as táticas e os componentes desenvolvidos externamente. Como dois dos *driver*s selecionados referem-se a atributos de qualidade, optou-se por utilizar táticas.

As táticas são decisões de *design* que influenciam o controlo da resposta de um atributo de qualidade a um dado estímulo. Cada tática é uma opção de *design* para o arquiteto. Os autores descrevem algumas táticas para vários atributos de qualidade, entre os quais a testabilidade. As táticas são organizadas hierarquicamente para definir um caminho para o arquiteto pesquisar pelas táticas apropriadas (Bass, Clements, & Kazman, 2003).

O objetivo das táticas para a testabilidade é permitir uma testagem mais fácil cada vez que um incremento de desenvolvimento de *software* é adicionado (Bass, Clements, & Kazman, 2003). A Figura 23 ilustra as táticas para a testabilidade para o estímulo e resposta referidos.

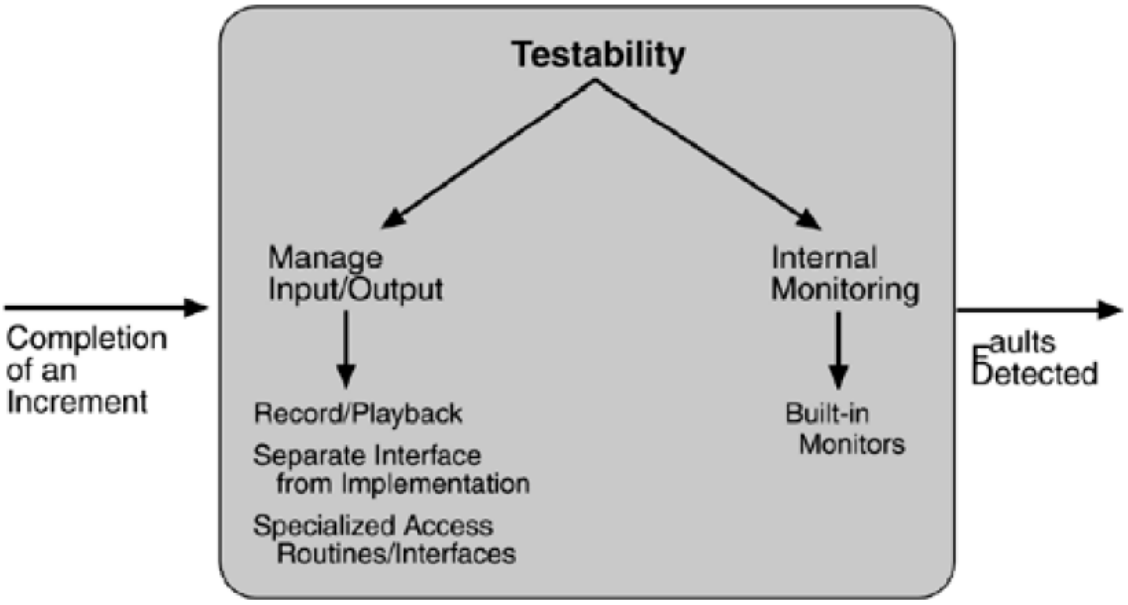

Figura 23 – Táticas para a testabilidade (Bass, Clements, & Kazman, 2003).

Uma das táticas de testabilidade referidas é monitorização embutida, que pode ser usada para melhorar a deteção das falhas como é sugerido no exemplo da Figura 23. Mas também podem ser usadas tecnologias de monitorização da cobertura de código.

Os autores (Bass, Clements, & Kazman, 2003) não sugerem nenhum conjunto de táticas para a manutenibilidade, mas também existem tecnologias para monitorizar a qualidade de código, e detetar *code smells*.

As tecnologias de monitorização podem ser consideradas componentes desenvolvidos externamente. Estes componentes não são criados como parte do projeto de desenvolvimento (Cervantes & Kazman, Designing software architectures: a practical approach, 2016), mas ajudam a cumprir requisitos do projeto. Um exemplo de ferramenta que monitoriza a qualidade de código é SonarQube, e foi a tecnologia selecionada para satisfazer QA-1 e QA-2.

5.3.3.2 SonarQube

Sonarqube é uma plataforma de código aberto para inspeção contínua da qualidade de código em várias ling uagens de programação. Esta ferramenta pode detetar problemas no código como *bug*s, *code smells*, complexidade de código, vulnerabilidades de segurança, duplicações de código, entre outros. E também analisa características como o número de linhas de código de um projeto e a percentagem de cobertura dos testes unitários realizados.

A plataforma utiliza várias métricas de qualidade, onde por exemplo para testes utiliza a percentagem de cobertura de código, e para manutenibilidade utiliza o número de *code smells*.

Os *code smells* são problemas de manutenção de código (SonarQube, 2022). Ou seja, quanto maior é o número de *code smells*, maior é a dificuldade de manutenção do código.

A percentagem de cobertura é percentagem de código que foi coberta pelos testes unitários.

A cobertura é uma mistura entre o calculo da cobertura de condição e o calculo da cobertura de linha (SonarQube, 2022). O cálculo da cobertura total de código é feito utilizando a equação 3 que é a seguinte:

$$Cobertura = \frac{CT + CF + LC}{2B + EL} \quad (4)$$

, onde CT é o número de condições avaliadas como verdadeiras pelo menos uma vez, CF é o número de condições avaliadas como falsas pelo menos uma vez, LC é o número de linhas cobertas, B é o número total de condições e EL é o número total de linhas executáveis (SonarQube, 2022).

Sonarqube classifica a manutenibilidade de um projeto através do cálculo da sua divida técnica. A dívida técnica é o esforço de tempo necessário para corrigir todos os *code smells* de um projeto. O calcula do índice de dívida técnica é a razão do custo de remediação dos *code smells* para o custo de desenvolvimento (SonarQube, 2022).

A classificação de projeto em termos manutenibilidade vai da classificação A, que á a mais alta, até E. Para ter a classificação A, o seu índice de dívida técnica deve ser inferior a 5% (SonarQube, 2022).

Em suma, quanto menor o número de *code smells*, menor será o índice de dívida técnica e maior será a classificação de manutenibilidade.

Com o ajuda da ferramenta descrita, as decisões de *design* baseiam-se na inspeção da cobertura dos *code smells*, e da cobertura para calcular as métricas de qualidade referidas abaixo. A Tabela 18 apresenta os conceitos de design para esta iteração.

Tabela 18 – Decisões de *design* para a segunda iteração

Decisões de design	**Racional**

Inspecionar a cobertura de código utilizando SonarQube	SonarQube analisa a cobertura de código recebendo um relatório de informações de cobertura dos testes unitários. O relatório é gerado por outra ferramenta. O relatório é enviado para SonarQube que por sua vez o exibe juntamente com as outras métricas (SonarQube, 2022). Em todas as classes do projeto, são exibidas as linhas que foram cobertas por testes, as que não foram cobertas por testes e as que foram cobertas parcialmente. Também é fornecida a percentagem total de cobertura de cada classe, de cada projeto, de cada pasta e da solução inteira.
Inspecionar os *code smells* e classificar a qualidade de código utilizando SonarQube	SonarQube faz uma análise estática do código, que inclui entre outras análises, a contagem do número de *code smells* em toda a solução, e em cada uma das suas classes. Em cada *code smell*, é disponibilizada a regra a que se refere o *code smell*, o seu grau de severidade, o tempo de esforço previsto para a sua resolução, a localização desse *code smell* na classe, entre outras informações. Conhecendo esta informação, a dívida técnica é calculada automaticamente, e a classificação de manutenibilidade é fornecida ao desenvolvedor.

5.3.4 Quinta etapa

A presente secção apresenta a etapa cinco de ADD.

A Tabela 19 resume as decisões tomadas para as soluções Installmets Process e *Installments Experience* para cumprir com os cenários de atributos de qualidade QA-1 e QA-2.

Tabela 19 – Decisões arquiteturais para a 2ª iteração

Decisão de design e Localização	**Racional**
Inspecionar a cobertura de código das duas soluções *Installments*	Criar dois projetos no Sonarqube, um para cada serviço *Installments*, que permitam fazer inspeção da cobertura de código em .NET. A cobertura é obtida de uma ferramenta suportada por .NET e pelo próprio SonarQube, que é a ferramenta Coverlet. Devem ser feitos os testes necessários até que o SonarQube indique que a cobertura de todo o serviço chegou a pelo menos 80% do código.
Inspecionar os *code smells* das duas soluções *Installments*	Devem ser criados dois projetos no Sonarqube, um para cada serviço *Installments*, que permitam fazer inspeção dos *code smells* em .NET. Os *code smells* detetados são exibidos e devem ser corrigidos em ambas as soluções .NET.
Remoção dos *code smells* dos dois serviços detetados por SonarQube	Elaborar as alterações solicitadas por SonarQube para corrigir os *code smells* das duas soluções *Installments*. Um dos *code smells* que a ferramenta dá mais tempo de esforço é a complexidade cognitiva. Este *code smell* solicita a redução da complexidade do código de um dado método, para aumentar a manutenibilidade do mesmo. A classe Operation costuma ter um dos métodos que mais tem *code smells* deste tipo, que é o método RunAsync porque é responsável pela lógica de negócio de um caso de uso para um determinado serviço. As correções podem ser feitas até que a classificação de manutenibilidade atribuída ao projeto seja A.

5.3.5 Sexta etapa

Na etapa 6, é feito um registo das decisões de *design* mais relevantes.

A decisão de *design* mais relevante que pode ser registada nesta fase é a redução da complexidade de código, que é o *code smell* mais severo.

Os *code smells* associados à complexidade de código identificados no serviço *Installments Experience* foram corrigidos reduzindo o número de estruturas de quebra de ciclo aninhadas

no método RunAsync. Essas estruturas de quebra de ciclo são por exemplo *if* e *for*. Algumas dessas estruturas foram transferidas para novos métodos privados que implementam parte do algoritmo da classe. Os métodos adicionados às classes Operation para reduzir a complexidade de código estão representados na Tabela 20.

Tabela 20 – Métodos adicionados às classes Operation de *Installments Experience*

Método	Descrição
	Elemento: AvailableTransactionsOperation
List<int>: GetAccountsIncludedInSearch (input, initCache)	Retorna uma lista de os identificadores da conta cartão correspondentes à lista de identificadores de cartão de crédito recebida de input (dados de entrada para UC-3). Essa correspondência é conhecida graças a valor da lista de cartões do cliente recebidos da variável InitCache. A lista de retorno é enviada posteriormente para *Installments Process*.
ValidateTransactionsFromProcess (availableResponse)	Método responsável pelo tratamento do erro proveniente da resposta à solicitação AvailableTransactions a *Installments*Process
	Elemento: CurrentPlansOperation
InstallmentItemDto: CreateInstallmentItem (plan,InitCache,name)	Mapeia o objeto que armazena o plano recebido da resposta de *Installments Process* num novo objeto do tipo InstallmentItemDto que será adicionado à resposta de *Installments Process*.
List<int>: GetAccountsIncludedInSearch (input, initCache)	Método com função idêntica ao método com o mesmo nome da classe AvailableTransactionsOperation, porque recebe também a lista dos identificadores dos cartões.
ValidateCurrentPlansFromProcess (input, initCache)	Método responsável pelo tratamento do erro proveniente da resposta à solicitação dos planos correntes a *Installments*Process..
	Elemento: PlanPayOffOperation
ValidatePlanPayOffFromProcess (payOffResponse)	Método responsável pelo tratamento do erro proveniente da resposta à solicitação do cancelamento do plano a *Installments Process*.

ValidatePlanPayOff SimulationFromProcess (payOffSimResponse)	Método responsável pelo tratamento do erro proveniente da resposta à solicitação de simulação do cancelamento do plano a *Installments Process*.
	Elemento: InitOperation
ValidateInitFromProcess (initResponse)	Método responsável pelo tratamento do erro proveniente da resposta à solicitação da operação inicial a *Installments Process*.

Os métodos da Tabela 20 são chamados no método RunAsync das respetivas classes para serem integrados no algoritmo do caso de uso.

Quando são feitos os testes unitários suficientes, e a remoção de *code smells* é suficiente, os cenários para QA-1 e QA-2 são cumpridos.

Após a realização de testes unitários necessários, as métricas de cobertura resultantes calculadas por SonarQube são apresentadas na Tabela 21.

Tabela 21 – Percentagens de cobertura dos serviços *Installments*

Projeto	**Cobertura**	**Cobertura de linha**	**Cobertura de condição**
Installments Process	85,7%	90,7%	72,5%
Installments Experience	83,0%	84,0%	75,9%

A Tabela 21 apresenta as percentagens de cobertura acima dos 80%. Por isso o cenário QA-1 foi cumprido.

Após a remoção de *code smells*, as métricas de manutenibilidade detetadas e calculadas por SonarQube são apresentadas na Tabela 22.

Tabela 22 – Métricas de manutenibilidade para os serviços *Installments*

Projeto	***Code smells***	**Dívida Técnica**	**Índice de Dívida Técnica**	**Classificação**
Installments Process	16	1h 45min	0,1%	A
Installments Experience	17	0 min	0,0%	A

Como se nota na Tabela 22, a divida técnica de *Installments Experience* é de 0 minutos. No entanto existem 17 *code smells*. Isto acontece porque os esforços de resolução de todos os *code smells* é de 0 minutos.

A principal diferença entre a manutenibilidade dos dois serviços representados na Tabela 22 é que *Installments Process* ainda contém *code smells* com o grau de severidade mais alto. Enquanto que todos os *code smells* com grau de severidade mais alto de *Installments Experience* foram resolvidos com os métodos da Tabela 20.

Note-se que o índice de dívida técnica é menor do que 5% em ambos os serviços. Por isso o cenário QA-2 foi alcançado.

5.3.6 Sétima etapa

Esta secção apresenta a etapa 7 da 2ª iteração de ADD.

O *Kamban Board* da segunda interação encontra-se apresentado no Anexo 2, onde todas as entradas de ADD estão completamente endereçadas. Os *driver*s arquiteturais que tinham sido completamente endereçados na iteração anterior foram eliminados nesta iteração.

Nesta iteração foram suportados os atributos de qualidade de testabilidade e manutenibilidade, e os seus cenários foram alcançados. Os casos de uso também foram desenhados, implementados e testados de acordo com os objetivos deste trabalho. Por isso o processo iterativo de ADD é concluído.

6 Implementação

O presente capítulo apresenta parte da implementação feita tendo em conta das decisões de design da primeira iteração do processo ADD para os UC-1 e UC-2. Recorde-se que foram usadas as tecnologias .NET, ASP.NET Web API, Redis e Cosmos DB.

Cada serviço *Installments* consiste numa Solução .NET composta por vários projetos.

6.1 Pedir Contacto

O caso de usos PedirContacto foi desenvolvido de raiz neste trabalho e o algoritmo é simples. Por isso a implementação deste caso de uso focou-se nas interações entre os diferentes serviços e na interação com o fornecedor de cache.

O serviço *Installments Experience* depende apenas de *Installments Process*. Por isso aqui é representado o excerto de código 1 que implementa a solicitação RequestContact do serviço *Installments Experience* para o serviço *Installments*Process.

```
var requestContactResponse = await
executionContext.GetApplicationClient<IInstallmentsProcessApplicationClient
>()
                .RequestContact(new
M3DF.Core.App.Message.SingleInputMessage<InstallmentsProcess.App.Contracts.
Dtos.RequestContactInputDto>
            {
                Data = new
InstallmentsProcess.App.Contracts.Dtos.RequestContactInputDto
                {
                    Metadata =
AutoMapperProvider.Map<M3DF.Infra.ApplicationClient.PTC.PTCMetadata,
InstallmentsProcess.App.Contracts.Dtos.PTCMetadata>(input.Metadata),
                    AccountId = input.AccountId,
                    PhoneNumber = input.PhoneNumber,
                    Email = input.Email,
                    ContactStartDate = input.ContactStartDate,
                    ContactEndDate = input.ContactEndDate,
                    Message = input.Message
                },
                Metadata = new()
            });
```

Código 1 – Solicitação RequestContact para *Installments Process*

Note-se que no excerto de código 1 estão representados os dados de entrada introduzidos pelo utilizador para o pedido de contacto. Esses dados consistem no número de telefone,

email, mensagem, data inicial e data final. Estes dados são guardados num objeto DTO, e são necessários para este caso de uso.

A resposta do *Installments Process* é guardada na variável requestContactResponse, que contem a resposta de sucesso ou insucesso ao pedido guardada numa variável booleana. Esta resposta por sua vez é usada por *Installments Experience* na resposta ao pedido do *frontend*.

O serviço *Installments Process* depende da cache Redis e do Sistema Central

A solicitação de um valor guardado na cache é feita chamando o método GetAsync que está contido no fornecedor de cache guardado na variável cacheProvider tal como apresenta o excerto de código 2.

```
var cacheProvider =
executionContext.GetProvider<IDistributedCacheProvider>();

var initCache = await
cacheProvider.GetAsync<InitOutputDto>(CacheConstants.GetSessionKey(CacheCon
stants.InitKey, input.Metadata.SessionId));
```

Código 2 – Leitura de valor da cache

No excerto de código 2 é feita uma leitura da cache que foi obtida a partir do identificador de sessão, que é a chave “SessionId”, e o valor da cache foi atribuído à variável initCache. Esta valor foi obtida da solicitação Init feito anteriormente, e o valor initCache contém a resposta a essa solicitação.

6.2 Consultar Planos Correntes

Tal como referido na decisão de design da 1ª iteração, a consulta dos planos correntes é feita introduzindo ou não um filtro no corpo da solicitação HTTP ao serviço *Installments Experience*. Esse filtro é composto pelos campos que compõem o corpo do pedido. Esses campos são o estado do plano (PlanStatus), identificadores dos cartões afetos aos planos (AccountIncludedInSearch), data de início dos planos (StartDate), e nome dos planos (PlanName).

A data foi obtida de um método específico para validar os dados de entrada, à qual é lhe atribuída um valor por defeito.

Antes de se adicionar as novas funcionalidades no caso de Uso a validação da data de início (startDate) era feita de cordo com o excerto de código 3.

```
if (input.StartDate == null)
{
      input.StartDate =
      System.DateTime.Now.AddMonths(ApplicationConstants.MonthsOfPlans);
}
```

Código 3 – Validação de StartDate antes das melhorias

O valor de "input" guarda os dados de entrada do serviço, sendo StartDate um dos atributos.

Nesse método deve ser verificado se a data foi filtrada pelo utilizador. Caso contrário não será possível distinguir entre a data introduzida pelo utilizador e a data atribuída por defeito a input.StartDate. Assim, foi criado um método booleano designado WasFilteredByDate ao qual é lhe atribuído um valor *true* apenas se o valor de *startDate* é vazio como é apresentado no excerto de código 4.

```
if (input.StartDate == null)
{
      input.StartDate =
      System.DateTime.Now.AddMonths(ApplicationConstants.MonthsOfPlans);
                this.WasFilteredByDate = false;
}
else
{
      this.WasFilteredByDate = true;
}
```

Código 4 – Validação de StartDate depois das melhorias

Após a verificação se a data foi filtrada, no método RunAsync foi criado uma variável denominada isFiltered que verifica se as restantes entradas introduzidas pelo cliente são filtradas. Às quais se junta WasFilteredByDate. Essa verificação é feita como no excerto de código 5.

```
var isFiltered = input.PlanStatus != null ||
input.AccountIncludedInSearch?.Count > 0 ||
                    this.WasFilteredByDate
|| !String.IsNullOrEmpty(input.PlanName);
```

Código 5 – Verificação se dados de entrada foram filtrados

O valor isFiltered foi adicionado como mais um campo da solicitação CurrentPlans para *Installments Process*, e foi atribuído ao parâmetro WasFiltered tal como indica o excerto de Código 6.

```
var currentPlansResponse = await
executionContext.GetApplicationClient<IInstallmentsProcessApplicationClient
>().CurrentPlans(new
M3DF.Core.App.Message.SingleInputMessage<InstallmentsProcess.App.Contracts.
Dtos.CurrentPlansInputDto>
            {
                Data = new
InstallmentsProcess.App.Contracts.Dtos.CurrentPlansInputDto
                {
                    Metadata =
AutoMapperProvider.Map<M3DF.Infra.ApplicationClient.PTC.PTCMetadata,
InstallmentsProcess.App.Contracts.Dtos.PTCMetadata>(input.Metadata),
                    AccountId = input.CardAccountId,
                    AccountIdList = accountIncludedInSearch.Count == 0 ?
accountsSearch : accountIncludedInSearch,
                    StartDate = input.StartDate,
                    Status = AutoMapperProvider.Map<PlanStatus?,
InstallmentsProcess.App.Contracts.Dtos.Installmentstatus?>(input.PlanStatus
),
                    Page = input.Page,
                    PageSize = input.PageSize,
                    PlanName = input.PlanName,
                    WasFiltered = isFiltered
                },
                Metadata = new()
            });
```

Código 6 – Solicitação CurrentPlans a *Installments Process*

No serviço *Installments Process*, na classe CurrentPlanOperation, são recebidos os dados de entrada no objeto input que incluem o atributo wasFiltered. Depois são feitas solicitações HTTP ao sistema central para se obter planos de vários estados.

Antes da implementação do novo desenvolvimento sabendo que a variável InstallmentPlsns contém todos os planos finalizados obtidos do sistema central. E sabendo que *Installments*List é a variável que guarda a lista de planos que são enviados na resposta a *Installments Experience*. O excerto de Código 7 mostra que todos os planos finalizados eram guardados e enviados para a mensagem de resposta, sempre que não era aplicado filtro.

```
if (InstallmentPlans?.Count > 0)
{

        installmentList.AddRange(currentsPlanDataOther.InstallmentPlans);


}
```

Código 7 – Obtenção de todos os planos finalizados em *Installments Process* antes das melhorias

Após os novos desenvolvimentos, o excerto de código resultante da correção de UC-1 encontra-se no Código 8.

```
if (InstallmentPlans?.Count > 0)
{
   if (!input.WasFiltered)
   {
        var plan = InstallmentPlans.Where(x =>
               x.PdOffDate >= DateTime.Today.AddMonths(-1)).ToList();

        installmentList.AddRange(plan);
   }
   else
   {

        installmentList.AddRange(InstallmentPlans);
   }
}
```

Código 8 – Obtenção de alguns planos finalizados em *Installments Process* depois das melhorias

Neste caso a variável plan filtra todos os planos de fracionamento concluídos para que aqueles com uma data de conclusão posterior ou igual ao mês anterior permaneçam na nova lista. A datada última cobrança do plano está definida pela variável PdOffDate.

Realça-se ainda um dos métodos que foi criado para reduzir a complexidade de código em *Installments Experience*. GetAccountsIncludedInSearch tem a função de converter os indentificadores dos cartões introduzidos pelo cliente nos respetivos identificadores de conta cartão, para serem adicionados à solicitação CurrentPlans a *Installments Process* tal como apresenta o Código 9.

```
private static List<int>
GetAccountsIncludedInSearch(CurrentPlansOperationInputDto input,
InitOperationOutputDto initCached)
        {
            var accountIncludedInSearch = new List<int>();
            if (input.AccountIncludedInSearch != null &&
input.AccountIncludedInSearch.Count > 0)
            {
                input.AccountIncludedInSearch.ForEach(accountId =>
                {
                    var account = initCached.CustomerCards.FirstOrDefault(a
=> a.CardId == accountId);
                    if (account != null)
                    {

accountIncludedInSearch.Add(account.CardAccountId.GetValueOrDefault());
                    }
                });
            }
            return accountIncludedInSearch;
        }
```

Código 9 – Implementação de GetAccountsIncludedInSearch

No Código 9, existem 3 estruturas de quebra de ciclo, que contribuíram para o aumento da complexidade no método RunAsync da classe CurrentPlansOperation. E por isso, o código constituído pelas estruturas if e ForEach foi transferido para o método externo GetAccountsIncludedInSearch.

7 Testes

O presente capítulo apresenta grande parte dos testes unitários feitos ao serviço *Installments Experience* para cumprir o QA-1. Os testes são feitos num projeto dedicado à realização de teste unitários na solução .NET.

Para cumprir com os objetivos de cobertura de código devem ser feitos testes unitários às classes Operation de todos os casos de uso nos serviços Installment Process e Installment Experience. Os testes necessitaram de *mocks*, que são objetos que simulam as respostas às solicitações de serviços subjacentes, e outros recursos externos (Redis e Cosmos DB). De forma a testar as diversas possibilidades, os testes abrangem *mocks* com diferences cenários de resposta. Para as várias combinações possíveis de cenários de resposta é possível criar vários casos de teste.

As secções que seguem apresentam os testes realizados no serviço *Installments Experience*. Os testes unitários da *Installments Process* foram feitos por outro *stakeholder* do projeto. Note-se que não estão representados todos os casos de teste feitos para cumprir a cobertura requerida, mas são apresentados os mais relevantes, tendo em conta as respostas de sucesso ou insucesso de cada *mock*.

7.1 Consultar Planos Correntes

A classe CurrentPlansOperation é responsável pela lógica de negócio de UC-1 no serviço *Installments Experience*. Esta classe recebe como entradas, os filtros aplicados opcionalmente pelo cliente. Esta classe também lê a informação pessoal do cliente proveniente da cache (obtida em UC-11), e se esta informação não estiver na cache a classe lança um erro. A classe envia a solicitação da consulta dos planos correntes para o serviço *Installments*Process. A resposta a essa solicitação caso não seja erro é a lista de planos correntes do cliente, que na classe CurrentPalnsOperation é devidamente organizada para ser disponibilizada ao cliente.

Em *Installments Experience*, os testes a UC-1 necessitas de 2 *mocks*. O primeiro obtém a informação da operação inicial lida da cache, e o segundo obtém a resposta de *Installments Process* para este caso de uso.

A Tabela 23 apresenta os casos de teste feitos para UC-1.

Tabela 23 - Casos de teste para UC-1

Identificador do caso de teste	**Entradas do cliente**	***Mock* da cache da operação inicial**	***Mock* de I. Process da resposta à solicitação dos planos correntes**	**Saída esperada**
Teste de UC-1 para sucesso dos planos correntes	Nome, identificadores dos cartões de crédito, data de início, estado dos planos	Cache com dados do cliente	Lista de planos correntes	Lista de planos correntes
Teste de UC-1 para insucesso dos planos correntes	Nome, identificadores dos cartões de crédito, data de início, estado dos planos	Cache com dados do cliente	Erro da operação	Erro
Teste de UC-1 para erro de cache vazia	Montante, identificador de sessão, ...	Cache vazia	Lista de planos correntes	Erro

7.2 Pedir Contacto

A classe RequestContactOperation é responsável pela lógica de negócio de UC-2 no serviço *Installments Experience*. Esta classe gere os dados de entrada deste caso de uso, realiza a solicitação de pedir contacto ao serviço *Installments Process*, e recebe a resposta de sucesso ou insucesso.

Em *Installments Experience*, os testes a UC-2 necessitam de informação da *mock* que simula a resposta à solicitação de pedir contacto a *Installments Process* que pode ser de sucesso ou insucesso.

A Tabela 24 apresenta os casos de teste para UC-2.

Tabela 24 – Casos de teste para UC-2

Identificador do caso de teste	**Entradas do cliente**	***Mock* de I. Process com a resposta ao pedido de contacto**	**Saída esperada**
Teste de UC-1 para sucesso do pedido de contacto	telefone, email, data inicial, data final, mensagem,...	Sucesso	Sucesso
Teste de UC-1 para insucesso do pedido de contacto	telefone, email, data inicial, data final, mensagem,...	Erro da operação	Insucesso

7.3 Consultar Transações Disponíveis

A classe AvailableTransactionsOperation é responsável pela lógica de negócio de UC-3 no serviço *Installments Experience*. Esta classe é responsável por atribuir valores aos campos de entrada, caso estes valores não tenham sido inseridos pelo utilizador. A classe também lê da cache a lista dos cartões do cliente. Se a cache estiver vazia o serviço retorna erro. O serviço envia os campos necessários para a solicitação deste caso de uso a *Installments Process*. Depois de receber a resposta, a classe organiza os itens da lista de transações proveniente de *Installments Process*, e solicita a simulação da primeira transação da lista. Finalmente, o serviço responde ao cliente com a lista de transações do cliente, e com a simulação da primeira da lista.

Em *Installments Experience*, os testes a UC-3 necessitas de 3 *mocks*. A primeiro obtém a informação da operação inicial lida da cache, a segundo obtém a resposta de *Installments Process* a este caso de uso. E a terceira obtém uma simulação do plano proveniente de *Installments Process*.

A Tabela 25 apresenta os casos de teste feitos para UC-3.

Tabela 25 – Casos de teste de UC-3

Identificação do caso de teste	**Entradas do cliente**	***Mock* da Cache da operação inicial**	***Mock* de I. Process da resposta à solicitação das transações disponíveis**	***Mock* de I. Process da resposta à solicitação de simulação do plano**	**Saída Esperada**
Teste de UC-3 para Sucesso das transações	Identificadores dos cartões, data inicial, data final,...	Cache com os dados do cliente	Lista das transações	Simulação de plano	Lista das transações
Teste de UC-3 para Insucesso das transações	Identificadores dos cartões, data inicial, data final,...	Cache com os dados do cliente	Erro na operação	Simulação do plano	Erro
Teste de UC-3 para insucesso da Simulação	Identificadores dos cartões, data inicial, data final,...	Cache com os dados do cliente	Lista das transações	Erro de Simulação	Lista das transações sem a simulação da primeira compra
Teste de UC-3 para Sucesso com Lista vazia	Identificadores dos cartões, data inicial, data final,...	Cache com os dados do cliente	Lista de transações vazia	Simulação do plano	Lista de transações vazia
Teste de UC-3 para erro de cache vazia	Identificadores dos cartões, data inicial, data final,...	Cache vazia	Lista das transações	Simulação de plano	Erro

7.4 Alterar Plano

A classe PlanChangeOperation é responsável pela lógica de negócio de UC-4 no serviço *Installments Experience*. Esta classe recebe como entradas, o identificador do plano que pretende alterar, o identificador do cartão, o novo nome, a nova categoria do plano, entre outros valores. Esta classe também lê a cache que guarda as informações do cliente. E obtém

da cache o identificador da conta cartão associada ao identificador do cartão fornecido pelo cliente. Esta classe envia a solicitação de alteração dos planos para o serviço *Installments Process*. A resposta a essa solicitação pode ser de sucesso ou insucesso. Caso seja de insucesso é retornado o erro correspondente.

Em *Installments Experience*, os testes a UC-4 necessitam de 2 *mocks*. O primeiro obtém a informação da operação inicial lida da cache, e o segundo obtém a resposta de *Installments Process* para este caso de uso.

A Tabela 26 apresenta os casos de teste feitos para UC-4.

Tabela 26 – Casos de teste para UC-4

Identificador do caso de teste	**Entradas do cliente**	***Mock* da cache da Operação Inicial**	***Mock* de I. Process com resposta à solicitação de alteração do plano**	**Saída esperada**
Teste de UC-4 para sucesso da alteração de plano	Identificador do plano, identificador do cartão, nome, categoria,...	Cache com dados do cliente	Sucesso	Sucesso
Teste de UC-4 para insucesso da alteração de plano	Identificador do plano, identificador do cartão, nome, categoria,...	Cache com dados do cliente	Erro da operação	Erro
Teste de UC-4 para cache vazia	Identificador do plano, identificador do cartão, nome, categoria,...	Cache vazia	Sucesso	Sucesso

7.5 Criar Plano

A classe PlanCreateOperation é responsável pela lógica de negócio de UC-5 no serviço *Installments Experience*. Esta classe recebe como entradas a duração do plano que quer criar, a chave da transação para a qual pretende criar plano, a posições do código multicanal, entre outros valores. Esta classe também lê a informação pessoal do cliente proveniente da cache (obtida em UC-11) e a informação das transações disponíveis (obtida em UC-3), e se estas informações não estiverem na cache a classe lança um erro. A classe também lê a informação da Simulação (obtida em UC-7) proveniente da cache. Se esta informação não estiver na cache, é solicitada a *Installments Process*. Depois de organizar a informação dos casos de uso referidos e da autenticação multicanal, é solicitada a criação de plano a *Installments Process*. A resposta a essa solicitação contém entre outras informações o identificador do novo plano. A seguir, a classe solicita os detalhes desse plano a *Installments Process* e caso a resposta seja de erro, é lançado o erro adequado, caso contrário, o serviço recebe os detalhes do novo plano. A nova informação recebida é organizada e enviada em resposta para o utilizador.

Em *Installments Experience*, os testes a UC-5 necessitam de várias *mocks*. Algumas simulam vários tipos de informação da cache provenientes de solicitações anteriores. Outras simulam respostas a diferentes solicitações a *Installments Process*.

A Tabela 27 apresenta os casos de teste feitos para UC-5.

Tabela 27 – Casos de teste para UC-5

Identificação do caso de teste	**Entradas do cliente**	***Mock* da Cache da operação inicial**	***Mock* da cache da lista de transaçõe s**	***Mock* da cache da simulação do plano**	***Mock* de I. Process da resposta à solicitação para criar plano**	***Mock* de I. Process da resposta à solicitação dos detalhs do plano**	**Saída Esperada**
Teste de UC-5 para Sucesso na criação de plano	duração do plano, chave da transação, posições do código multicanal,..	Informaçã o pessoal do cliente e lista de cartões	Cache da lista de transaçõe s disponívei s	Cache da Simulação de plano	Identificad or do novo plano,...	Detalhes do plano	Identifica dor do novo plano, detalhes do plano, ...

Teste de UC-5 para Insucesso da Criação de Plano	duração do plano, chave da transação, posições do código multicanal,..	Informação pessoal do cliente e lista de cartões	Cache lista de transações disponíveis	Cache Simulação do plano	Erro da operação	Detalhes do plano	Erro da criação do plano
Teste de UC-5 para Insucesso dos Detalhes do Plano	duração do plano, chave da transação, posições do código multicanal,..	Informação pessoal do cliente e lista de cartões	Cache da Lista de transações disponíveis	Cache da Simulação do plano	Identificador do novo plano,...	Erro dos detalhes do plano	Retorna erro para a criação do novo plano
Teste de UC-5 para erro de cache das informações do cliente vazia	duração do plano, chave da transação, posições do código multicanal,..	Cache vazia	Cache da Lista de transações disponíveis	Cache da Simulação do plano	Identificador do novo plano,...	Detalhes do plano	Erro
Teste de UC-5 para erro da Cache da Simulação	duração do plano, chave da transação, posições do código multicanal,.	Informação pessoal do cliente e lista de cartões	Cache da Lista de transações disponíveis	Cache Vazia	Identificador do novo plano	Detalhes do plano	Erro
Teste de UC-5 para sucesso da Cache das Transações Disponíveis	duração do plano, chave da transação, posições do código multicanal,.	Informação pessoal do cliente e lista de cartões	Cache Vazia	Cache da Simulação de plano	Identificador do novo plano	Detalhes do plano	Identificador do novo plano, detalhes do plano, ...

7.6 Consultar Detalhes do Plano

A classe PlanDetailOperation é responsável pela lógica de negócio de UC-6 no serviço *Installments Experience*. Esta classe recebe como entradas, o identificador do plano, o identificador do cartão, entre outros valores. Esta classe também lê a cache que guarda as informações do cliente, para obter o identificador da conta cartão associada ao cliente. Também é solicitada a operação inicial a *Installments Process* para ler a lista de contas do cliente. Caso essa solicitação retorne erro ou não seja encontrada o id da conta cartão referido na lista é lançado erro, e não é possível continuar a operação. A classe também solicita os detalhes do plano a *Installments Process*. Em resposta recebe os detalhes do plano pedido que são organizados juntamente com os dados da conta do cliente e são disponibilizados ao utilizador.

Em *Installments Experience*, os testes a UC-6 necessitam de 3 *mocks*. O primeiro simula a informação da operação inicial lida da cache, outro simula a resposta de *Installments Process* paraos detalhes do plano e outro simula a informação da operação inicial obtida de Installements Process.

A Tabela 28 apresenta os casos de teste feitos para UC-6.

Tabela 28 – Casos de teste para UC-6

Identificação do caso de teste	**Entradas do cliente**	***Mock* da Cache da operação inicial**	***Mock* de I. Process com resposta à solicitação dos detalhes do plano**	***Mock* de I. Process com a resposta à solicitação para iniciar operação**	**Saída Esperada**
Teste de UC-6 para Sucesso dos Detalhes do Plano	identificador do plano, o identificador do cartão	Cache com os dados do cliente	Detalhes do plano	Informações do cliente	Detalhes do plano
Teste de UC-6 para Insucesso dos Detalhes do Plano	identificador do plano, identificador do cartão,...	Cache com os dados do cliente	Erro da operação	Informações do cliente	Erro dos detalhes do plano
Teste de UC-6 para insucesso da operação Inicial	identificador do plano, identificador do cartão,...	Cache com os dados do cliente	Detalhes do plano	Retorna erro	Erro dos detalhes do plano

Teste de UC-6 erro da cache da Informação pessoal vazia	identificador do plano, identificador do cartão,...	Cache vazia	Detalhes do plano	Informações do cliente	Erro nos detalhes do plano

7.7 Visualizar Simulação

A classe SimulationOperation é responsável pela lógica de negócio de UC-7 no serviço *Installments Experience*. Esta classe recebe como entradas a chave da transação e o montante e outros valores. Esta classe também lê a cache que guarda as informações do cliente. E obtém da cache o identificador da conta cartão associada ao identificador do cartão fornecido pelo cliente. Se a cache com essa informação estiver vazia, é retornado erro e a operação termina. A classe solicita a simulação a *Installments Process* fornecendo as entradas necessárias, e pode receber como resposta os dados de simulação ou pode receber erro de simulação. Caso a resposta seja de sucesso, os dados da simulação são enviados para os dados de saída. Caso seja de insucesso é retornado o erro correspondente.

Em *Installments Experience*, os testes a UC-4 necessitam de 2 *mocks*. O primeiro contém a informação do cliente lida da cache, e o segundo contém a resposta de *Installments Process* à solicitação da simulação.

A Tabela 29 apresenta os casos de teste feitos para UC-7.

Tabela 29 – Casos de teste para UC-7

Identificador do caso de teste	**Entradas do cliente**	***Mock* da cache de informações pessoais**	***Mock* de I. Process da resposta Simulção**	**Saída esperada**
Teste de UC-7 para sucesso da Simulação	chave da transação, montante,...	Cache com dados do cliente	Simulação	Simulação
Teste de UC-7 para insucesso da Simulação	chave da transação, montante,...	Cache com dados do cliente	Erro de Operação	Erro de simulação
Teste de UC-7 para erro de cache vazia	chave da transação, montante,...	Cache vazia	Simulação	Erro de Simulação

7.8 Visualizar Simulação Básica

A classe BasicSimulationOperation, responsável pela lógica de negócio de UC-8 no serviço *Installments Experience*, recebe o montante pedido pelo cliente e lê a informação pessoal do cliente proveniente da cache. Se cache estiver vazia, a classe lança um erro, e já não é efetuada a operação. Esta classe envia a solicitação da simulação básica para o serviço *Installments Process*. A resposta a essa solicitação caso não seja erro é organizada juntamente com os dados dos cartões do cliente para disponibilizar ao utilizador.

Em *Installments Experience*, os testes a UC-8 necessitas de 2 *mocks*. O primeiro obtém a informação da operação inicial lida da cache, e o segundo obtém a resposta de *Installments Process* a este caso de uso.

A Tabela 30 apresenta os casos de teste feitos para UC-8.

Tabela 30 – Casos de testes para UC-8

Identificador do caso de teste	**Entradas do cliente**	***Mock* da cache da operação inicial**	***Mock* de I. Process com resposta à solicitação da simulação básica**	**Saída esperada**
Teste de UC-8 para sucesso da simulação básica	Montante, identificador de sessão, ...	Cache com dados do cliente	Simulação básica	Simulação básica de planos de pagamento
Teste de UC-8 para insucesso da simulação básica	Montante, identificador de sessão, ...	Cache com dados do cliente	Erro da operação	Erro de simulação básica
Teste de UC-8 para erro de cache vazia	Montante, identificador de sessão, ...	Cache vazia	Simulação básica	Erro de Simulação básica

7.9 Simular Cancelamento do Plano

A classe PlanPayOffSimulationOperation é responsável pela lógica de negócio de UC-9 no serviço *Installments Experience*. Esta classe recebe como entradas o identificador do plano a cancelar, e outros valores. Também é feita a leitura da cache com a informação do cliente,

que se estiver vazia, é lançado erro e a operação termina. Nesta classe é solicitado a *Installments Process* os detalhes do plano e os dados da simulação do cancelamento do plano. Se as respostas às solicitações referidas não forem mensagens de erro, os detalhes da simulação do cancelamento do plano são disponibilizados ao cliente na resposta à simulação do cancelamento do plano.

Em *Installments Experience*, os testes a UC-9 necessitam de informação de várias *mocks*. A maioria delas simula as respostas às diferentes solicitações a *Installments Process*.

A Tabela 31 apresenta os casos de teste para UC-9.

Tabela 31 - Casos de teste para UC-9

Identificação do caso de teste	**Entradas do cliente**	***Mock* da Cache da informação pessoal do**	***Mock* de I. Process com resposta à solicitação dos detalhes do plano**	***Mock* de I. Process com a resposta da simulação do cancelamento**	**Saída Esperada**
Teste de UC-9 para Sucesso da Simulação do cancelamento	identificador do plano,...	Informação pessoal do cliente e lista de cartões	Detalhes do plano	Simulação do cancelamento	Simulação do cancelamento do plano
Teste de UC-9 para Insucesso dos Detalhes do Plano	identificador do plano,...	Informação pessoal do cliente e lista de cartões	Erro na operação	Simulação do cancelamento	Erro

Teste de UC-9 para cache das informaçõe s do cliente vazia	identifica dor do plano,...	Cache vazia	Detalhes do plano	Simulação do cancelamento	Erro
Teste de UC-9 para erro da Simulação do cancelemt o	identifica dor do plano,...	Informação pessoal do cliente e lista de cartões	Detalhes do plano	Erro na operação	Erro

7.10Cancelar Plano

A classe PlanPayOffOperation é responsável pela lógica de negócio de UC-10 no serviço *Installments Experience*. Esta classe recebe como entradas o identificador do plano a cancelar, as posições do código multicanal do cliente e outros valores. Também é feita a leitura da cache com a informação do cliente para se obter o identificador da conta cartão do cliente. Se a cache estiver vazia não é retornado erro. Nesta classe é solicitado a *Installments Process* os detalhes do plano. Também é solicitado os dados da simulação do cancelamento do plano. Caso as solicitações anteriores não tenham retornado erro, a os dados de resposta das mesmas e a informação da autenticação multicanal são utilizados na solicitação para cancelar o plano a *Installments Process*.

Em *Installments Experience*, os testes a UC-10 necessitam de informação de várias *mocks*. A maioria delas simula as respostas às diferentes solicitações a *Installments Process*.

A Tabela 32 apresenta os casos de teste feitos para UC-10.

Tabela 32 – Casos de teste para UC-10

Identificação do caso de teste	**Entradas do cliente**	***Mock* da Cache da informação pessoal do**	***Mock* de I. Process com resposta à solicitação detalhes do plano**	***Mock* de I. Process com a resposta à simulação do cancelamento**	***Mock* de I. Process com a resposta à solicitação do cancelento**	**Saída Esperada**
Teste de UC-10 para Sucesso do cancelamento	identificador do plano, posições do código multicanal,...	Informação pessoal do cliente e lista de cartões	Detalhes do plano	Simulação do cancelamento	Sucesso do cancelamento	Sucesso do cancelamento do plano
Teste de UC-10 para Insucesso do cancelamento	identificador do plano, posições do código multicanal,...	Informação pessoal do cliente e lista de cartões	Detalhes do plano	Simulação do cancelamento	Erro na operação	Erro de cancelamento do plano
Teste de UC-10 para Insucesso dos Detalhes do Plano	identificador do plano, posições do código multicanal,...	Informação pessoal do cliente e lista de cartões	Erro na operação	Simulação do cancelamento	Sucesso do cancelamento	Erro de cancelamento do plano

Teste de UC-10 para cache das informaçõe s do cliente vazia	identifica dor do plano, posições do código multicana l,...	Cache vazia	Detalhes do plano	Simulação do cancelamento	Sucesso do cancelame nto	Sucesso do cancelam ento do plano
Teste de UC-10 para erro da Simulação do cancelemt o	identifica dor do plano, posições do código multicana l,...	Informação pessoal do cliente e lista de cartões	Detalhes do plano	Erro na operação	Sucesso do cancelame nto	Erro de cancelam ento do plano

7.11Iniciar Operação

A classe InitOperation é responsável pela lógica de negócio de UC-11 no serviço *Installments Experience*. Esta classe começa por verificar se existe informação do cliente em cache pesquisando pelo identificador de sessão do cliente. Se existir, essa mesma informação é retornada ao cliente e a operação termina. Caso contrário, a classe solicita a *Installments Process* a operação inicial. A resposta contém a informação pessoal do cliente, que é organizada e é guardada em cache para posterior utilização. A classe também atualiza a base de dados Cosmos DB com informação do novo cliente, caso este tenha sido o seu acesso. Por fim, os dados pessoais do cliente e a sua lista de cartões são disponibilizados ao cliente.

Em *Installments Experience*, os testes a UC-11 necessita de 3 *mocks*. A primeiro obtém a informação das informações do cliente lida da cache, a segundo obtém a resposta de *Installments Process* a este caso de uso. E o terceiro simula o fornecedor da base de dados Cosmos DB.

A Tabela 33 apresenta os casos de teste feitos para UC-11.

Tabela 33 – Casos de usos para UC-11

Identificação do caso de teste	**Entradas do cliente**	***Mock* da Cache da operação inicial**	***Mock* de I. Process da resposta à solicitação à operação inicial**	***Mock* do repositório da base de dados Cosmos DB**	**Saída Esperada**
Teste de UC-11 para Sucesso da cache	identificador de sessão, nome, ..	Cache com os dados do cliente	Informação do cliente	Cliente existente	Informações pessoais e lista de cartões
Teste de UC-11 para Insucesso da operação inicial	identificador de sessão, nome, ..	Cache Vazia	Erro na operação	Simulação do plano	Erro
Teste de UC3 para sucesso da operação inicial	identificador de sessão, nome, ..	Cache Vazia	Informação do cliente	Cliente existente	Informações pessoais e lista de cartões

8 Conclusões

O trabalho desenvolvido nesta dissertação focou-se no desenvolvimento de novas funcionalidades em microsserviços dedicados ao projeto *Installments*. E no teste dos métodos já existentes. O projeto *Installments* é um projeto de *Internet Banking* cujo objetivo é a criação de planos de fracionamento sobre movimentos de cartão de crédito acima de 150€. O desenvolvimento tinha de ser feito em dois microsserviços que são *Installments Process* e *Installments Experience* que são desenvolvidos em .NET. Este microsserviços aplicam uma abordagem API-led Connectivity, que permite a reutilização das suas APIs.

Todas as funcionalidades mais importantes dos serviços são representadas pelos casos de uso UC-1 a UC-11. De todos os casos de uso referidos, aquele que tinha de ser desenvolvido de raiz era UC-2. O caso de uso UC-1 apesar de já ter sido implementado, não tinha o algoritmo correto, e por isso este também foi corrigido durante o processo de desenvolvimento. Após a implementação e validação por outros *stakeholders* dos casos de uso implementados, foram efetuados testes unitários, e também foram removidos a maioria dos *code smells*. Estas duas últimas tarefas tinham como objetivo satisfazer QA-1 e QA-2, e foram satisfeitas com a ajuda da ferramenta SonarQube, que mostrava os resultados das métricas.

Uma das principais decisões tomadas na elaboração deste documento foi a metodologia de *design* a utilizar. Foi selecionada o método ADD que é adequado para desenvolver arquiteturas de sistemas novos ou para refinar arquiteturas já existentes, com base em atributos de qualidade. Este método foi selecionado após o conhecimento dos requisitos do sistema.

Uma das alternativas ao *design* referido é o *design* orientado a domínio (DDD). Este método não foi utilizado porque os microsserviços deste projeto apenas solicitam funcionalidades ao Sistema Central. E os dados são recebidos e enviados em forma de objetos DTOs. Portanto não existiu a necessidade de criação de uma base de dados que englobasse todo o modelo de negócio.

O projeto *Installments* está em constante evolução. Apesar de os objetivos definidos neste documento terem sido cumpridos, outros aspetos do projeto *Installments* ainda deverão ser refinados. Também, enquanto eram desenvolvidos os casos de uso em UC-1 e UC-2, outros *stakeholders* do projeto estariam a desenvolver outros casos de uso do projeto. E esses novos desenvolvimentos poderão obrigar a modificar testes unitários já feitos. Por todos os motivos indicados, a abordagem API-led Connectivity é útil para agilizar os projetos de desenvolvimento de *software* que envolvem o banco cliente.

Podem-se definir como trabalhos futuros as restantes fazes deste projeto já definidas no capítulo 1, que são o desenvolvimento de novas funcionalidades para a liquidação total e parcial dos planos de fracionamento e para outros tipos de fracionamento.

Referências

Ajamian, G. M., Boyce, S., Clamen, A., Fisher, E., Fountoulakis, S. A., Puri, P., & Seibert, R. (2004). Fuzzy Front End: Effective Methods, Tool, and Techniques. Em P. Belliveau, A. Griffin, & S. Somermeyer, *The PDMA toolbook 1 for new product development* (pp. 5-35). New York: John Wiley & Sons.

Ali, N., & Solis, C. (2014). Exploring how the attribute driven design method is perceived. Em I. Mistrik, R. Bahsoon, P. Eeles, R. Roshandel, & M. Stal, *Relating System Quality and Software Architecture* (pp. 23-40). Elsevier.

Appandairaj, A., & Murugappan, S. (2013). Service oriented architecture design for web based home banking systems with cloud based service. *International Journal of Emerging Technology and Advanced Engineering, 3*(1), 138-143.

Ariff, M. S., Yun, L. O., Zakuan, N., & Ismail, K. (2013). The impacts of service quality and customer satisfaction on customer loyalty in internet banking. *Procedia-Social and Behavioral Sciences, 81*, 469-473.

B2B International. (2022). *What is the Value Proposition Canvas?* Obtido em fevereiro de 2022, de B2B International: https://www.b2binternational.com/research/methods/faq/what-is-the-value-proposition-canvas/

Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., & Wood, B. (2006). *Attribute-Driven Design (ADD), version 2.0.* Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

Bahsoon, R., Emmerich, W., & Macke, J. (2005). Using real options to select stable middleware-induced software architectures. *IEE Proceedings-Software, 152*, 167-186. doi:10.1049/ip-sen:20045059

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice.* Addison-Wesley Professional.

Cervantes, H., & Kazman, R. (abril de 2015). *ADD 3.0: Rethinking Drivers and Decisions in the Design Process.* Obtido de Software Engineering Institute | Carnegie Mellon University: https://resources.sei.cmu.edu/asset_files/Presentation/2015_017_101_438648.pdf

Cervantes, H., & Kazman, R. (2016). *Designing software architectures: a practical approach.* Addison-Wesley Professional.

Chou, D. C. (01 de 03 de 2000). A Guide to the Internet Revolution in Banking. *Information Systems Management, 17*, 51-57. doi:10.1201/1078/43191.17.2.20000301/31227.6

Couto, J. P., Tiago, T., & Tiago, F. (2013). An analysis of Internet Banking in Portugal: the antecedents of mobile banking adoption. *International Journal of Advanced Computer Science and Applications, 4*(11), 117-123. doi:10.14569/IJACSA.2013.041116

Dobrica, L., & Niemela, E. (2002). A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering, 28*(7), 638-653.

European Central Bank. (23 de Julho de 2021). *Payments statistics: 2020.* Obtido de European Central Bank | Eurosystem: https://www.ecb.europa.eu/press/pr/stats/paysec/html/ecb.pis2020~5d0ea9dfa5.en.html

Gkoutzinis, A. A. (2006). *Internet banking and the law in Europe: Regulation, financial integration and electronic commerce.* Cambridge: Cambridge University Press.

Gorton, I. (2006). *Essential Software Architecture.* New York: Springer Berlin Heidelberg.

Grupo Marktest. (2 de Janeiro de 2019). *Aumentam utilizadores de Internet Banking.* Obtido em Fevereiro de 2022, de Grupo Marktest: https://www.marktest.com/wap/a/n/id~2480.aspx

Grupo Marktest. (7 de Janeiro de 2020). *Um em cada três possuidores de conta bancária usa mobile banking.* Obtido em fevereiro de 2022, de Grupo Marktest: https://www.marktest.com/wap/a/n/id~25be.aspx

Grupo Marktest. (3 de Novembro de 2021). *4,7 milhões contactam o banco por canais digitais.* Obtido em fevereiro de 2022, de Grupo Marktest: https://www.marktest.com/wap/a/n/id~280d.aspx

IBM Cloud Education. (7 de abril de 2021). *What is SOA (Service-Oriented Architecture)?* Obtido em fevereiro de 2022, de IBM: https://www.ibm.com/cloud/learn/soa

IBM Cloud Team. (14 de maio de 2021). *SOA vs Microservices.* Obtido em fevereiro de 2022, de IBM: https://www.ibm.com/cloud/blog/soa-vs-microservices

Isac, C., & Drigă, I. (2015). Internet banking services - a business necessity in the third millennium. *Annals of the University of Petroşani. Economics, 15*, 53-62.

Kainz, J. F. (2004). Migrating to Simpler Distributed Applications. *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications* (pp. 290–293). Vancouver, BC, CANADA: Association for Computing Machinery. doi:10.1145/1028664.1028773

Khan, I. (17 de agosto de 2015). *Scale Out - Distributed Caching On The Path To Scalability.* Obtido de Microsoft Learn: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/brownfield/distributed-caching-and-scalability

Koen, P., Ajamian, G., Burkart, R., Clamen, A., Davidson, J., D'Amore, R., . . . Johnson, A. (2001). Providing clarity and a common language to the "fuzzy front end". *Research-Technology Management, 44*(2), 46-55.

Martins, C., Oliveira, T., & Popovič, A. (1 de fevereiro de 2014). Understanding the Internet banking adoption: A unified theory of acceptance and use of technology and perceived risk application. *International Journal of Information Management, 34*, 1-13. doi:10.1016/j.ijinfomgt.2013.06.002

Mia, M. A., Rahman, M. A., & Uddin, M. (2007). E-Banking: Evolution, Status and Prospect. *The Cost and Management, 35*(1), 36-48. Obtido de https://ssrn.com/abstract=2371134

Microsoft. (2022). *Microservice architecture style .* Obtido de Microsoft: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

Microsoft Patterns & Practices Team. (2009). *Microsoft® Application Architecture Guide, 2nd Edition (Patterns & Practices).* Microsoft Press.

Miniwatts Marketing Group. (3 de Julho de 2021). *Internet World Stats - Usage and Population Statistics.* Obtido em fevereiro de 2022, de Internet World Stats: www.internetworldstats.com

Morais, J. R. (20 de Fevereiro de 2017). *Banca fecha balcões, mas internet banking não cresce.* Obtido de Diário de Notícias.

MuleSoft. (s.d.). *API-led connectivity - The next step in the evolution of SOA.*

Nawaz, M., Motiwalla, L., & Deokar, A. V. (2018). Adaptive User Interface for a Personalized Mobile Banking App. *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization* (pp. 141–142). Singapore: Association for Computing Machinery. Obtido de https://doi.org/10.1145/3213586.3226209

Oracle. (2010). *Enterprise Beans.* Obtido de The Java EE 5 Tutorial: https://docs.oracle.com/javaee/5/tutorial/doc/bncmb.html

Oracle. (setembro de 2010). *Overview of the Duke's Bank Application.* Obtido de The Java EE 5 Tutorial: https://docs.oracle.com/javaee/5/tutorial/doc/bncma.html

Osterwalder, A., & Pigneur, Y. (2003). Modeling value propositions in e-Business. *Proceedings of the 5th international conference on Electronic commerce* (pp. 429-436). Pittsburgh, Pennsylvania, USA: Association for Computing Machinery. doi:10.1145/948005.948061

Pearlman, S. (12 de julho de 2017). *What is API-led Connectivity?* Obtido de MuleSoft: https://blogs.mulesoft.com/learn-apis/api-led-connectivity/what-is-api-led-connectivity/

Pereira, D. (5 de Março de 2019). *Canvas da Proposta de Valor.* Obtido em Fevereiro de 2022, de O Analista de Modelos de Negócios: https://analistamodelosdenegocios.com.br/canvas-da-proposta-de-valor/

Pereira, D. (16 de Janeiro de 2021). *What is the Value Proposition Canvas?* Obtido em Fevereiro de 2022, de The Business Model Analyst: https://businessmodelanalyst.com/value-proposition-canvas/

Red Hat. (2005). *Chapter 4. The Duke's Bank Application.* Obtido de JBoss.org: https://docs.jboss.org/jbossas/getting_started/v4/html/dukesbank.html

Richardison, C. (2014). *What are microservices?* Obtido em fevereiro de 2022, de Microservices.io: https://microservices.io/

Richardson, C. (2019). *Microservice Architecture Pattern.* Obtido em fevereiro de 2022, de Microservices.io: https://microservices.io/patterns/microservices.html

Roser, M., Ritchie, H., & Ortiz-Ospina, E. (2015). *Internet.* Obtido em fevereiro de 2022, de Our World in Data: https://ourworldindata.org/internet

Saaty, R. W. (1987). The analytic hierarchy process—what it is and how it is used. *Mathematical modelling, 9*(3-5), 161-176.

Saaty, T. L. (1988). What is the Analytic Hierarchy Process? *Mathematical Models for Decision Support* (pp. 109-121). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-83555-1_5

Saaty, T. L. (2004). Decision making—the analytic hierarchy and network processes (AHP/ANP). *Journal of systems science and systems engineering, 13*(1), 1-35.

Sakhaei, S. F., Afshari, A., & Esmaili, E. (2014). The impact of service quality on customer satisfaction in Internet banking. *Journal of mathematics and computer science, 1*(1), 33-40.

Shih, Y.-Y., & Fang, K. (2006). Effects of network quality attributes on customer adoption intentions of internet banking. *Total Quality Management & Business Excellence, 17*(1), 61-77.

Software Engineering Institute | Carnegie Mellon University. (2015). *SATURN 2015: Rethinking Drivers + Decisions in the Design Process by Kazman and Cervantes [Video]*. Obtido de Youtube: https://www.youtube.com/watch?v=d_gRoI3oHag

SonarQube. (2022). *.NET Test Coverage.* Obtido em outubro de 2022, de SonarQube Docs: https://docs.sonarqube.org/latest/analysis/test-coverage/dotnet-test-coverage/

SonarQube. (2022). *Concepts.* Obtido em setembro de 2022, de SonarQube Docs: https://docs.sonarqube.org/latest/user-guide/concepts/

SonarQube. (2022). *Metric Definitions.* Obtido em setembro de 2022, de SonarQube Docs: https://docs.sonarqube.org/latest/user-guide/metric-definitions/

Statista. (16 de Dezembro de 2021). *Smartphone sales worldwide.* Obtido de Statista: https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/

Woodall, T. (2003). Conceptualising 'value for the customer': an attributional, structural and dispositional analysis. *Academy of marketing science review, 12*(1), 1-42.

Anexo 1

Tabela 34 – *Kamban Board* da 1ª iteração

Não endereçado	**Parcialmente endereçado**	**Completamente endereçado**	**Decisões de *design* tomadas durante a 1ª iteração**
	UC-1		Este caso de uso foi desenvolvimento tendo em conta que teria de ser desenhado e implementado em serviços com responsabilidades distintas. E também sabendo que teriam de ser reutilizadas informações em comum com outros casos de uso, através da implementação de cache distribuída.
	UC-2		Desenvolveu se o algoritmo que complementa o funcionamento deste caso de uso tendo em conta que teria de ser desenhado e implementado em serviços com responsabilidades distintas.
UC-3			
UC-4			
UC-5			
UC-6			
UC-7			
UC-8			
UC-9			
UC-10			
UC-11			
QA-1			
QA-2			

		CON-1	A funcionalidade dos dois casos de uso desenvolvidos nesta iteração está distribuída entre os serviços *Installments Experience* e *Installments Process* tendo em conta que os serviços tem responsabilidades distintas.
		CON-2	O *design* e implementação de UC-1 e UC-2 preveem que *Installments Process* consuma as APIs fornecidas pelo Sistema Central.
		CON-3	A implementação dos casos de uso e a execução dos mesmos em *Installments Experience* e *Installments Process* necessita da utilização das tecnologias referidas. Além disso, foi implementada cache distribuída em UC-2, fornecida por Redis.
		CRN-1	Nenhuma decisão relevante tomada.
		CRN-2	Utilização de cache Redis em UC-2

Anexo 2

Tabela 35 – *Kamban Board* da 2ª iteração

Não endereçado	**Parcialmente endereçado**	**Completamente endereçado**	**Decisões de *design* tomadas durante a 2ª iteração**
		UC-1	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-2	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-3	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-4	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-5	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-6	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-7	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-8	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-9	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-10	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		UC-11	Utilização de SonarQube para medir a manutenibilidade e a cobertura.
		QA-1	Utilização de SonarQube para a cobertura dos serviços .
		QA-2	Utilização de SonarQube para monitorizar a manutenibilidade dos serviços