INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Migração de uma plataforma web na área de e-commerce para uma arquitetura de micro-serviços

LUCAS SOARES DE ANDRADE
Outubro de 2021

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Migração de uma plataforma web na área de e-commerce para uma arquitetura de micro-serviços

Lucas Andrade

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

Orientador: Alexandre Bragança

Júri:
Presidente:

Vogais:

Porto, 15 de Outubro de 2021

Resumo

A implementação de um sistema usando o estilo arquitetural dos micro-serviços promete várias vantagens atraentes para o desenvolvimento de software mas também acarreta a introdução de várias complexidades que se não forem bem geridas podem ditar a desgraça de um projeto.

Nesta dissertação é feita uma revisão da literatura sobre essas complexidades e é tomado o caso de estudo de um monólito para a divisão em micro-serviços. Então, são propostos dois desenhos para essa divisão e é implementado um deles. O sistema daí resultante é avaliado e comparado com o monólito, principalmente em termos de tamanho, complexidade, comportamento sob carga, e do atraso da propagação da informação devido ao uso de meios de comunicação assíncronos.

Palavras-chave: Micro-serviços, consistência, acoplamento, complexidade, assíncrono

Abstract

The implementation of a system using the micro-service architectural style promises many advantages that can be attractive for the development of the software, but also introduces many complexities that, if not taken care of, can spell doom and gloom for the project.

In this dissertation a review of the literature about such complexities is done, and a monolith is divided into micro-services. Two designs are proposed for this division and one of them is implemented. Then the system resulting from such division is evaluated and compared with the monolith, mainly in terms of its size, complexity, behaviour under load and in the delay of propagation of information due to the use of asynchronous communication methods.

Keywords: Micro-services, consistency, coupling, complexity, asynchronous

Agradecimentos

Queria agradecer à minha esposa Vanessa que está ao meu lado desde há 10 anos e ao meu filho de 4 meses cujo amor e carinho me manteve resoluto mesmo nas piores alturas. Sem vós a vida não teria cor.

Queria agradecer aos meus pais, que toda a vida se esforçaram para me dar uma vida melhor. Sem vós, nada disto seria possível.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
ATG	Art and Technology Group
BASE	Basically Available, Soft state, Eventual consistency
BC	Bounded Context
CAP	Consistency, Availability and Partition
CD	Continuous Delivery
CI	Continuous Integration
CORS	Cross-Origin Resource Sharing
CQRS	Command Query Responsability Segregation
CRUD	Create, Read, Update, Delete
DDD	Domain-driven design
DTO	Data Transfer Object
ESB	Enterprise Service Bus
HATEOAS	Hypermedia as the Engine of Application State
HMAC	Hash-Based Messaging Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
PaaS	Platform as a Service
PR	Pull Request
REST	Representational State Transfer
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SSO	Single Sign-On
TLS	Transport Layer Security
UI	User Interface
URI	Uniform Resource Identifier

Capítulo 1

Introdução

1.1 Contexto

O comércio eletrónico começou a ganhar popularidade em meados da década de 90 [1], e desde então esse domínio já teve bastante tempo para se tornar num domínio bastante maduro [2]. Existem no mercado diversas ferramentas que podem ser utilizadas para facilitar e acelerar o trabalho de colocar no mercado um novo serviço.

No comércio eletrónico é importante disponibilidade e confiabilidade para evitar perdas de receita. Tendo em conta que podem haver grandes números de utilizadores simultâneos, bem como grandes picos de utilização (como por exemplo, em certos mercados na altura do Natal e da *black friday*) é necessário garantir que o sistema tem a capacidade de lidar com esses níveis de concorrência.

Historicamente o desenvolvimento da maioria das aplicações era feito usando uma arquitetura monolítica, ou seja, numa única unidade indivisível de código. Tipicamente o problema da escalabilidade pode ser resolvido replicando instâncias da aplicação, mas isso pode ser bastante dispendioso para aplicações muito grandes. Também a manutenção a longo prazo se torna cada vez mais dispendiosa com o acumular de funcionalidades e complexidade [3].

O domínio do comércio eletrónico é, à partida, atraente para uma implementação numa arquitetura de micro-serviços, a qual tem o potencial de aliviar os problemas relacionados com a arquitetura monolítica [4]. Além disso, há imensa complexidade para ser decomposta em comércio eletrónico, e podem ser identificadas algumas áreas distintas, tais como: inventário, catálogo, carrinho, *fulfillment* de encomendas, *shipping*, e pagamentos [5].

Por outro lado uma implementação de micro-serviços resulta usa, em geral, métodos de comunicação assíncronos, o que pode levar a um atraso na forma como a informação se espalha no sistema [6]. O comércio eletrónico lida com alguns problemas que necessitam geralmente de um sincronismo imediato, como o inventário e preços [7], pois geralmente o negócio não tolera incertezas nessas áreas.

1.2 Problema

A Kodly é uma consultora da área do comércio eletrónico e pretende com este trabalho estudar os custos e os benefícios que o estilo arquitetural de micro-serviços pode trazer a um projeto na área do comércio eletrónico. A Kodly pretende compreender a partir de que nível de complexidade e de volume de tráfego os micro-serviços são uma escolha viável, para melhor poder aconselhar os seus clientes.

O cliente da Kodly pretende criar um produto que se pode definir como um *marketplace* de prestação de serviços. Tem como objetivo final expandir para vários países africanos de forma a dominar o mercado emergente no continente. Torna-se assim importante o requisito não funcional da escalabilidade.

A Kodly teve oportunidade de implementar esse projeto desde a sua raiz e de o moldar. Inicialmente analisou-se a possibilidade de usar alguma *framework* já existente no mercado, mas optou-se por uma implementação de raiz de um monólito, deixando sempre em aberto a possibilidade de o transformar numa implementação de micro-serviços. Entre setembro de 2020 e fevereiro de 2021 o autor do documento esteve envolvido na implementação do monólito que satisfaz os requisitos funcionais do cliente.

O problema neste momento é desconhecer-se os impactos que o avanço para uma arquitetura de micro-serviços teria no sistema, e se traria sequer benefícios para o projeto.

1.3 Objetivos

Pretende-se implementar este caso de estudo na arquitetura de micro-serviços para avaliar o impacto da transformação. Para isso é pretendido estudar alguns fatores como:

- Qual é o aumento total de tamanho e complexidade.
- Qual é o ganho de performance.
- Qual é o impacto do uso de métodos de comunicação assíncronos.

1.4 Contribuições para a empresa

Um dos clientes da Kodly pretende implementar um *marketplace* para publicação e contratação de serviços. Os requisitos principais são:

- Implementar um novo *marketplace* para publicação e contratação de serviços.
- Ser a primeira empresa a oferecer este tipo de *marketplace* no mercado africano para se tornarem na plataforma dominante nos países da África.
- Ter a habilidade de fazer essa expansão sem a plataforma ter problemas ao lidar com o aumento do número de utilizadores.
- Tudo deve ser implementado com um orçamento limitado.

O objetivo deste *marketplace* é oferecer uma plataforma para negócios publicarem os seus serviços (canalizadores, limpezas, jardineiros, explicadores, serviços financeiros, etc), aqui chamados de fornecedores; e também, para o público em geral poder pesquisar pelos serviços e contactar os fornecedores para contratar os seus serviços. O modelo de negócio é cobrar uma subscrição aos fornecedores de serviços para eles poderem publicar os seus contactos pela plataforma.

Como ainda não existe na Kodly nenhuma experiência em primeira mão de uma análise de custos e benefícios do uso de micro-serviços, é importante que esse primeiro estudo seja feito de forma cuidadosa. Este projeto não é tão complexo quanto um projeto de comércio eletrónico normalmente é, no entanto tem muitas similaridades o que o torna interessante para fazer esta análise.

Assim a contribuição principal para a Kodly é a possibilidade de extrapolar os resultados deste trabalho para outros projetos no futuro, de maneira a que consiga melhor aconselhar os seus clientes sobre qual das duas arquiteturas será preferencial para o seu caso. Uma contribuição secundária seria que esta nova versão em micro-serviços seja mais adequada para o caso em estudo.

1.5 Abordagem

A abordagem tomada para esta investigação enquadra-se com a metodologia pesquisa-ação [8] primeiro foi planeada a transformação a fazer, de seguida esse plano foi aplicado e por fim o resultado foi avaliado. Foi, no entanto, feita apenas uma iteração da metodologia.

1.6 Estrutura do documento

Este documento encontra-se dividido em 9 capítulos:

1. **Introdução:** É introduzido o contexto, o problema que se está a tentar resolver, bem como os objetivos que se pretende atingir para resolver o problema.
2. **Estado da arte:** É feita uma revisão da literatura sobre alguns tipos de ferramentas de comércio eletrónico, sobre arquitetura monolítica e sobre a arquitetura de micro-serviços.
3. **Análise de valor:** É feita uma análise sobre o valor que este projeto gera para o cliente.
4. **Avaliar soluções e abordagens existentes:** É aplicado o processo de decisão chamado *Analytic Hierarchy Process* para decidir entre um dos tipos de ferramentas de comércio eletrónico e uma implementação de raiz.
5. **Análise:** É feita uma análise dos requisitos funcionais e não funcionais do cliente e do modelo de domínio que foi criado durante a implementação do monólito.
6. **Design da solução:** É exposta arquitetura do monólito implementado usando o modelo C4 de documentação para Arquitetura de Software, e é planeada a implementação em micro-serviços.
7. **Implementação da solução:** É feita uma explicação daquilo que foi implementado, e é exposta a arquitetura da solução final usando novamente o modelo C4.
8. **Avaliação:** A solução final é avaliada e comparada com o monólito, em resposta aos objetivos da dissertação.
9. **Conclusão:** São resumidamente referidas as conclusões gerais do documento.
10. **Bibliografia:** O conjunto de referências que foram usadas.
11. **Anexos:** Encontram-se alguns documentos em anexo que, apesar de não essenciais, dão mais contexto ao trabalho, tais como: o diagrama de componentes do monólito com todas as componentes, uma proposta da implantação do sistema de micro-serviços para um ambiente de produção, diagramas de classes e a REST API exposta em cada um dos micro-serviços, e, por fim, o relatório completo obtido a partir da ferramenta CodeMR sobre a complexidade do código em todos os micro-serviços.

Capítulo 2

Estado da arte

Neste capítulo é explorado o estado da arte das tecnologias de comércio eletrónico, e é também explorado o estilo arquitetural dos micro-serviços.

2.1 Comércio eletrónico

O comércio eletrónico pode ser definido como a compra e venda de bens ou serviços pela internet, e a acompanhante transferência de dinheiro. Os objetos de troca podem ser bens físicos, bens digitais, serviços, subscrições ou até *crowd funding* pode ser considerado comércio eletrónico [9].

Nesta secção são identificados três tipos de ferramentas principais para comércio eletrónico, que são classificadas como: *frameworks*, *Software-as-a-Service* (SaaS) extensíveis, e outras plataformas. Para cada um deles são dados alguns exemplos, mas não se procura criar uma lista completa.

Frameworks

Uma *framework* é, por si só uma aplicação. Contém já a maioria das funcionalidades básicas que a aplicação final irá necessitar. Diferem de bibliotecas no sentido em que há uma inversão de controlo: a *framework* usa o nosso código, e o nosso código é que usa bibliotecas [10].

A sua utilização poupa imenso tempo de desenvolvimento pois deixa de ser necessário repetir a implementação das mesmas funcionalidades básicas em todos os projetos, bem como de outras questões como por exemplo a segurança. *Frameworks* são também flexíveis e altamente personalizáveis [11].

De acordo com a experiência na Kodly, as *frameworks* de comércio eletrónico podem ser bastante complexas e o desconhecimento da equipa de desenvolvimento em relação às suas capacidades pode levar ao seu mau uso e a alguns problemas. Em projetos de ATG que o autor do documento auditou surgiram problemas como: configurações erróneas nos repositórios ATG que levavam a que todas as propriedades dos produtos fossem colocadas em *cache*, o que esgotava a memória do servidor e a aplicação falhava; ou, injeção de dependências usando introspeção, quando o ATG inclui o Nucleus para esse propósito [12].

Exemplos: Oracle commerce/ATG (Nucleus, Java) [13], SAP Commerce/Hybris (Spring, Java) [14], Demandware [15], HCL Commerce (NodeJS, Javascript) [16], Magento (PHP) [17]

Software-as-a-service (SaaS)

Um SaaS, ou software como um serviço, faz entrega de uma aplicação através da internet. Inclui como serviço a infraestrutura, o *middleware*, a gestão de dados, e o próprio processo de implantação da aplicação. Isso liberta o negócio dos custos de comprar, manter e escalar a infraestrutura, e liberta o negócio da responsabilidade de entregar o serviço aos clientes, diminuindo o risco do negócio [18].

De acordo com a experiência que temos na Kodly, geralmente o vendedor de SaaS decide que funcionalidades da *framework* subjacente podem ser estendidas e quais estão protegidas, pois querem sempre garantir a qualidade do serviço. Assim, troca-se a capacidade de personalizar a aplicação por uma diminuição do risco.

Exemplos (e frameworks subjacentes): Oracle CX Cloud (Oracle commece/ATG) [19], Salesforce Commerce Cloud (Demandware) [20], SAP Commerce Cloud (SAP Commerce/Hybris) [21]

SaaS em micro-serviços

Existe uma subsecção dos SaaS que oferecem um *backend* já dividido em micro-serviços. Oferecem bastante flexibilidade por permitirem escolher quais micro-serviços usar e estender [22].

Exemplos: Commerce Tools [23], Elasticpath [24]

Outras plataformas

Existem ainda outras plataformas que oferecem a possibilidade de criar *sites* de comércio eletrónico com o mínimo de programação, fazendo toda a gestão através da sua plataforma, sem a necessidade de uma equipa de desenvolvimento [25].

De acordo com a experiência que temos na Kodly, são as ferramentas mais baratas do mercado mas também as menos personalizáveis.

Exemplos: Shopify [26], Squarespace [27], Wix [28], BigCommerce [29], Shift4shop [30]

2.1.1 Custos

Nesta secção são comparados os custos das várias tecnologias. Cada organização tem um esquema de preços diferentes o que por vezes torna a comparação difícil e há parâmetros que variam com o tráfego recebido, mas o objetivo aqui é ter uma estimativa. A análise dos custos é importante pois o cliente tem um orçamento muito limitado, o que torna este fator dominante na possível escolha de uma ferramenta.

É possível verificar que existe uma grande diferença entre *frameworks*/SaaS e as restantes plataformas.

Frameworks

	Licenças	Variáveis com o volume de vendas	Implementação	Implantação
Oracle Commerce	500 mil $ inicial + 100 mil $/ano [31]			
SAP Commerce	54 mil $/ano [31]			
Demandware	400 mil $	$\sim$ 700 mil $/ano [32]		
Magento Open Source	0		30-100 mil $	1000$ [33]
Magento Enterprise	17900$/ano [31]	22-152 mil $/ano	> 60 mil $	7000$ [33]

Tabela 2.1: Comparação de estimativas de custos associados com *frameworks* de comércio eletrónico.

SaaS

	Licenças	Recorrentes	Implementação
Commerce Tools	200-500 mil $/ano		300 mil - 1 milhão $ [33]
Elastic Path		> 1995 $/mês [33]	
Oracle CXC	250 mil $/ano	175 mil $/ano	125 mil $ [34]
Salesforce CC	700 mil $/ano	1 a 3 % das vendas + 250 a 600 mil $/ano	400 mil $ [34]
SAP CC	54 mil $/ano		500 mil a 2 milhões $ [35]

Tabela 2.2: Comparação de estimativas de custos associados com SaaS de comércio eletrónico.

Outros

	Variáveis com o volume de vendas	Mensal
BigCommerce	> 700 $/ano (para 800 mil vendas)	30 a 250 $ [33]
Shopify	2,15 - 3% de cada transação	9 a 2000 $ [33]
Squarespace	3% de cada transação	24 a 36 $ [36]
Wix		17 a 35 $ [36]
Shift4shop		29 a 229 $ [37]

Tabela 2.3: Comparação de estimativas de custos associados com outros serviços de comércio eletrónico.

2.2 Arquitetura monolítica

Um sistema monolítico pode ser considerado um estilo arquitetural onde todas a funcionalidades e módulos de um sistema estão contidos numa única entidade, e geralmente corre num único processo em uma única máquina. Historicamente estes sistemas eram a norma, e eram consequência dos processos de desenho de software tradicionais [4].

Esta abordagem tem algumas vantagens, tais como [4]:

Facilidade de implantação Pode ser implantado como uma única unidade coesa.

Facilidade de testar end-to-end Os testes são feitos contra uma única aplicação.

Facilidade de debug O *debug* de um sistema auto-contido é mais simples que o de um sistema distribuído.

Facilidade de implementação É mais fácil implementar para novos projetos

Monólitos têm também desvantagens, e várias surgem do facto de terem baixa coesão e um acoplamento forte. Coesão é uma medida do quanto um módulo faz apenas uma coisa, e acoplamento é medida do quanto diferentes módulos necessitam de outros [4]. Esta abordagem torna-se limitadora quando, com o acumular de funcionalidades, a aplicação se torna muito grande e muito complexa. Algumas das suas limitações são [3]:

Desenvolvimento O desenvolvimento deixa de conseguir ser ágil. A aplicação é demasiado grande e complexa para que uma única pessoa a consiga conhecer toda; implementar novas funcionalidades torna-se mais difícil e demorado; e, a aplicação aproxima-se cada vez mais do anti-padrão *big ball of mud* [3].

Tempo de inicialização A aplicação demora cada vez mais tempo a iniciar, o que tem também um impacto negativo na rapidez de desenvolvimento e na produtividade da equipa. Para desenvolver é necessário reiniciar a aplicação várias vezes, por isso, passa-se cada vez mais tempo é passado à espera [3].

Entrega contínua A entrega contínua é afetada negativamente, por fatores como: longos tempos para iniciar a aplicação, o impacto das mudanças não ser facilmente compreendido; e, pela necessidade de grandes quantidades de testes manuais [3].

Confiabilidade A confiabilidade também pode ser negativamente afetada, pois como todos os módulos estão a correr dentro do mesmo processo, se houver algum erro toda a aplicação pode vir abaixo [3].

Dificuldade de adotar novas frameworks Se um monólito usa uma certa *framework*, mudar implicaria voltar a desenvolver tudo de raiz. Um monólito fica sempre preso a qualquer escolha de tecnologia feita no início do projeto [3].

Passado alguns anos, tem-se uma aplicação cujo sucesso é crítico para o negócio, e se tornou num monólito enorme, de baixa confiabilidade, escrito usando uma tecnologia já obsoleta para a qual é difícil recrutar programadores. O desenvolvimento torna-se lento e custoso [3].

2.3 Micro-serviços

Micro-serviços podem ser definidos de uma maneira muito simples:

Definição 2.3.1 (Micro-serviços) *Serviços pequenos e autónomos que trabalham em conjunto para atingir um objetivo [38].*

Isso não deixa de ser verdade, no entanto há muitas mais preocupações a ter em conta para se criar um sistema de micro-serviços com qualidade. As suas propriedades mais importantes são:

Responsabilidade única/Alta coesão

Robert C. Martin define em [39] o princípio de única responsabilidade como: Deve-se juntar aquilo que muda pela mesma razão, e separar o que muda por razões diferentes.

As razões para mudar vêm essencialmente do negócio. Se duas funções fazem operações relacionadas com diferentes áreas do negócio (ex: departamento financeiro e departamento de recursos humanos), então essas duas devem estar em classes diferentes [39].

Este mesmo princípio pode ser aplicado a nível dos micro-serviços. As fronteiras entre micro-serviços devem ser as mesmas fronteiras do negócio para tornar óbvio onde o código de cada funcionalidade deve ser implementada, e isso ajuda a manter os serviços pequenos [38].

Colocar funcionalidades relacionadas perto, e as não relacionadas longe, aumenta a coesão dos serviços; e, evita que uma modificação numa funcionalidade não tenha que ser feita em mais que um micro-serviço. Isso diminui o risco envolvido com as modificações e aumenta a rapidez do desenvolvimento [38].

Pequeno

Um micro-serviço deve ser pequeno, mas é impossível criar uma regra que defina o que significa pequeno, especialmente porque negócios diferentes têm necessidades diferentes. Regra geral, pode-se dizer que deve ser pequeno o suficiente para fazer o seu trabalho e não mais pequeno que isso [38].

Existe um *tradeoff* com a pequenez. Quanto mais pequeno for um micro-serviço, mais se maximizam os seus benefícios mas também as suas desvantagens. Ou seja, quanto mais pequeno, maiores as vantagens com a interdependência dos micro-serviços, mas também aumentam as complexidades de ter o sistema dividido em muitas partes [38].

Autónomo/Baixo acoplamento

Cada micro-serviço deve ser autónomo e dentro do possível ser implantado na sua própria máquina. Todas as comunicações entre micro-serviços devem ser feitas através da rede, de forma a forçar a separação e manter um acoplamento fraco. Isso significa que devem poder ser modificados e implantados independentemente dos outros e sem necessitar que os consumidores mudem. Para isso, a API deve ser cuidadosamente escolhida para manter o encapsulamento dos componentes internos do micro-serviço. Se não for possível modificar um micro-serviço sem modificar mais nada, então muitas das vantagens dos micro-serviços serão muito mais difíceis de atingir [38].

2.3.1 SOA vs micro-serviços

SOA (Service-Oriented Architecure) é um padrão arquitetural que pode ser definido de forma muito similar aos micro-serviços como sendo um conjunto de serviços que trabalham em conjunto para fornecer um conjunto de funcionalidades [38], no entanto não são a mesma coisa, por isso torna-se importante distingui-los.

SOA surgiu na década de 90 e é um marco importante na história da informática pois veio facilitar fortemente a integração de aplicações. SOA permite-nos organizar vários serviços e definir como eles devem comunicar (usando protocolos como SOAP ou HTTP) [40]. Na Figura 2.1 pode ser encontrado um exemplo de SOA.

Escala/Granularidade

A principal diferença entre SOA e micro-serviços é a escala a que trabalham. SOA organiza os serviços (historicamente monólitos) de toda uma empresa, enquanto que micro-serviços

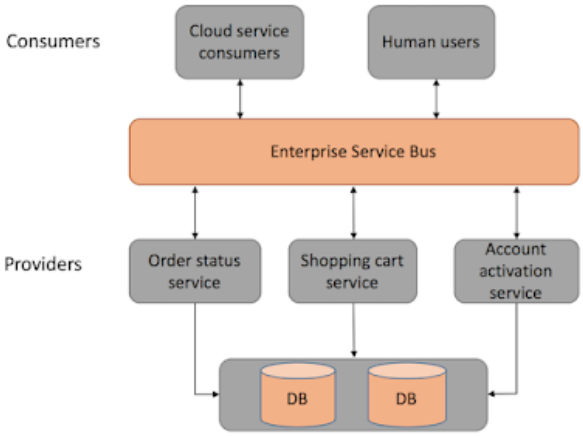


Figura 2.1: Exemplo de SOA: Os consumidores (fora da organização) comunicam com os serviços através de um ESB, os quais comunicam diretamente com a(s) base(s) de dados. Retirado de [41].

organizam apenas a estrutura de uma aplicação, e define como os micro-serviços comunicam (nunca como aplicações diferentes comunicam) [40]. Não competem por fazer a mesma coisa, podendo até completar-se [42]. Na Figura 2.2 encontra-se uma representação da diferença de escalas entre SOA e micro-serviços.

Sincronicidade das chamadas

Em SOA as chamadas são predominantemente síncronas [42]. Em micro-serviços fazer o mesmo resultaria numa perda de resiliência (pois existiriam dependências em tempo real entre micro-serviços), e aumentaria também a latência. Por isso, são preferidos métodos de comunicação assíncronos [42].

Duplicação de dados

Em SOA tem-se por objetivo centralizar os dados, e todas as aplicações alterarem numa fonte primária. Em micro-serviços, cada serviço terá a sua própria base de dados, o que obriga a repetição de dados e aumenta a complexidade [42].

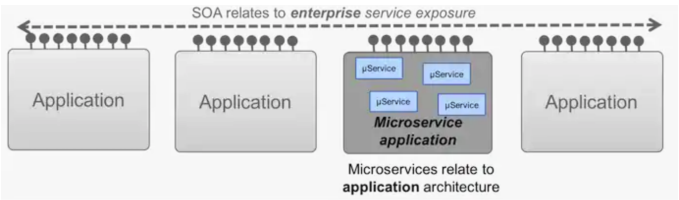


Figura 2.2: Comparação da granularidade de SOA e micro-serviços. Retirado de [42].

Comunicação

Em SOA cada serviço tem que usar um componente comum de comunicação, que é o ESB. Um abrandamento em algum serviço pode ter repercussões em todo o sistema. Em micro-serviços cada serviço é independente e pode ter o seu próprio protocolo de comunicações [42].

Interoperabilidade

Em micro-serviços pretende-se simplicidade e um protocolo leve para beneficiar a *performance*, como HTTP. SOA pode ser mais heterogéneo [42].

Agilidade e desacoplamento

Micro-serviços surgem com o aumento da virtualização, computação na nuvem, e DevOps. As suas grandes vantagens surgem como consequência do desacoplamento dos serviços. Podem ser testados, modificados e implantados independentemente dos ciclos de desenvolvimento do resto do sistema, o que oferece uma maior agilidade às equipas de desenvolvimento. Qualquer serviço pode ser replicado rapidamente na nuvem para responder a aumentos de carga, o que oferece uma melhor escalabilidade. O falhanço de um micro-serviço não afeta os demais, o que resulta num sistema mais resiliente [40], enquanto que em SOA podem haver componentes como o ESB que são um ponto único de falha [41].

2.3.2 Teorema CAP

Existe em informática o teorema CAP, que pode ser definido como:

Definição 2.3.2 (Teorema CAP) *Um sistema distribuído só pode ter duas destas três características: Consistência, Disponibilidade e Tolerância de partições [7].*

Estas características podem se descrever como [7]:

Consistência Todos os clientes recebem os mesmos dados, independentemente do nodo a que se conectam.

Disponibilidade Todas as chamadas recebem uma resposta (que não de erro), mesmo que alguns nodos estejam em baixo.

Tolerância de partições Uma partição é uma falha na comunicação entre dois nodos, ou seja uma perda, ou atraso na comunicação. Tolerância de partições significa que o sistema deve continuar a funcionar apesar de falhas na comunicação.

Os micro-serviços naturalmente originam um sistema distribuído, por isso este teorema torna-se importante no seu estudo.

2.3.3 Interfaces

É importante que todas as interfaces sigam o mesmo padrão, isto é, não basta dizer que é usado HTTP/REST, mas também definir como são usados os nomes e verbos, como é feita a paginação de recursos, como é feito o versionamento dos *endpoints*, etc. Todos esses fatores deverão seguir o mesmo padrão, ou o mínimo de padrões diferentes [38].

Uma API deve ser independente da tecnologia usada para a implementação (por exemplo, usar RMI força tanto o cliente como o servidor a usar JVM), e deve ser simples de consumir,

ou seja, deve ser minimizado o custo de consumir a API do sistema de micro-serviços. A API não deve mostrar os detalhes internos da implementação, para evitar que uma mudança interna implique uma mudança na API, de forma a beneficiar o encapsulamento e baixo acoplamento [38].

2.3.4 Comunicação

Com uma comunicação síncrona, quando uma chamada é feita ao servidor, há uma conexão bloqueante que fica a correr até a operação ser concluída, e ser dada uma resposta ao consumidor, que é informado se ela teve sucesso ou não. A este método de comunicação chama-se *request/response*, e pode ser implementada usando tecnologias como RPC ou REST [38].

Com uma comunicação assíncrona, a chamada é feita, mas não espera pela operação ser concluída. É especialmente útil no caso de operações demoradas; quando se necessita de uma latência baixa; e quando uma conexão aberta durante algum tempo pode resultar num atraso do sistema. Este método de comunicação diz-se ser baseado em eventos [38], e pode ser implementado usando um *message broker*.

Numa comunicação baseada em eventos, em vez do cliente ditar o que precisa de ser feito e ficar a espera de uma resposta, apenas tem que dizer que algum evento aconteceu, e as restantes partes do sistema saberão o que fazer. É um sistema naturalmente assíncrono, a lógica de negócio encontra-se mais distribuída em vez de centralizada, e favorece um acoplamento fraco. [38].

2.3.5 Orquestração vs coreografia

Em orquestração, a comunicação entre micro-serviços é centralizada num componente que guia o processo, e comunica com todos os micro-serviços, e os micro-serviços não comunicam diretamente uns com os outros. Isto é similar a um maestro a dirigir a sua orquestra [38]. Encontra-se um exemplo na Figura 2.3, onde se mostra um serviço central que chama três outros para completar as operações necessárias.

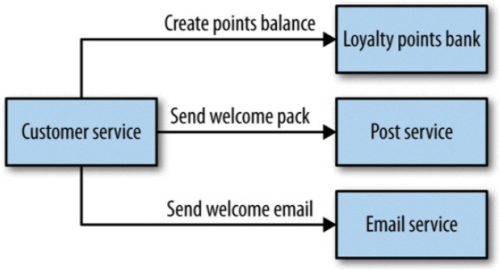


Figura 2.3: Exemplo de um sistema de registo de utilizadores via orquestração. Retirado de [38].

A desvantagem com esta abordagem é que a lógica pode acabar por se concentrar no nodo central, o qual passa a fazer chamadas a serviços que oferecem apenas operações CRUD. Estes sistemas têm tendência a tornar-se mais frágeis e a terem um maior custo de mudança [38].

Em coreografia, cada serviço é informado de eventos e sabe fazer o seu trabalho. Não existe um componente central e os serviços podem comunicar uns com os outros [38]. Um exemplo

semelhante ao anterior encontra-se na figura 2.4. Neste caso é publicado um evento, e os três serviços escutam o evento.

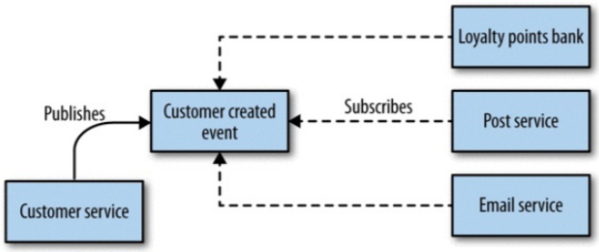


Figura 2.4: Exemplo de um sistema de registo de utilizadores via coreografia.
Retirado de [38].

Esta abordagem oferece um acoplamento mais fraco pois se for necessário introduzir mais algum micro-serviço para lidar com a criação de utilizadores, o novo serviço pode simplesmente subscrever aos eventos, e não é necessário modificar mais nenhum nodo [38].

Coreografia tem também as suas desvantagens. Uma delas é que a vista explícita do negócio que se encontra na figura 2.4 não se encontra explícita no sistema, apenas está lá implicitamente. Outra é que torna-se necessário algum trabalho adicional para monitorizar que as operações estão a acontecer corretamente [38].

Devido ao acoplamento mais fraco, a coreografia é uma opção atraente para implementar micro-serviços, apesar de representar uma maior complexidade devido à necessidade de introduzir um *message broker*. A possibilidade de mensagens serem pedidas, ou impossíveis de processar (devido a algum defeito no sistema) obriga a existência de um bom sistema de monitorização, e é útil também a utilização de *correlation IDs* [38] (identificadores de pedidos e mensagens que se referem a uma transação ou cadeia de mensagens em particular [43]).

2.3.6 Versionamento de APIs

A melhor forma de reduzir o impacto de mudanças na API é não as fazer, para não quebrar a integração entre serviços. É possível escolher tecnologias que minimizam essas mudanças, como REST, bem como usar certas estratégias ao ler dados de outro endpoint. Por exemplo, ao consumir uma API, não fazer *bind* de todos os campos da resposta, apenas dos que forem necessários. Isso deixa os restantes serem alterados e removidos da forma como o serviço consumido necessitar, e deixar o serviço consumidor inalterado [38].

O leitor de uma resposta deve ser o mais tolerante possível, desde que encontre o que necessita. Deve ser obedecida a lei de Postel (ou princípio de robustez) [38]:

Definição 2.3.3 (Lei de Postel) *Deve ser-se moderado com aquilo que se faz, e tolerante com o que se aceita de outros. [38].*

Dito isto, por vezes é mesmo necessário alterar uma API. Quando essas situações acontecem, devem ser detetadas o mais cedo possível para que todos os intervenientes possam ser alertados. Para diminuir o impacto não se deve obrigar nenhum outro serviço ou sistema a evoluir ao mesmo tempo, e deve versionar-se a API, e permitir que coexistam diferentes versões da API, ou diferentes versões do serviço. Dessa forma a migração para a nova API pode ser feita de forma gradual [38].

A identificação de qual API está a ser chamada pode ser feita nos *headers* do pedido, ou no URI, como por exemplo numa API REST: `/v2/user` [38].

2.3.7 Base de dados

Se todos os micro-serviços partilham a mesma base de dados, então todos têm que partilhar o mesmo *schema*, e todos têm que partilhar alguns detalhes internos. Além de se sacrificar uma boa porção de heterogeneidade tecnológica, uma alteração em um micro-serviço pode resultar em alterações em outros. A base de dados acaba por se tornar numa API grande e frágil, e resulta num maior acoplamento entre os micro-serviços e numa diminuição da coesão. Começa a ser difícil de introduzir mudanças e grandes números de testes têm que ser feitos, pois cada modificação envolve um maior risco. Este padrão nega muitas das vantagens dos micro-serviços e deve ser evitado [38].

Deve então seguir-se o padrão de uma base de dados por serviço, o que pode ser conseguido de várias formas [44]:

Tabelas-privadas-por-serviço Uma base de dados pode ter vários conjuntos de tabelas privadas, cada um dos quais só é visível para um micro-serviço.

Schema-por-serviço Uma base de dados pode ter vários schemas, cada um dos quais só é visível para um micro-serviço.

Base-de-dados-por-serviço Uma instância de servidor de base de dados por serviço.

As duas primeiras opções têm um menor custo à cabeça, no entanto, para serviços que recebem muito tráfego, a terceira opção será necessária [44].

Em alguma altura pode ser necessário reunir dados que estão separados em bases de dados diferentes. Nessas alturas é impossível fazer o *join* desses dados numa query à base de dados como se faria num monólito. Por isso surgem algumas desvantagens do padrão base de dados por serviço [44]:

- Transações que abrangem vários serviços são mais complexas e estão sujeitas ao teorema CAP.
- Fazer o join de dados em várias tabelas pode ser desafiante.
- Complexidade de gerir várias bases de dados que podem ser tanto SQL como NoSQL.

Existem alguns padrões que ajudam a lidar com isso, por exemplo para *queries* em várias bases de dados existem a composição de APIs (Secção 2.3.8) e o padrão CQRS (Secção 2.3.9); e, para transações em vários serviços existe o padrão Saga (Secção 2.3.10) [44].

2.3.8 Composição de APIs

O padrão de decomposição de APIs consiste em ter um serviço que recebe o pedido, e por sua vez invoca cada um dos micro-serviços a quem pertence os dados, e após receber as respostas, reúne-as na mesma estrutura para devolver ao consumidor [45]. Um exemplo encontra-se na Figura 2.5.

Nem sempre este padrão é aplicável, pois depende de fatores como: a forma como os dados são particionados; e, das capacidades tanto das APIs e das bases de dados. Por exemplo, pode ser necessário realizar uma operação ineficiente de um grande volume de dados causando latência [45].

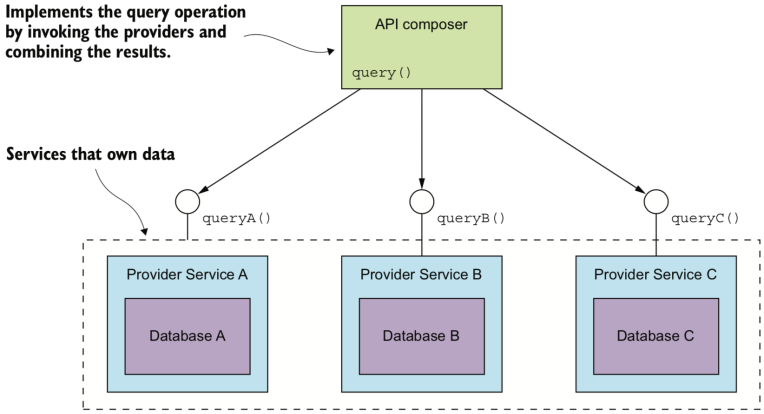


Figura 2.5: O padrão de decomposição de APIs. O elemento "API composer" invoca os outros serviços e combina os resultados. Retirado de [45].

Este é um padrão muito simples e fácil de aplicar, no entanto tem algumas desvantagens, tais como [45]:

Maior overhead Ao invocar vários serviços e bases de dados são necessários mais poder computacional e mais recursos de rede, especialmente em comparação com um monólito, em que o o mesmo é feito apenas usando um pedido e uma query à base de dados.

Risco de disponibilidade reduzida A disponibilidade baixa com o número de serviços envolvidos, pois basta que um deles esteja indisponível que o serviço fica todo indisponível. Uma forma de mitigar isso é, se possível, aceitar a devolução de dados incompletos.

Falta de consistência transacional No caso de um monólito, a transação com a base de dados é garantidamente ACID. Neste caso, são feitas várias *queries* a várias bases de dados, por isso há um risco de serem encontrados dados inconsistentes, no caso das *queries* serem feitas ao mesmo tempo que se está a espalhar uma atualização pelas bases de dados do sistema.

2.3.9 CQRS

O padrão *Command query Responsibility Segregation* (CQRS) assenta-se na ideia de separar comandos (ações que modificam dados) de *queries* (ações de consulta, sem efeitos colaterais). Esta ideia surgiu no contexto das linguagens orientadas a objetos [46], mas pode ser aplicado outras situações, tais como micro-serviços; e, permite tirar mais vantagem de padrões arquiteturais como *Event sourcing* [47].

Event sourcing é um padrão arquitetural em que se procura guardar todas as mudanças de estado como uma sequência de eventos. Isto cria um histórico de todos os eventos, que foram ocorrendo, o que permite operações como: se em alguma altura for necessário descartar o estado atual da aplicação, voltar a correr sequencialmente os eventos para voltar a obter o mesmo estado; fazer *queries* temporais, para saber o estado em algum ponto no passado; e, corrigir algum evento incorreto, e voltar a correr os eventos para obter o estado atual correto [6].

Um sistema que comece por apresentar apenas as operações CRUD, com a introdução de novas funcionalidades vai ficando cada vez mais complexo, e começam a haver várias

representações dos dados. Então quando os utilizadores interagem como sistema acabam por usar várias representações dos mesmos dados. Internamente isto pode resultar numa estrutura complexa de várias camadas de representação. O que CQRS introduz é a ideia de existirem modelos diferentes para *queries* e para comandos [47].

Os dois modelos podem ser mapeados para a sua própria base de dados, e comunicam entre si usando *event sourcing*, e dessa forma a base de dados do modelo *query* pode receber as atualizações. O modelo *query* é mais simples que o de comando pois não tem que lidar com lógica de negócio, e pode usar a base de dados mais apropriada para o tipo de *queries* implementadas [45]. Encontra-se um exemplo na Figura 2.6.

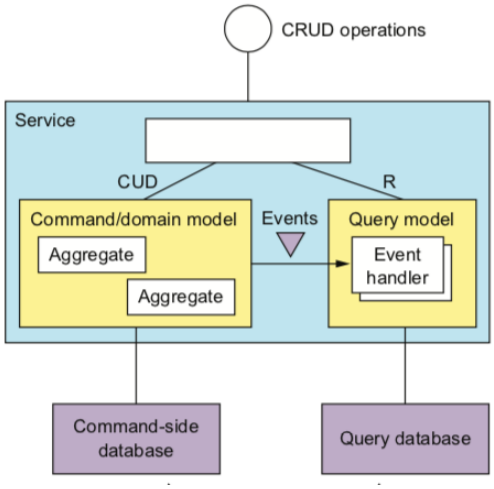


Figura 2.6: Exemplo de CQRS, onde existem dois módulos num serviço, um com as operações de modificação e o outro com as operações de leitura. Cada um com o seu modelo e a sua base de dados, e comunicam entre si através de mensagens. Retirado de [45].

Existem vários benefícios com esta abordagem em micro-serviços, tais como:

Implementação de queries eficientes Um serviço de *query* pode subscrever a eventos de vários serviços de comando diferentes e assim juntar na sua base de dados todos os dados que necessita. Não há a necessidade de juntar dados em memória de várias fontes, como no caso de composição de APIs [45].

Implementação de queries diversas Evita a limitação de ter uma única base de dados, e permite ter bases de dados especializadas para cada tipo de operação [45].

Permite queries no padrão event sourcing Uma das principais limitações de *event sourcing* é permitir apenas fazer *queries* pelo identificador. Ao subscrever aos eventos de várias fontes, CQRS permite criar uma vista de agregados de dados, e assim permite fazer *queries* mais complexas, superando essa desvantagem de *event sourcing* [45].

Melhora a separação de responsabilidades Ao criar a separação, os lados de comando e o de *query* ficam geralmente mais fáceis de manter [45]. Essa separação facilita também a escalabilidade ao permitir as duas responsabilidades, e assim escalar as duas independentemente no caso de aplicações com grandes disparidades entre leituras e comandos [47].

Apropriado para domínios complexos Os domínios complexos, que podem ser decompostos em *bounded contexts* através da aplicação de Domain-Driven Design (DDD), são

os que mais têm a ganhar com esta decomposição entre comando e *query* [47], pois DDD permite identificar os *bounded contexts* onde se pode aplicar CQRS [48] (*bounded contexts* são um contexto para o qual se aplica uma certa parte do modelo, a qual deve ser estritamente consistente dentro das suas fronteiras [49]).

Tal como qualquer padrão, tem as suas desvantagens:

Mais complexidade É necessário implementar e manter mais serviços e bases de dado, o que torna o sistema mais complexo [45]. Há sistemas que encaixam bem com um modelo CRUD, e nesses casos este aumento de complexidade pode ser um preço demasiado elevado a pagar pois o benefício será pequeno. Assim, deve apenas ser aplicado nos *bounded contexts* em que esta decomposição se justifique [47].

Atraso na replicação Como a comunicação entre as duas partes é assíncrona, haverá um atraso entre a modificação e a sua publicação. O consumidor do serviço pode ter que ser propositadamente desenhado para esconder estas inconsistências do utilizador. Uma solução pode passar por informar o consumidor da versão da entidade modificada de forma a ser possível compreender quando a versão do serviço *query* está desatualizada, e ir consultando até estar atualizada [45]. Isso pode causar uma sobrecarga, e deve ser evitado, preferindo notificar as partes interessadas [50].

O atraso na replicação pode levar a certas más práticas que devem ser evitadas, como: implementar a lógica de negócio também do lado do consumidor para não ter que se esperar pela confirmação [45]; assumir que o comando correu certo sem esperar por qualquer validação, fingindo assim uma consistência forte; forçar o bloqueio da UI enquanto se espera; forçar uma transação distribuída bloqueante; ou devolver a entidade diretamente a partir do serviço do comando [50].

Este atraso entre a modificação e a sua visibilidade na consulta, significa que a funcionalidade como um todo será eventualmente consistente. Aqui encontra-se a ideia de que o sistema convergirá num valor quando não forem recebidas mais modificações em rápida sucessão. Mesmo que o período de inconsistência seja normalmente curto, é possível que a fila de mensagens fique cheia em alguma altura de elevado tráfego (como por exemplo no pico de vendas durante as promoções, num sistema de comércio eletrónico) [48].

Escolher introduzir consistência eventual numa funcionalidade chave do negócio pode ter consequências terríveis. Há realmente casos em que a disponibilidade é a propriedade necessária, mas há os casos em que consistência é a propriedade mais importante, onde pode ser melhor não tomar nenhuma decisão do que tomar uma baseada em informação desatualizada. Isto deve ser sempre questionado ao escolher CQRS com *event sourcing*, pois essa decisão acarreta sempre risco, e deve ser apenas usadas para certos *bounded contexts* apropriados a isso, nunca a todo o sistema [48].

Em conclusão, CQRS pode ser uma ferramenta interessante para se ter na algibeira, no entanto é necessário ter cuidado com a sua implementação, pois aplicá-lo no sítio errado pode levar a um aumento desnecessário de complexidade, e introduzir mais risco no projeto [47].

2.3.10 Padrão Saga com transações ACID

O padrão saga serve para manter a consistência sem ter que recorrer ao uso de transações distribuídas, e trata-se de uma sequência de transações ACID locais, onde cada uma atualiza dados apenas dentro de um serviço. A conclusão de cada uma delas espoleta a próxima, e

todos os passos devem ser concluídos mesmo que algum dos participantes esteja indisponível [45]. Encontra-se um exemplo de uma saga na Figura 2.7, onde se pode ver um conjunto de transações que são feitas uma de cada vez, dentro de um conjunto de micro-serviços.

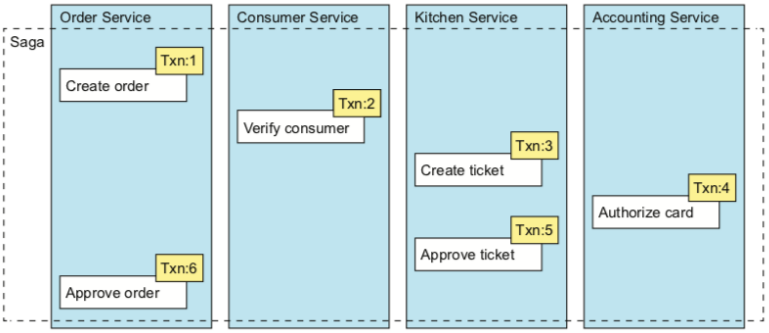


Figura 2.7: Exemplo de uma Saga, onde se podem ver 6 transações (Txn:1-6), que vão sendo executadas em 4 micro-serviços diferentes. O tempo corre de cima para baixo, como num diagrama de sequência. Retirado de [45].

Em transações ACID é possível reverter a transação caso se verifique que são violadas as regras de negócio. Numa saga é impossível fazer isso porque as transações anteriores já fizeram o *commit* nas suas bases de dados locais. Nesse caso devem ser criadas transações que compensem o que foi feito [45].

A implementação de uma saga consiste na lógica que coordena os passos da Saga, incluindo os passos de reversão, se necessário. Essa lógica pode ser coordenada por orquestração ou por coreografia [45].

Apesar de cada transação dentro da saga ser ACID, a saga no seu todo é apenas ACD, faltando-lhe o isolamento. Isto acontece porque mal é feita uma transação num dos serviços, a modificação fica logo visível para o resto do sistema. Quando alguma transação lê ou escreve dados de uma forma que não aconteceria se houvesse isolamento acontece o que se chama de uma anomalia, que pode ser de três tipos [45]:

Modificações perdidas Uma saga escreve modificações sem ter lido as de outra saga.

Leituras antecipadas Uma transação lê dados modificados antes que uma saga tenha terminado todas as suas modificações.

Leituras não repetíveis Dois passos de uma saga fazem a mesma leitura, mas obtêm dados diferentes por já terem sido modificados por outra saga.

É necessário implementar sagas de maneira a eliminar ou minimizar o impacto das anomalias no negócio. Para isso há várias medidas que podem ser tomadas [45]:

Lock semântico É uma *flag* que pode ser deixada numa entidade para indicar que alguma transação não concluída a modificou ou pode ainda modificar. Pode ser usado para impedir outras transações de usarem a entidade ou para servir de aviso. Este método essencialmente garante o isolamento de uma transação ACID, e as sagas que alteram o mesmo item são serializadas. A desvantagem é ter que gerir *locks*,e implementar alguma monitorização de *deadlocks* que consiga reverter e voltar a correr alguma saga que tenha ficado em *deadlock*.

Atualizações comutativas As atualizações podem ser desenhadas para serem comutativas (poderem ser feitas em qualquer ordem). Destas forma não há maneira de escrever por cima de outra modificação.

Vista pessimista Reordena os passos da saga para evitar o risco para o negócio de uma leitura antecipada.

Reler valores Um valor deve reler-se primeiro e verificar-se que ele não mudou, e só depois ser modificado. Se o valor tiver sido modificado a saga aborta e reinicia.

Ficheiro de versionamento Regista as operações que foram sendo feitas numa entidade, de maneira a que possam ser reordenadas, e executadas na ordem correta. É uma forma de tornar operações não-comutativas em operações comutativas.

Por valor É uma estratégia de selecionar mecanismos de concorrência baseado no risco do negócio. São usadas as propriedades de cada pedido para decidir entre usar sagas e transações distribuídas. Os pedidos de baixo risco são feitos usando sagas, e os de alto risco com transações distribuídas.

Uma ou mais destas medidas podem ser usadas ao implementar sagas [45].

2.3.11 Padrão Saga com transações BASE

Enquanto que transações ACID são imediatas, bloqueantes e úteis para garantir consistência. Transações BASE são assíncronas, não bloqueantes e úteis para garantir disponibilidade elevada, sacrificando a consistência. BASE significa [51]:

- **B**asically **A**vailable: disponibilidade elevada, replicando-a em muitos nodos do sistema.
- **S**oft-state: o estado pode mudar mesmo mesmo sem solicitação, e a consistência fica ao encargo do programador.
- **E**ventual consistency: a consistência não é garantida imediatamente, mas o sistema atingi-la-á em algum tempo.

As Sagas que usam transações BASE têm essencialmente as mesmas características que as que usam ACID, com a desvantagem acrescida de não garantirem a consistência. Tornam-se úteis, no entanto, para situações em que a disponibilidade é mais importante que a consistência [51].

2.3.12 Implantação

A implantação de um sistema de micro-serviços é mais complexa que a de um sistema monolítico, por isso tem que ser feita com algum cuidado, usando técnicas como *Continuous Integration* (CI) e *Continuous Delivery* (CD) [38].

CI O objetivo é verificar que as modificações de todos os programadores sincronizam corretamente com o código existente. Para isso, o servidor de CI tem que ficar à escuta que novo código seja colocado no repositório, e corre algumas verificações contra esse código. Tem o benefício de devolver informação aos programadores num curto ciclo para que problemas possam imediatamente ser resolvidos [38].

CD O objetivo é criar continuamente um artefacto que pode ser promovido para o ambiente de produção. Para isso, o servidor de CD corre um *pipeline* (uma *build* com um conjunto de passos) a cada vez que alguma modificação do código é detetada, que

passa por várias fases de teste, e devolve constantemente informação sobre a prontidão com que se pode promover o código mais recente a produção. O código vai passando por essas fases, e tendo sucesso, avança para a seguinte, podendo haver alguma fase manual. Isto permite aumentar a visibilidade da qualidade do código e diminuir o tempo entre *releases* [38].

Para aplicar CI a micro-serviços é necessário pensar como se pretende mapear os micro-serviços às *builds* e ao código fonte. Cada micro-serviço deve poder ser modificado, testado e implantado independentemente dos outros. Uma forma de conseguir isso é tendo o código de cada micro-serviço no seu próprio repositório e uma *build* de CI para cada um deles [38].

Para aplicar CD a micro-serviços é necessário criar um *pipeline* por cada micro-serviço. O objetivo é criar um artefacto que possa ir passando as várias fases de teste até chegar a produção. O artefacto pode ser muito diferente dependendo do projeto e da tecnologia usada, e uma mistura de tecnologias pode ainda dificultar a tarefa [38].

Os artefactos podem ser executáveis, ou podem precisar de ser lançados usando um servidor aplicacional, ou podem ser artefactos de sistema operativo (como rpm, deb ou msi). Ferramentas de automação de configurações (como Chef, Puppet ou Ansible) são úteis para instalar as dependências necessárias para correr os artefactos, mas podem demorar algum tempo para correr num sistema distribuído com muitos nodos, e é necessário cuidado para evitar que se manifestem diferenças de configurações ao longo do tempo [38].

Para mitigar esse problema pode usar-se uma ferramenta de virtualização (como Docker) que empacote o software e todas as suas dependências numa única imagem. Assim essa imagem é criada apenas uma vez, e pode ser reutilizada em vários ambientes ou nodos, e pode ser tratada como um artefacto. As diferenças entre tecnologias são abstraídas pelo facto de que tudo pode ser tratado como uma imagem. Todas as configurações devem estar guardadas no repositório, e quaisquer alterações devem ser feitas através do repositório [38].

Será necessário implantar os micro-serviços em vários ambientes distintos, ao longo dos quais cada micro-serviço é o mesmo, mas o ambiente serve um propósito diferente, por conseguinte, configurações diferentes. Essas configurações devem ser mantidas a um mínimo e devem fazer parte do processo de implantação [38].

Para isso, uma opção poderia ser construir um artefacto por ambiente, com as configurações já embutidas, mas isso vai contra o princípio de CD. Nunca nenhum teste teria sido corrido contra o artefacto de produção. O ideal é então construir um artefacto por micro-serviço e gerir as configurações de forma externa. Isso pode ser simplesmente um ficheiro *properties* em cada ambiente, ou pode ser gerido por um sistema dedicado [38].

Como distribuir micro-serviços por máquinas

Os micro-serviços podem ser mapeados por máquinas de várias formas. Nesta subsecção procura-se fazer uma análise de algumas estratégias que podem ser empregues.

Vários serviços em apenas uma máquina física Esta opção oferece bastante simplicidade, pois não é preciso gerir tantas máquinas, e os custos associados são menores. Por outro lado, a monitorização dos recursos da máquina pode ser mais difícil, e se um dos micro-serviços estiver sob grande carga pode reduzir os recursos disponíveis para os outros. A implantação pode ficar mais complexa pois é necessário garantir que a implantação de um dos micro-serviços não afeta os outros, e pode dar-se o caso que com a evolução dos serviços um deles passe a ter dependências contraditórias às dos

outros. As configurações da máquina têm que ser partilhadas entre os micro-serviços, e negociadas entre as equipas encarregues deles [38].

Um único serviço por máquina física Permite evitar as desvantagens de ter mais que um micro-serviço por máquina e reduz os pontos únicos de falha, pois se uma máquina estiver em baixo apenas um micro-serviço será afetado. Esta opção consegue diminuir bastante a complexidade, especialmente se não houver um sistema de PaaS (*Platform as a Service*) disponível. Por outro lado, fica-se com mais máquinas para gerir o que trás uma carga de trabalho maior e isso implica também um custo acrescido [38].

PaaS Permite um nível de abstração sobre as máquinas físicas, e há várias plataformas deste tipo que possuem ferramentas que permitem automaticamente implantar e correr o artefacto. Podem também oferecer algumas opções de em quantos nodos os micro-serviços devem correr, no entanto, a maior parte da complexidade encontra-se do lado da plataforma. Por outro lado, uma PaaS pode não se ajustar bem ao projeto por não suportar o tipo de tecnologia ou ter quaisquer outro tipo de limitação [38].

Várias VMs (Máquinas virtuais) por máquina física Permite poupar custos ao ter vários *hosts* que partilham a mesma camada física, no entanto, a própria VM não é grátis e consome também recursos. Tem a vantagem que cada VM é um sistema isolado, mas o seu *overhead* significa que quanto mais dividimos uma máquina menores são os benefícios [38].

Vários containers por máquina física Em vez de *hosts* virtuais separados para cada micro-serviço, os *containers* criam um processo na máquina onde corre o micro-serviço e as suas dependências em isolamento dos restantes. Tem o benefício de necessitar de muitos menos recursos para correr que uma VM, e inicializa muito mais rápido. O Docker permite criar *containers* leves que escondem a tecnologia subjacente. É apenas necessário criar as *builds* para as imagens Docker, e armazená-las no registo do Docker. É como se fosse uma espécie de PaaS que funciona em apenas uma máquina. Para gerir serviços em várias máquinas usando Docker, será necessário o uso de outras ferramentas [38] que fazem orquestração de *containers*, como Docker Swarm, Kubernetes ou Apache Mesos [52].

2.3.13 Testes

Testar corretamente e eficientemente a funcionalidade de uma aplicação é uma tarefa que só por si é normalmente complexa. Com micro-serviços adiciona-se ainda mais em toda uma camada de complexidade. Compreender os tipos de testes que podem ser feitos é importante para gerir as necessidades de testar o software corretamente e ter ciclos de *release* curtos [38].

Na Figura 2.8 encontram-se especificados alguns *scopes* diferentes de teste. De cima para baixo na pirâmide eles são: de Interface de Utilizador (UI), de Serviço e Unitários. Quanto mais para cima na pirâmide, mais alargado é o scope dos testes, e mais componentes do software são testados. Quanto mais para baixo, menor é o scope, e os testes são feitos com mais isolamento [38].

A forma da pirâmide indica também que os níveis inferiores da pirâmide devem ter mais testes que os superiores. Regra geral, de cima para baixo, a cada nível sucessivo da pirâmide o número de testes deve aumentar uma ordem de grandeza [38].

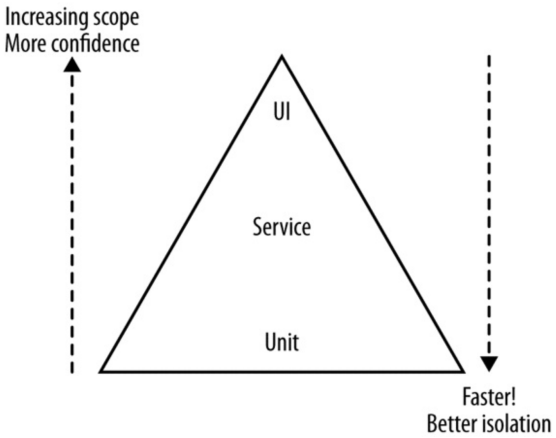


Figura 2.8: *Scopes* de teste. Retirado de [38].

Testes unitários

Testam geralmente apenas a chamada de uma função em isolamento dos restantes e sem inicializar qualquer serviço ou chamar qualquer serviço externo (sem efeitos colaterais, ou muito limitados). Devem ser feitos em grande número, mas devem ser executados rapidamente (aos milhares por minuto) [38].

Estes testes devem permitir encontrar a maioria dos defeitos e dar um ciclo de *feedback* curto. São especialmente úteis ao fazer *refactoring* e reestruturações do código pois permitem verificar que a funcionalidade continua intacta [38].

Testes de serviço

Ignoram a UI e focam-se em testar diretamente os serviços. Em micro-serviços têm o objetivo de testar cada serviço em isolamento, e todas as chamadas a outros serviços devem ser substituídas por *mocks*. O seu *scope* será um pouco mais alargado que o dos testes unitários e é natural que os testes demorem um pouco mais a correr, especialmente se for testado contra uma base de dados real [38].

Os *mocks* podem ser feito criando micro-serviços que respondem aos pedidos com respostas pré-definidas, e encaminhar para lá os pedidos gerados durante os testes de serviço [38].

Testes end-to-end (de ponta a ponta)

Este tipo de testes são feitos contra todo o sistema, e podem ser feitos por exemplo contra a UI, em que um sistema automatizado imita a interação de um utilizador com o sistema [38].

Estes testes têm o *scope* mais alargado pois correm contra vários micro-serviços ao mesmo tempo. Como abrangem mais partes do sistema têm a tendência a ser mais quebradiços, devolvem o ciclo de *feedback* mais lento, pode ser mais difícil perceber exatamente qual foi a funcionalidade que quebrou quando eles falham. Por outro lado, quando passam, há um grande grau de confiança que o código testado vai funcionar em produção [38].

A sequência de testes na *pipeline* de CD deve primeiro correr os testes unitários e de serviço contra cada serviço, e, após todos esses passarem os testes *end-to-end* devem ser lançados contra todo o sistema. Esta estratégia, porém, tem as suas desvantagens [38].

Uma falha nos testes *end-to-end* pode não significar necessariamente uma quebra na funcionalidade, mas pode significar outros problemas tais como uma falha na rede, *race conditions*, ou um *timeout*. É essencial eliminar testes traiçoeiros que falham num lançamento dos testes, e passam no seguinte. Se houverem vários *commits* num curto intervalo de tempo é possível que escondam funcionalidades quebradas, e no fim do dia, tem-se uma *build* quebrada sem se saber exatamente qual foi o *commit* que a quebrou. A *build* não deve nunca estar em estado de falha. Quando estiver, corrigi-la deve ser a prioridade e essa responsabilidade deve estar claramente definida [38].

Também é importante que estes testes não demorem demasiado tempo a correr, de forma a não impactar o tempo de uma *release*. Quando estiverem a demorar demasiado tempo é necessário investir algum tempo em torná-los mais eficientes e verificar que não há sobreposições na cobertura. Se os testes demoram muito tempo, podem se acumular vários *commits* depois de uma falha, e dá-se um acumular de erros [38].

Para evitar o crescimento desmedido do número de testes é necessário pensar exatamente quantos testes *end-to-end* é necessário realmente fazer. Deve-se evitar a tentação de adicionar um novo teste *end-to-end* a cada história, e testar apenas algumas cadeias relevantes de passos que o utilizador pode tomar [38].

Testes Consumer-Driven (orientados ao consumidor)

Este tipo de testes tem o objetivo de garantir que mudanças em um micro-serviço não quebram os seus consumidores, definindo quais são as expectativas dos consumidores e verificando que o serviço não quebra esse contrato. Este tipo de teste pode ser incluído na *build* de CI. Devem ser executados contra um serviço em isolamento dos restantes, encontrando-se ao mesmo nível na pirâmide da Figura 2.8 que os testes de Serviço. Existem ferramentas, como Pact, que permitem gerar a cada *build* uma especificação JSON daquilo que o consumidor espera receber, o que facilita a tarefa de criar teste Consumer-Driven [38].

Testes após produção

Ao fazer uma *release* para produção não se deve simplesmente acreditar que tudo correu bem. Utilizando um *blue/green deployment* (onde há duas cópias do ambiente de produção, mas apenas uma delas se encontra exposta a pedidos reais) é possível fazer o *deployment* para aquela que não está exposta, e correr alguns *smoke tests* contra esse ambiente. Devem ser apenas alguns testes simples e rápidos para verificar que o *deployment* correu bem [38].

Esta estratégia tem algumas desvantagens pois é mais trabalhosa e requer o dobro das máquinas em produção, no entanto, permite reduzir o risco do *deployment*, permite ter a possibilidade de reverter se não correr bem e de fazer um *deployment* com zero tempo de indisponibilidade do sistema [38].

É possível ir mais longe com uma *canary release*, onde ambas as versões do software podem coexistir durante um certo período de tempo. Isso permite comparar os resultados das duas versões e fazer mais testes na nova versão, tanto funcionais como não funcionais [38].

Testes não funcionais

As características não funcionais, por vezes, podem apenas ser testadas em produção. Estes testes deverão também seguir a pirâmide da Figura 2.8, por exemplo testes de carga serão

do tipo *end-to-end*. Devem ser implementados cedo no ciclo de vida do software, e revistos regularmente [38].

Testes de *performance* são especialmente importantes pois chamadas que envolveriam apenas um acesso à base de dados num monólito, em micro-serviços podem-se espalhar por mais que um micro-serviço e fazer outras tantas chamadas a bases de dados [38].

É conveniente ter uma mistura de granularidades de *scope* para estes testes, em que alguns podem ser feitos ao nível do serviço, e outros a *end-to-end*. No ambiente onde correm os testes de performance deve haver um volume de dados similar ao de produção, e uma infraestrutura ao mesmo nível. Uma das principais dificuldades será não apenas encontrar a raiz de um problema quando ele surgir, mas também identificar tanto os falsos positivos como os falsos negativos [38].

É normal que os testes de performance demorem bastante tempo para correr por isso não é exequível corrê-los a cada *release*. Devem ser corridos tão frequentemente quanto possível, e é necessário ter a disciplina de verificar os resultados e tomar ações quando não forem considerados bons [38].

2.3.14 Monitorização

A monitorização de um sistema que está dividido em muitas partes é mais desafiante pois existem muitos serviços para monitorizar, muitos ficheiros de log e muitos pontos onde pode ser introduzida latência no sistema. É necessário agregar toda a informação para conseguir ter visibilidade sobre todo o sistema [38].

Para isso é necessário um sistema especializado para ler todos os logs relevantes e disponibilizá-los todos no mesmo sítio. Uma ferramenta como Kibana permite ver logs, e até oferece uma sintaxe para executar *queries* nos logs e gerar gráficos [38].

Além dos logs, é necessário tirar outras métricas, como por exemplo, percentagem de CPU, ou o número de respostas de erro por segundo. Para saber que valores são preocupantes é necessário observar a evolução do sistema ao longo de algum tempo para reconhecer padrões. Além disso, é necessário observar métricas médias no sistema, tanto ao longo do sistema todo, como de todas as instâncias de um único serviço. Existem ferramentas como Graphite que ajudam com essa tarefa [38].

Ao conhecer os padrões de utilização do sistema, poderemos também saber quando se está a chegar ao limite, e planear adequadamente. Com serviços de *Infrastructure as a Service* (IaaS) é possível escalar a infraestrutura muito rapidamente, e assim ter sempre apenas a quantidade de infraestrutura suficiente [38].

Nalguma altura em que não haja tempo de conhecer os padrões do sistema pode usar-se a estratégia da monitorização sintética. Isto consiste em lançar alguns eventos falsos, que o sistema tem que conseguir reconhecer como falsos, mas ainda assim processá-los. Desta forma pode-se verificar o comportamento do sistema [38].

Métricas do serviço

Além de recolher as métricas do sistema operativo, cada serviço deve expor métricas, como por exemplo o número de respostas de erro 4xx e 5xx, tempos de resposta, número de vezes que os utilizadores fazem certas ações, ou número e montante de vendas (no caso de um sistema de comércio eletrónico) [38].

Uma melhor compreensão de como os utilizadores usam o sistema permitirá ter visibilidade sobre quais funcionalidades não são usadas e melhor conseguir reagir à maneira como os utilizadores usam o sistema [38].

IDs de correlação

Quando uma chamada é recebida num micro-serviço, e a partir daí são criadas várias chamadas a outros micro-serviços, é possível que falhem uma (ou mais) das chamadas subsequentes. Nesse caso, pode ser difícil rastrear nos logs de onde veio a chamada. Para facilitar essa tarefa podem usar-se IDs de correlação. Isso consiste em atribuir um ID à chamada inicial, e passá-lo a todas as chamadas subsequentes, para que todas os logs registados a partir daí sejam registados com esse mesmo ID [38].

O tratamento destes IDs deve ser padronizado por todo o sistema, e pode sê-lo feito com ferramentas como Zipkin. Quanto mais cedo no ciclo de vida isto for implementado melhor (especialmente se estiver a ser usada alguma arquitetura orientada a eventos), pois o custo de mudança é elevado [38].

Visibilidade da informação

É importante que, de toda a informação recolhida, a informação certa seja mostrada às pessoas certas. Por exemplo, os valores do uso de CPU devem ser mostrados a quem esteja encarregue da infraestrutura, mas as tendências de vendas a quem está do lado do negócio. Desta forma, cada parte da organização pode reagir imediatamente às alterações do sistema, o que é muito importante hoje em dia, dado que é possível fazer mais que uma *release* por dia [38].

2.3.15 Segurança

Nesta secção são analisadas algumas preocupações sobre segurança em micro-serviços.

Autenticação de utilizadores

Autenticação é o mecanismo com que se verifica que o utilizador é realmente quem diz ser (ex: par nome de utilizador e palavra passe, ou impressão digital). Autorização é o mecanismo com que se verifica se o utilizador está realmente autorizado a fazer a ação que pretende. Em sistemas monolíticos é normal que a própria aplicação tenha a lógica de autenticação e autorização, no entanto em micro-serviços isso torna-se mais complexo pois é necessário ter uma única identidade para cada utilizador que se autentica apenas uma vez, e não ter que autenticar o utilizador em todos os micro-serviços [38].

Single Sign-On (SSO) Quando um utilizador tenta aceder a algum recurso é redirecionado para se autenticar com algum *identity provider*. Após uma autenticação com sucesso esse *identity provider* devolve informação ao serviço para que possa decidir se dá acesso ao utilizador [38].

O *identity provider* pode funcionar de várias maneiras diferentes, como por exemplo, pode seguir o padrão SAML, que é baseado em SOAP e é bastante complexo; ou o padrão OpenID Connect, que é uma implementação específica de OAuth 2.0, e é mais simples, usando REST [38].

Em micro-serviços, se cada serviço tiver que saber redirecionar para o *identity provider* acaba por dar origem a muita lógica repetida [38].

Gateway de SSO Para evitar que todos os micro-serviços redirecionem para o *identity provider* essa responsabilidade pode ser unicamente de uma *gateway*, que consulta o *identity provider* e de seguida passa o pedido aos micro-serviços adicionando a identidade do utilizador na chamada [38].

Esta estratégia significa que toda a responsabilidade da autenticação está na *gateway*, a qual se torna num ponto único de falha. Ela pode trazer um sentimento falso de segurança, por isso é importante não esquecer que devem ser implementadas medidas de segurança em outros pontos do sistema [38].

É ainda necessário garantir que a *gateway* não acumula demasiada funcionalidade, pois torna-se num grande ponto de acoplamento, e, quanto maior for, também maior é a superfície sujeita a ataques [38].

Autenticação de serviços em serviços

Após um utilizador ser autenticado na camada da *gateway* o pedido é passado aos micro-serviços. Pode assumir-se que as chamadas dentro do perímetro do sistema são implicitamente confiáveis dado que já passaram essa camada de segurança, e, dependendo da sensibilidade dos dados, isso pode ser o suficiente. No entanto, o sistema fica vulnerável a ataques de *man-in-the-middle* [38].

Autenticação básica de HTTP(S) Permite enviar um par nome de utilizador e palavra passe num *header* HTTPS para que o servidor possa verificar que o cliente tem permissão para aceder ao serviço (deve ser usado HTTPS para que os dados sejam enviados de forma segura) [38].

O servidor terá que gerir o seu próprio certificado SSL, o que pode ser problemático ao lidar com muitas máquinas. Gerar certificados é uma carga de trabalho adicional significativa, e certificados auto-assinados não podem ser facilmente revogados, em caso de desastre. Além disso permite apenas conhecer as credenciais, e não há qualquer garantia que o pedido vem de alguma máquina esperada [38].

SAML ou OpenID Connect Já que já está a ser usado o *identity provider* ou *gateway* para autenticar a chamada inicial, ele pode ser reutilizado em cada chamada subsequente, e assim reutilizar infraestrutura existente. Para isso, cada serviço terá que ter a sua conta com as suas credenciais, as quais, terão que ser guardadas seguramente [38].

Certificados do cliente É possível utilizar as capacidades de Transport Layer Security (TLS), em que cada cliente tem um certificado que é usado para estabelecer a ligação entre o cliente e servidor, garantindo que o cliente é válido. A complexidade operacional em torno destes certificados é muito grande por isso são apropriados apenas para comunicação de dados muito importantes e sensíveis [38].

HMAC sobre HTTP Em HMAC (hash-based messaging code) o corpo do pedido e uma chave privada são codificados e enviados. O servidor usa a sua cópia da chave privada para reconstituir a mensagem e a chave. Se a chave for igual, então o pedido é aceite. Se houver um ataque *man-in-the-middle* a chave não vai corresponder. O *overhead* desta estratégia é geralmente menor que o uso de HTTPS [38].

Por outro lado, esta estratégia apresenta também alguns problemas. Assume que há uma chave secreta que tem que ser partilhada entre cliente e servidor, a qual tem que ser guardada seguramente, o que pode também dificultar a revogação de acesso, caso haja algum problema. Além disso, garante apenas que o pedido não foi manipulado, o resto dos dados no pedido serão ainda visíveis [38].

Chaves de API Permitem identificar quem está a fazer a chamada e colocar limites nas ações que podem executar. Os detalhes de implementação de chaves de API para comunicação entre micro-serviços dependerão da tecnologia usada. Pode, por exemplo, ser usada uma chave partilhada, com uma abordagem semelhante a HMAC; ou então, um par de chaves publica e privada [38].

As chaves de API são feitas para serem fáceis de usar por programas, e em comparação com outras estratégias como SAML, são muito mais simples de usar. É possível atribuir chaves a sistemas e utilizadores na organização, e controlar o seu ciclo de vida da mesma forma como se podem gerir credenciais normais [38].

Proteger os dados

Mesmo que uma terceira parte consiga passar por todas as defesas, apenas conseguirá ter acesso aos dados se estes forem legíveis. Para evitar isso é necessário codificar os dados. Há muitas estratégias de o fazer, e algumas podem conter falhas, nomeadamente, se se tentar criar de raiz um algoritmo de encriptação. Existem muitos algoritmos de fácil acesso, bem vistos pela comunidade, e regularmente mantidos. É sempre preferível usar um desses do que tentar reinventar a roda. Por exemplo, para codificar pode-se usar algo como AES-128 ou AES-256, e para passwords a técnica de *salted password hashing* [38].

Para codificar os dados é preciso gerar e guardar uma chave. Se a chave for guardada no mesmo sítio que os dados, a codificação é inútil. Elas precisam de ser guardadas em segurança. Pode-se usar qualquer aplicação separada para codificar e descodificar dados, ou então, guardar as chaves num cofre de chaves separado, ao qual o sistema pode aceder quando precisar. Existem bases de dados que suportam codificação [38].

Codificar tudo alivia o processo de decisão sobre o que precisa de ser codificado, no entanto gera uma quantidade de *overhead* significativa. Mesmo assim, é necessário pensar bem sobre aquilo que pode ser colocado em log. De maneira geral, os dados devem ser codificados assim que entram no sistema, e as cópias de segurança também o devem ser [38].

Outras medidas

É boa ideia ter várias *firewalls* que impeçam a comunicação entre máquinas que não precisam de comunicar, e até tê-las em redes separadas quando possível. Adicionalmente pode ter-se um sistema de deteção de intrusos, que fica à escuta de atividade suspeita e cria um log para a rastrear. O próprio sistema operativo e as ferramentas que usamos para correr a aplicação podem introduzir vulnerabilidades no sistema. Deve-se sempre correr o software usando utilizadores com o mínimo possível de permissões, e manter o sistema atualizado. O elemento humano também é extremamente importante na segurança de sistemas. Devem haver políticas em vigor para tentar diminuir o seu efeito, como por exemplo, expirar utilizadores e palavras passe quando um colaborador sai da organização, bem como medida para minimizar o impacto de engenharia social. Educar a equipa sobre segurança é também importante assim como, a realização de auditorias externas para avaliar o sistema [38].

2.3.16 Vantagens de micro-serviços

Os benefícios são vários e maioritariamente vêm do facto de micro-serviços serem um sistema distribuído [38].

Fortes barreiras modulares

Dividir o software em módulos desacoplados é sempre algo que deve ser feito, especialmente quanto maior for o tamanho do software. Colocar os módulos em serviços separados torna essas barreiras mais fortes. Além disso, o facto de cada serviço ter a sua base de dados elimina a necessidade de uma base de dados de integração (que serve mais que uma aplicação) [53].

Esta vantagem torna-se numa desvantagem se as fronteiras entre os serviços não forem colocadas no sítio certo, o que é um argumento forte para desenvolver sempre um monólito primeiro. Deve-se sempre compreender bem o domínio antes de avançar para uma estrutura de micro-serviços [53].

Heterogeneidade de tecnologias

Como os micro-serviços comunicam apenas através de APIs, é possível que cada um seja implementado com uma tecnologia diferente. Isso permite escolher a melhor ferramenta para cada trabalho em vez de ter uma tecnologia genérica para tudo. Por exemplo alguma parte do sistema pode ter certas necessidades de *performance*, ou podem haver certas necessidades diferentes para guardar os dados [38]. Encontra-se um exemplo na figura 2.9.

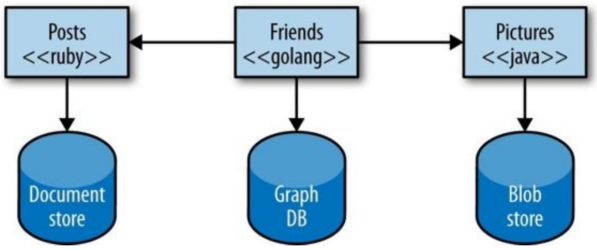


Figura 2.9: Exemplo de um *stack* tecnológico heterogéneo. Retirado de [38].

Outra vantagem da heterogeneidade é facilitar a adoção de tecnologias, pois a escolha de uma tecnologia experimental tem um impacto muito mais limitado e um micro-serviço é muito mais facilmente substituível se correr mal [38].

Resiliência

Uma falha num micro-serviço não cria uma cascata de falhas, e o resto do sistema pode continuar a funcionar normalmente [38]. Com as vantagens da virtualização um micro-serviço em baixo pode ser automaticamente e rapidamente re-instanciado.

Por outro lado são introduzidos mais pontos onde o sistema pode falhar, como por exemplo pode haver alguma falha na rede [38].

Escalabilidade

Num monólito, a escalabilidade é feita replicando todo o sistema, então a escalabilidade é limitada pela *performance*. Com serviços mais pequenos, é apenas necessário replicar aqueles que estão a receber mais tráfego, o que permite poupar bastante na infraestrutura [38].

Facilidade de implantação

Num monólito, a alteração de uma linha de código representa a re-compilação e re-implantação de toda a aplicação, o que representa um processo de grande impacto e alto risco. Em micro-serviços, cada um pode ser modificado independentemente dos restantes, e se algo correr mal, o processo de reversão é muito mais rápido. Isto diminui o risco e facilita a entrega de novas funcionalidades [38].

Alinhamento organizacional

Equipas mais pequenas são mais produtivas. Micro-serviços permitem melhor ajustar a arquitetura ao negócio, para minimizar o número de pessoas a trabalhar em cada micro-serviço, e assim maximizar a produtividade [38].

Composibilidade

Micro-serviços oferecem uma grande possibilidade de reutilização de código. A mesma funcionalidade pode ser consumida por diferentes componentes para diferentes propósitos. Isso torna-se especialmente importante se os serviços forem ser consumidos por aplicações de várias naturezas, como web, mobile, nativa, *wearable*, etc... [38]

Substituibilidade

Um sistema *legacy* é difícil de substituir. É um trabalho enorme e envolve um grande risco. Dada a simplicidade e pequenez de um micro-serviço é um trabalho muito menor e muito menos arriscado substituí-lo, desde que se mantenha a mesma API [38].

Também facilitam a substituição de um monólito *legacy*, permitindo fazê-lo pouco a pouco e diminuindo o risco de cada passo, até ao monólito eventualmente deixar de ser usado.

2.3.17 Desvantagens de micro-serviços

Não existem soluções perfeitas em informática, e micro-serviços não fogem à regra.

Consistência Eventual

Consistência eventual cria problemas de usabilidade, pois por vezes um utilizador pode fazer uma alteração, só para atualizar a página e vê-la desaparecer, e reaparecer pouco tempo depois. Isto pode causar confusão e frustração [53].

Por causa do uso de métodos de comunicação assíncronos, um sistema de micro-serviços não se torna adequado para um sistema em que a consistência imediata seja importante [7].

Isto leva alguns sistemas a terem que implementar *locks* [53], entre outros mecanismos para lidar com inconsistências.

Distribuição

Micro-serviços originam um sistema distribuído, logo, é um sistema mais complexo, e não se deve adotar de forma leviana. Uma dessas complexidades é o desempenho, pois quantas mais chamadas distribuídas pior será o desempenho. Há formas de mitigar isso como aumentar a granularidade das chamadas ou usar métodos de comunicação assíncronos, mas ambas opções também trazem as suas complexidades [53].

Outra das complexidades é a confiabilidade, pois uma chamada remota pode falhar a qualquer altura, e torna-se necessário desenhar o sistema para que possa lidar com essas falhas [53].

Complexidade operacional

A habilidade de implantar rapidamente pequenos serviços independentes é muito vantajoso para o desenvolvimento, no entanto trás uma complexidade adicional à área das operações, pois um pequeno número de aplicações pode-se transformar em centenas de micro-serviços. Gerir e monitorizar tantos micro-serviços torna a prática da entrega contínua fundamental para micro-serviços [53].

A simplicidade na compreensão do serviço e facilidade de desenvolvimento e manutenção são compensadas pelo respetivo aumento na complexidade operacional criado por um maior número de micro-serviços. Uma boa escolha nas fronteiras entre serviços ajuda a mitigar esse problema [53].

Esta complexidade operacional requer as capacidades e ferramentas necessárias para introduzir uma cultura de devops, isto é, uma maior colaboração entre desenvolvimento, operações, e toda a gente que estiver envolvida na entrega do software [53].

Crosscutting concerns

Torna-se mais difícil gerir as preocupações transversais a todo o sistema, tais como configurações externalizadas, *logging*, monitorização/*health checks*, entre outras [54].

Debugging

Ao analisar algum defeito ou erro no sistema, torna-se necessário fazer o *debugging* em cada um dos serviços separadamente, o que pode aumentar significativamente a carga de trabalho [54].

Maiores custos

Sem um planeamento cuidadoso, os custos dos serviços na nuvem pode aumentar rapidamente devido ao grande número de chamadas entre micro-serviços [54], e a equipa de desenvolvimento deve levar isso em consideração [55]. Micro-serviços requerem também mais recursos, pois cada serviço corre no seu próprio ambiente com o seu próprio CPU [55].

Segurança

Muitos serviços são expostos à rede, por isso há uma maior parte do sistema exposta a potenciais ataques. Além disso, como os micro-serviços são facilmente replicáveis, um pequeno defeito poderá ser copiado várias vezes. Assim, tornam-se necessárias as ferramentas e o treino corretos para minimizar as vulnerabilidades de segurança do sistema [55].

Testes

É necessário testar as interações entre os serviços, as quais se não forem testadas convenientemente podem tornar-se muito frágeis. Para isso pode ser necessário introduzir tecnologias especializadas para esse propósito [56].

2.4 Padrões de decomposição de monólitos

Quando se tem um monólito problemático pode surgir a necessidade de o substituir por um sistema dividido em micro-serviços. Esse novo sistema pode ser implementado de raiz ou então pode-se tomar partido do monólito, e ele decomposto em micro-serviços.

Existem mais que uma estratégia que podem ser utilizadas na decomposição de um monólito [57]. Nesta secção são analisados quatro padrões de decomposição de monólitos. Uma destas estratégias foi aplicada ao caso de estudo para decompor o monólito.

2.4.1 Decomposição por capacidade de negócio

Capacidades de negócio são as atividades que o negócio faz para gerar valor, como por exemplo: vendas, serviço ao cliente, marketing, etc. Geralmente, uma organização abrange várias atividades, e este padrão de decomposição pode ser usado quando a equipa tem boa visibilidade sobre essas atividades [57].

Vantagens [57]:

- Gera um conjunto estável de micro-serviços se as capacidades de negócio forem estáveis.
- As equipas de desenvolvimento estarão mais focadas em gerar valor para o negócio em vez de apenas entregar funcionalidades.
- Os serviços ficam mais fracamente acoplados.

Desvantagens [57]:

- O desenho da aplicação fica fortemente acoplado com o modelo de negócio.
- Requer um conhecimento profundo do negócio.

2.4.2 Decomposição por subdomínio

Este padrão usa Domain Driven Design (DDD) para quebrar o modelo de domínio em subdomínios mais pequenos. Cada um desses subdomínios é ele próprio um domínio e o seu *scope* é chamado *bounded context*. O resultado é que cada micro-serviço centra-se em um único *bounded context* [57].

Este padrão é apropriado para monólitos que já tenham fronteiras bem definidas entre módulos. Se for esse o caso, pode ser possível simplesmente empacotar esses módulos como micro-serviços, sem grandes alterações ao código [57].

Vantagens [57]:

- Arquitetura fracamente acoplada oferece escalabilidade, resiliência, manutenibilidade, estensibilidade, transparência de localização, independência de protocolos e dependência temporal.

- Sistemas mais escaláveis e previsíveis.

Desvantagens [57]:

- Pode originar demasiados micro-serviços, o que dificulta a descoberta e integração de micro-serviços.
- Sub-domínios do negócio podem ser difíceis de identificar pois isso requer um conhecimento mais profundo do negócio.

2.4.3 Decomposição por transações

Para se evitar problemas de latência é possível agrupar os micro-serviços de forma a minimizar o número de transações feitas para cada chamada. Este padrão é apropriado para os casos em que os tempos de resposta são importantes, mas é importante ter em atenção para não se criar um monólito [57].

Vantagens [57]:

- Menores tempos de resposta.
- Consistência dos dados menos importante.
- Maior disponibilidade.

Desvantagens [57]:

- Podem ser empacotados vários módulos juntos, o que pode levar a criação de um monólito.
- Maior custo e complexidade devido a várias funcionalidades implementadas em um único micro-serviço.
- Micro-serviços podem ficar demasiado grandes se houver um grande número de domínios de negócio e de dependências entre eles.
- Possibilidade de haver versões inconsistentes do mesmo domínio de negócio disponíveis em simultâneo.

2.4.4 Decomposição por equipa

É possível fazer a divisão pelas equipas de maneira a que cada equipa fique encarregue de desenvolver, testar e implantar o seu micro-serviço; e, interage com outras equipas principalmente para negociar APIs. Se uma equipa for grande o suficiente pode criar sub-equipas, onde cada uma pode ser proprietária do seu próprio micro-serviço [57].

Vantagens [57]:

- Equipas podem trabalhar independentemente com o mínimo de coordenação.
- O código não é partilhado por mais que uma equipa.
- Equipas podem inovar e iterar rapidamente.
- Equipas diferentes podem usar tecnologias diferentes.

Desvantagens [57]:

- Pode ser difícil alinhar equipas para funcionalidades ou capacidades de negócio.

- Maior dificuldade em entregar incrementos da aplicação de forma coordenada, especialmente se existirem dependências circulares entre equipas.

2.5 Trabalhos similares

Existem várias outras dissertações que abordam o tema dos micro-serviços. Por exemplo:

- João Neves em [58] faz uma análise da literatura e questiona profissionais para compilar os desafios técnicos que rodeiam a arquitetura de micro-serviços. Difere deste trabalho pois pretendo não apenas rever a literatura, mas também aplicar a arquitetura de micro-serviços a um projeto caso de estudo.
- Meng Wang em [59] analisa vários métodos de integração de serviços e aplica em um caso de estudo. Tem por objetivo estudar essas abordagens na integração de serviços. Isso difere deste trabalho pois pretendo estudar também aspetos como o aumento de complexidade de um projeto devido à introdução da arquitetura de micro-serviços.
- Tiago Santos em [60] tem um objetivo semelhante ao deste trabalho. Ele faz uma análise da arquitetura de micro-serviços, aplica a migração de um monólito para micro-serviços, e, avalia os desafios dessa solução. Será interessante comparar os resultados deste trabalho com os do trabalho do Tiago Santos.

Capítulo 3

Análise de valor

É importante não apenas importar uma solução, mas também gerar o máximo valor possível para o nosso cliente, com o custo mais baixo possível. Neste capítulo é feita a análise do valor gerado, e de algumas decisões importantes que foram tomadas.

3.1 Oportunidade

A ideia de criar um *marketplace* de serviços não é nova, no entanto, não existe ainda uma plataforma que o faça de forma dedicada ao mercado africano. Neste projeto a oportunidade que está a ser explorada é a de ser a primeira plataforma desse género no mercado africano, com o objetivo de se tornar na plataforma predominante na região.

3.2 Valor para o cliente

Valor para o cliente pode ser um conceito algo abstrato que pode ser interpretado de várias formas. Tony Woodall em [61] oferece a seguinte definição:

Definição 3.2.1 (Valor para o cliente) *É qualquer precessão pessoal de vantagem que advém da associação do cliente com uma organização, e pode ocorrer com uma redução no sacrifício (comparativamente com alguma alternativa), ou seja, é a resultante de uma combinação de sacrifício e benefício, nem que isso surja de uma agregação feita ao longo do tempo.*

Ou seja, de acordo com [62] é possível escrever a seguinte equação para representar a ligação entre os custos e benefícios:

$$\text{valor} = \frac{\text{benefcio}}{\text{custo}}. \tag{3.1}$$

No caso deste projeto existem dois segmentos de clientes para os quais é gerado valor:

Fornecedores de serviços São o segmento do qual virá toda receita da plataforma, por isso é importante gerar o máximo de valor para eles.

- **Benefício** Os fornecedores podem publicitar os seus serviços a um grande número de potenciais clientes.
- **Custo** Custo mais baixo em relação a um sistema publicitário mais tradicional (ex: televisão, *billboard*, jornal, etc...)

Público geral Sem a atenção do público é impossível gerar valor para os fornecedores, por isso é importante gerar valor para o público também.

Benefício Capacidade de encontrar serviços perto de si de forma centralizada em apenas uma plataforma.

Custo Zero, pois é de livre utilização para fazer pesquisas e pedidos de contacto.

3.3 Perceção de valor

Valarie Zeithaml define o valor percebido em [63] como:

Definição 3.3.1 (Valor percebido) *A avaliação que um consumidor faz da utilidade de um produto, baseando-se nas perceções daquilo que é dado e recebido.*

Diferentes clientes valorizam o mesmo produto ou serviço de formas diferentes, e há vários fatores que influenciam essa valorização. Jozée Lapierre em [64] define os seguintes fatores: soluções alternativas, qualidade do produto, personalização do produto, responsividade, flexibilidade, confiabilidade, competência técnica, imagem (reputação/credibilidade), confiança no fornecedor/vendedor, solidariedade, preço, tempo/esforço/energia, conflito (ou falta dele).

3.4 Cadeia de valor

De acordo com [62] pode-se definir como

Definição 3.4.1 (Cadeia de valor) *Um conjunto de operações que uma organização realiza para gerar valor para os seus clientes.*

Além disso, são também definidos os seguintes componentes de uma cadeia de valor visíveis na Figura 3.1 [65].

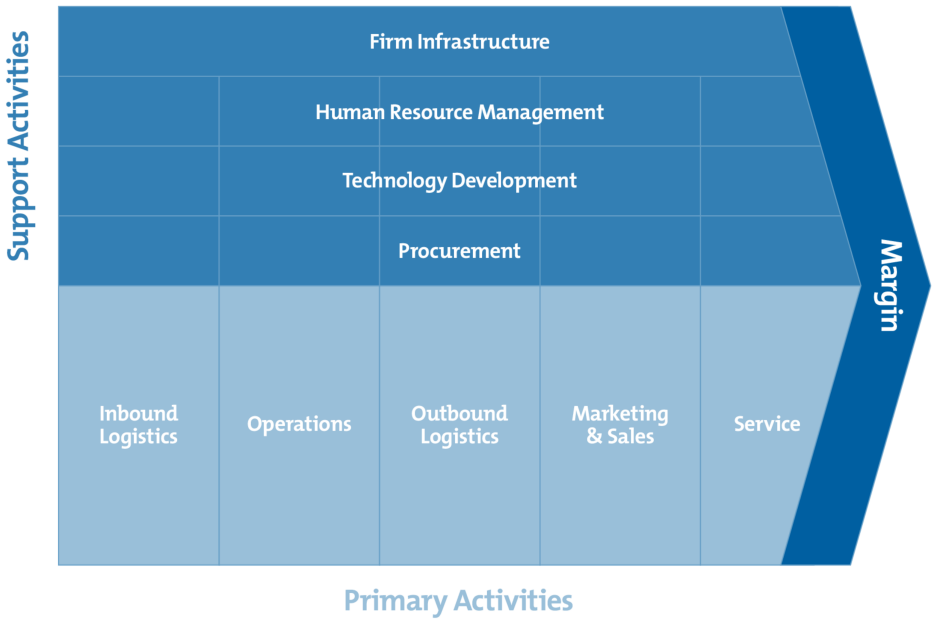


Figura 3.1: Representação esquemática da cadeia de valor. Retirado de [65].

As atividades primárias estão diretamente relacionadas com a criação física de um serviço, da sua venda, manutenção e suporte, e são [65]:

Logística de entrada Processos relacionados com as entradas (matérias-primas).

Operações Atividades de transformação das entradas e saídas que serão vendidas ao cliente.

Logística de saída Atividades relacionadas com a entrega dos produtos/serviços ao cliente.

Marketing e vendas Atividades que persuadem o cliente a comprar.

Serviço Atividades relacionadas com manter o valor do serviço após a venda.

As atividades de suporte suportam as primárias, cada uma delas pode ter algum papel em qualquer uma das primárias, e são [65]:

Compras Aquisição dos recursos que a organização precisa para operar.

Gestão de recursos humanos Atividades relacionadas com recrutar, treinar, motivar, compensar, e reter os seus colaboradores.

Desenvolvimento tecnológico Atividades relacionadas com gerir e processar informação, minimizar os custos tecnológicos, e acompanhar os avanços da tecnologia.

Infraestrutura Os sistemas que permitem a organização funcionar, como contabilidade, administração e área legal.

No final da cadeia de valor aparece a margem, que pode ser definida como [65]:

$$\text{margem} = \text{valor criado} - \text{custo de criar o valor}. \tag{3.2}$$

No caso deste projeto, o trabalho executado faz parte da atividade de suporte de desenvolvimento tecnológico. Sendo que o valor gerado para os fornecedores é o ganho de visibilidade, e para o público geral é o acesso aos fornecedores, não há bens físicos a serem trocados nem transformados. Para fazer uma analogia com a cadeia de valor, considero então que as operações que os fornecedores podem usar para inserir informação fazem parte da logística de entrada. As operações realizadas sobre essa informação faz parte da atividade das operações. As operações de pesquisa que podem ser realizadas para obter a informação dos fornecedores fazem parte da logística de saída.

3.5 Proposta de valor

Jalili e Rezaie definem a proposta de valor em [66] como

Definição 3.5.1 (Proposta de valor) *A estratégia que permite angariar novos clientes.*

A proposta de valor é uma frase curta onde se pretende identificar os benefícios oferecidos aos potenciais clientes. Ela deve identificar o produto, o público alvo, que problemas são resolvidos, e o que o cliente ganha de valor [67].

Para o caso deste projeto coloco a seguinte proposta de valor:

É a primeira plataforma dedicada ao mercado africano para pequenos e médios negócios publicitarem os seus serviços a milhares de utilizadores por uma fração do preço que a publicidade tradicional; e também deixar donos de casa e de negócios (ou encarregados em

negócios) encontrar mais facilmente os serviços que vão de encontro às suas necessidades domésticas e dos seus negócios, respetivamente.

Até agora neste documento referi-me sempre ao segmento de utilizadores que querem encontrar serviços como público geral pois qualquer pessoa pode usar a plataforma para encontrar fornecedores, no entanto, para a proposta de valor decidi restringir esse segmento a um público alvo mais restrito.

Capítulo 4

Avaliar soluções e abordagens existentes

Neste capítulo é feita uma análise dos tipos de tecnologias para comércio eletrónico para avaliar qual seria a melhor escolha entre eles ou uma implementação de raiz. Esta foi uma das primeiras e mais importantes decisões tomadas neste projeto e merece ser analisada em mais detalhe. Isto tem especial impacto neste trabalho pois se tivesse escolhido usar uma *framework* desde o início, uma decomposição para micro-serviços teria sido provavelmente impossível, e a análise neste trabalho não poderia ter sido feita da mesma forma.

4.1 Analytic Hierarchy Process

O *Analytic Hierarchy Process* (AHP) é um processo que permite tomar uma decisão de uma forma organizada. Para isso é necessário gerar as prioridades necessárias para decompor a decisão usando os seguintes passos [68]:

1. Definir o problema e determinar o tipo de conhecimento procurado.
2. Estruturar a decisão hierárquica, desde o objetivo da decisão, pelos critérios, até às alternativas.
3. Construir um conjunto de matrizes de comparação.
4. Usar as prioridades obtidas nas comparação para pesar as diferentes prioridades para cada elemento, para obter as prioridades das alternativas.

4.1.1 Fase 1 - Construção da árvore hierárquica de decisão

Tem-se então por objetivo implementar o *marketplace* para o cliente. Na primeira fase de AHP é necessário identificar alternativas para concluir esse objetivo, e também os critérios que permitem determinar qual delas é a mais adequada.

É necessário minimizar os custos da tecnologia a adotar, pois o orçamento é limitado, por isso o custo é um critério importante. Há também a necessidade de fazer uma implementação rápida para aproveitar a oportunidade identificada na Secção 3.1, e também para minimizar o custo da implementação, por isso a rapidez de implementação é também um critério importante. É necessário implementar os requisitos funcionais de acordo com a especificação do cliente, que tem certas especificidades sobre como devem funcionar as subscrições e o *backoffice*. Por isso a modificabilidade da tecnologia usada também é um critério importante. Como já foi mencionado a escalabilidade é também um dos requisitos no cliente, e será

também usado aqui como critério. Desta forma, foram identificados quatro critérios para AHP.

As alternativas são baseadas nos tipos diferentes de tecnologias para comércio eletrónico identificados na Secção 2.1:

Frameworks Consideram-se neste grupo as *frameworks* específicas de comércio eletrónico, como os exemplos que foram referidos na Secção 2.1. Não se considera Magento, que apesar de ter uma opção grátis, usa tecnologias como PHP [69], para as quais não há competências na equipa. Algo como *Spring boot* é de facto uma *framework*, mas não foi uma criada especificamente para comércio eletrónico, por isso não é considerada nesta alternativa.

SaaS Na Secção 2.1 são referidos os SaaS e os SaaS em micro-serviços separadamente. São suficientemente parecidos que aqui são considerados dentro da mesma alternativa, apenas chamada SaaS.

Plataformas Estas são as restantes plataformas identificadas na Secção 2.1, que não se encaixam em nenhuma das duas alternativas acima.

Nenhuma ferramenta Considera-se aqui a alternativa de não usar nenhuma das de comércio eletrónico referidas acima, mas sim alguma outra ferramenta de propósito genérico e sem custos de licenças, como por exemplo *Spring Boot*.

Com isso, pode-se construir seguinte a divisão hierárquica:

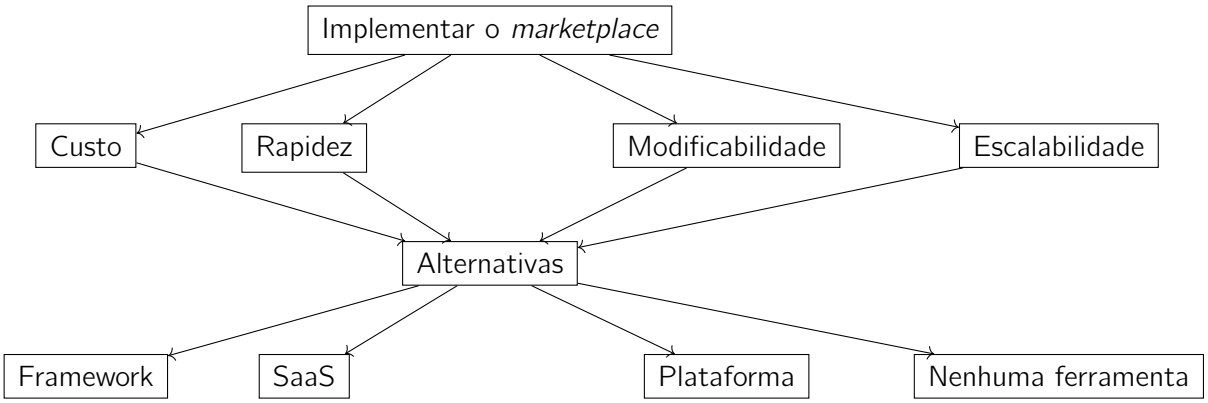


Figura 4.1: Divisão hierárquica.

4.1.2 Fase 2 - Comparação das alternativas e critérios

O custo e a modificabilidade são os mais importantes, pois se não for possível responder aos requisitos funcionais do cliente então não serve para nada a implementação, similarmente, se a tecnologia for demasiado cara o cliente não terá orçamento para a pagar. A rapidez de implementação vem a seguir em importância, e por fim a escalabilidade, pois mesmo que seja difícil ou trabalhoso, é normalmente possível escalar qualquer tecnologia.

Então ao completar a matriz é necessário pontuar cada um dos critérios em relação aos outros. Em cada linha, pontua-se 1 se o critério na linha for de igual importância ao da coluna, e valores superiores até 9, se o critério na linha for de maior importância. Usam-se os valores inversos, se o critério na coluna for mais importante que o da linha.

Assim, foi criada a seguinte matriz de comparação de critérios:

	Custo	Rapidez	Modificabilidade	Escalabilidade
Custo	1	5	1	7
Rapidez	1/5	1	1/3	3
Modificabilidade	1	3	1	4
Escalabilidade	1/7	1/3	1/4	1

Tabela 4.1: Matriz de comparação de critérios.

4.1.3 Fase 3 - Prioridade relativa de cada critério

A matriz normalizada é:

	Custo	Rapidez	Modificabilidade	Escalabilidade
Custo	0.43	0.54	0.39	0.47
Rapidez	0.09	0.11	0.13	0.20
Modificabilidade	0.43	0.32	0.39	0.27
Escalabilidade	0.06	0.04	0.10	0.07

Tabela 4.2: Matriz normalizada de comparação de critérios.

As médias de cada linha são o peso relativo de cada critério:

Custo	0.45
Rapidez	0.13
Modificabilidade	0.35
Escalabilidade	0.07

Tabela 4.3: Peso relativo de cada critério.

4.1.4 Fase 4 - Avaliar a consistência das prioridades relativas

É necessário calcular a Razão de Consistência para determinar o quanto os juízos na Tabela 4.1 foram consistentes. Para isso é necessário calcular uma matriz onde cada valor é dado pelo valor da Tabela 4.1 multiplicado com o respetivo valor próprio. Essa matriz encontra-se na Tabela 4.4, onde estão também representadas as somas de cada linha da matriz.

	Custo	Rapidez	Modificabilidade	Escalabilidade	Soma
Custo	0.45	0.65	0.35	0.46	1.91
Rapidez	0.09	0.13	0.12	0.20	0.53
Modificabilidade	0.45	0.39	0.35	0.26	1.46
Escalabilidade	0.06	0.04	0.09	0.07	0.26

Tabela 4.4: Matriz de comparação de critérios multiplicada pelos valores próprios, e a soma de cada linha.

λ_{max} é definido como a média das somas da Tabela 4.4 cada uma delas dividida pelo respetivo valor próprio:

$$\lambda_{max} = \text{media}\left(\frac{1.91}{0.45}, \frac{0.53}{0.13}, \frac{1.46}{0.35}, \frac{0.26}{0.07}\right) \cong 4.12. \tag{4.1}$$

O índice de consistência é definido como:

$$CI = \frac{\lambda_{max} - n}{n-1} \cong 0.0388, \tag{4.2}$$

onde n é o número de critérios. Finalmente, é possível calcular a Razão de Consistência:

$$RC = \frac{CI}{ICA} \cong 0.0431, \tag{4.3}$$

onde ICA é o Índice de Consistência Aleatória, e para quatro critérios tem o valor de 0.9, de acordo com [70]. Como a Razão de Consistência é menor que 0.1 então pode-se considerar que há uma inconsistência aceitável [70].

4.1.5 Fase 5 - Construção da matriz de comparação paritária para cada critério

É necessário comparar as diferentes alternativas, para cada um dos critérios. Então é feita uma comparação entre todas as alternativas no contexto de cada critério, e são criadas matrizes de comparação com as mesmas regras de pontuação como foi feito para a comparação de critérios na Fase 2 do AHP.

Critério custo

Geralmente as *frameworks* têm os maiores custos com licenças, de seguida os SaaS, e por fim as plataformas, de acordo com a Secção 2.1.1. Para a implementação de raiz, é assumido custo zero pelo uso de tecnologias sem custos relacionados com licenças.

Critério custo	Framework	SaaS	Plataforma	Nenhuma
Framework	1	$1/3$	$1/5$	$1/9$
SaaS	3	1	$1/3$	$1/9$
Plataform	5	3	1	$1/5$
Nenhuma	9	9	5	1

Tabela 4.5: Matriz de comparação das alternativas tendo em consideração o critério do custo.

Para esta matriz obtém-se uma Razão de Consistência de 0.069.

Critério rapidez

Considera-se que as plataformas têm o tempo para o mercado mais rápido, de seguida os SaaS, as *frameworks* e por fim a implementação de raiz, pois não é possível reutilizar nenhuma funcionalidade já implementada, e tudo tem que ser feito de raiz.

Critério rapidez	Framework	SaaS	Plataforma	Nenhuma
Framework	1	1/3	1/4	3
SaaS	3	1	1/2	3
Plataform	4	2	1	4
Nenhuma	1/3	1/3	1/4	1

Tabela 4.6: Matriz de comparação das alternativas tendo em consideração o critério da rapidez.

Para esta matriz obtém-se uma Razão de Consistência de 0.067.

Critério modificabilidade

É considerado que as *frameworks* e a implementação de raiz oferecem a maior modificabilidade, seguido de perto pelo SaaS. A modificabilidade das plataformas é mais variável, depende da plataforma, e de como a solução está construída. Considera-se então que são menos modificáveis que as outras.

Critério modificabilidade	Framework	SaaS	Plataforma	Nenhuma
Framework	1	3	7	1
SaaS	1/3	1	5	1/3
Plataform	1/7	1/5	1	1/7
Nenhuma	1	3	7	1

Tabela 4.7: Matriz de comparação das alternativas tendo em consideração o critério da modificabilidade.

Para esta matriz obtém-se uma Razão de Consistência de 0.028.

Critério escalabilidade

Nesta secção são comparados o esforço envolvido em escalar as diferentes alternativas. As *frameworks* originam geralmente um monólito que pode ter os problemas normais relacionados com a escalabilidade de monólitos. A implementação de raiz pode ser um monólito, ou pode ser alguma outra coisa. Pode dar mais ou menos trabalho para escalar, mas como é uma solução especializada para este projeto, mesmo que seja um monólito, será um monólito mais pequeno. Então considera-se mais fácil de escalar que uma *framework*. Por fim, os SaaS e as plataformas consideram-se igualmente fáceis de escalar, pois esse trabalho não está do nosso lado.

Critério escalabilidade	Framework	SaaS	Plataforma	Nenhuma
Framework	1	1/5	1/5	1/3
SaaS	5	1	1	3
Plataform	5	1	1	3
Nenhuma	3	1/3	1/3	1

Tabela 4.8: Matriz de comparação das alternativas tendo em consideração o critério da escalabilidade.

Para esta matriz obtém-se uma Razão de Consistência de 0.016.

4.1.6 Fase 6 - Obter a prioridade composta para as alternativas

Fazendo cálculos similares aos da Fase 3, para cada tabela da Fase 5, obtiveram-se os pesos relativos das alternativas, para todos os critérios. Combinando todos esses resultados na mesma matriz, obtém-se o seguinte resultado:

	Custo	Rapidez	Modificabilidade	Escalabilidade
Framework	0.0473	0.1522	0.3936	0.0687
SaaS	0.0927	0.2889	0.1645	0.3889
Plataforma	0.1991	0.4723	0.0483	0.3889
Nenhuma	0.6609	0.0867	0.3936	0.1535

Tabela 4.9: Matriz da prioridade composta para as alternativas.

Ao multiplicar esta matriz com o vetor dos pesos de cada critério que se encontram na Tabela 4.3 obtém-se a prioridade para cada uma das alternativas:

Framework	0.1838
SaaS	0.1627
Plataforma	0.1942
Nenhuma	0.4593

Tabela 4.10: Prioridade de cada alternativa.

4.1.7 Fase 7 - Escolha da alternativa

Na tabela 4.10 verifica-se que a alternativa de fazer a implementação de raiz é a alternativa que tem o maior valor, por isso é a mais apropriada, tendo em conta os critérios estabelecidos. O monólito que foi implementado para o cliente da Kodly foi adequadamente implementado de raiz.

Capítulo 5

Análise

Neste capítulo são analisados os requisitos do cliente e é exposto o modelo de domínio que foi criado durante a implementação do monólito.

5.1 Requisitos funcionais

O objetivo do projeto é criar um marketplace de serviços, ou seja, uma plataforma onde fornecedores de serviços se podem inscrever para publicar os seus serviços. Utilizadores interessados em contratar serviços devem poder pesquisar e filtrar serviços, fazer pedidos de contacto e deixar revisões sobre o serviço do fornecedor.

Utilizadores podem se inscrever como compradores ou como fornecedores. Compradores podem mais tarde se tornar em fornecedores. Utilizadores podem também gerir o seu perfil e lista de desejos, mas apenas fornecedores podem publicar e gerir os seus serviços.

Um pedido de contacto pode apenas ter sucesso se o fornecedor contactado tiver uma subscrição válida, ou seja, uma subscrição anual ativa ou então pelo menos um token na sua carteira virtual de tokens. Ou seja, um fornecedor pode subscrever à plataforma por um ano, escolhendo um de vários planos de pagamento disponíveis (com diferentes frequências de pagamentos). Se preferir não ter esse tipo de pagamentos fixos, pode comprar número de tokens que ficam guardados na sua carteira virtual. A cada vez que for feito um pedido de contacto, um token é descontado da carteira. Se um fornecedor não tiver uma subscrição ativa nem tokens na carteira, ele é incontactável através da plataforma. As subscrições e tokens podem ser pagas através de Paypal e de Paystack.

Compradores podem deixar revisões sobre serviços e fornecedores, e, fornecedores podem deixar revisões sobre compradores. Ambos podem deixa revisões sobre a própria plataforma (as quais ficam apenas visíveis para utilizadores de negócio). As revisões são compostas por uma classificação, um campo de escrita livre, e também algumas questões que têm que ser respondidas, tais como: “Conseguiu contactar o fornecedor?”, “Conseguiu contratar o serviço?”, e “Ficou satisfeito com o serviço?”.

Utilizadores de negócio podem gerir subscrições, questões para os utilizadores responderem nas revisões, suspender, apagar e validar serviços e fornecedores, gerir categorias e gerir conteúdos da plataforma.

Estes requisitos funcionais encontram-se implementados no monólito e devem ser mantidos para o sistema de micro-serviços.

5.2 Requisitos não funcionais

O único requisito não-funcional de relevância é o da escalabilidade, dada a expansão que o negócio pretende fazer. A forma como isso foi tratado no monólito foi fazendo a implantação da aplicação na nuvem, usando uma *elastic pool* que possa ir aumentando o número de recursos ou máquinas consoante o tráfego. Outra medida tomada foi a implementação do monólito usando uma estrutura interna que facilitasse uma possível subsequente divisão em micro-serviços. Essa estrutura é detalhada no próximo capítulo.

A motivação para este trabalho é precisamente tentar levar esta capacidade de escalar o projeto um passo mais longe, e por isso este requisito é também muito importante para a aplicação por micro-serviços.

5.3 Modelo de domínio

Na Figura 5.1 encontra-se o modelo de domínio que foi concebido no monólito.

5.3.1 Utilizadores

A entidade User representa os utilizadores, que podem ser de três tipos diferentes: comprador (buyer), fornecedor (provider) ou de negócio (business).

Por linhas gerais: os fornecedores são os únicos que têm a permissão de criar serviços; os utilizadores de negócio de suspender e apagar serviços e fornecedores, gerir categorias, subscrições e perguntas de revisões; e, os compradores apenas podem consultar os serviços e fazer pedidos de contacto aos fornecedores.

5.3.2 Subscrições

Existem dois tipos de subscrições que os fornecedores podem escolher, anual e por token. Sem uma subscrição não é possível receberem pedidos de contacto por parte dos compradores.

A entidade AnnualSubscription serve para modelar quais as diferentes opções de subscrição anual, com diferentes preços e frequências de pagamento. A entidade TokenSubscription serve para modelar os pacotes de *tokens* que estão disponíveis para venda e os seus preços.

5.3.3 Order

No modelo de negócio e na implementação chamou-se ServiceOrder a um pedido de contacto. Isto é porque inicialmente isto seria utilizado para encomendar o serviço e incluiria um pagamento do comprador ao fornecedor feito através da plataforma, então o nome Order faria sentido. O âmbito desta funcionalidade foi reduzido a um mero pedido de contacto, mas o nome acabou por se manter.

5.3.4 Review

Uma ServiceReview é uma revisão deixada sobre um serviço e o seu fornecedor, uma BuyerReview é uma revisão sobre um comprador, e uma PlatformReview é uma revisão sobre a plataforma.

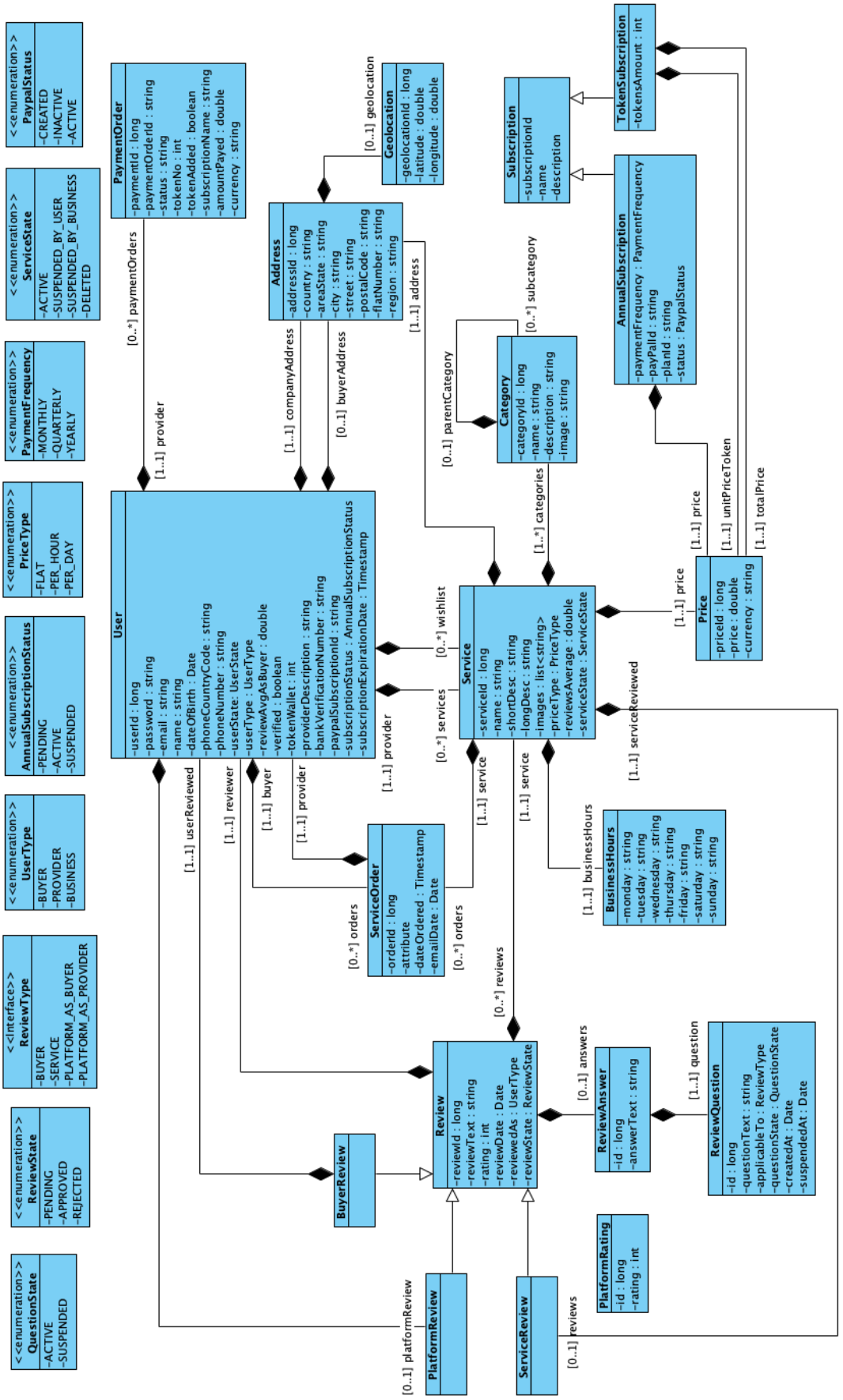


Figura 5.1: Modelo de domínio.

Capítulo 6

Design da solução

Neste capítulo analisa-se primeiro o design do monólito para ser conhecido aquilo que se pretende decompor. De seguida, dois padrões de decomposição são escolhidos e são aplicados ao monólito de forma a prever o tipo de arquitetura que originariam. Um deles é escolhido para a implementação.

6.1 Monólito

Nesta secção é exposta a arquitetura do monólito, usando o modelo C4 de documentação de Arquitetura de Software, nas granularidades de sistema e de aplicação.

6.1.1 Granularidade de sistema

Na granularidade de sistema o objetivo é ter uma vista do sistema como um todo, e analisar como as suas componentes se relacionam e interagem. Não se deve entrar em detalhes internos sobre nenhuma delas.

Ponto de vista lógico

Existem duas aplicações de interface do utilizador (UI), uma que é o próprio *marketplace*, que é para ser usada pelos utilizadores de tipo comprador e fornecedor. A outra é o *backoffice* para ser usada pelos utilizadores de negócio. Ambas consomem o mesmo monólito, que por sua vez consome a base de dados e também algumas APIs externas, tal como está representado na Figura 6.1.

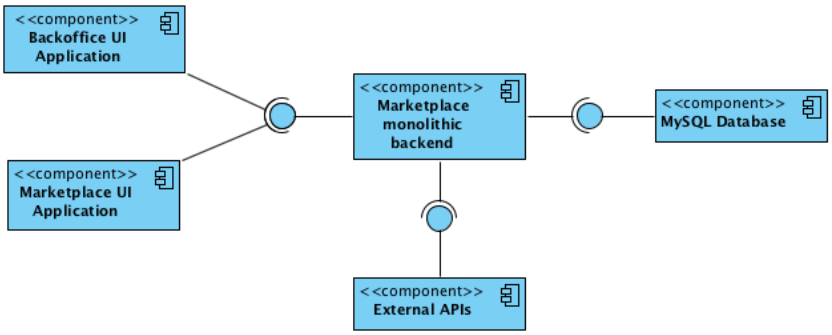


Figura 6.1: Ponto de vista lógico na granularidade de sistema do monólito.

Ponto de vista de processo

Para o ponto de vista de processo apresenta-se na Figura 6.2 o exemplo de um comprador criar uma nova revisão sobre um serviço. Primeiro é necessário carregar quais são as perguntas que devem ser apresentadas. Depois, quando é feita a submissão da nova revisão, é necessário obter da base de dados os dados do utilizador bem como do serviço, para guardar a nova revisão, e fazer o cálculo da nova pontuação média para atualizar o serviço.

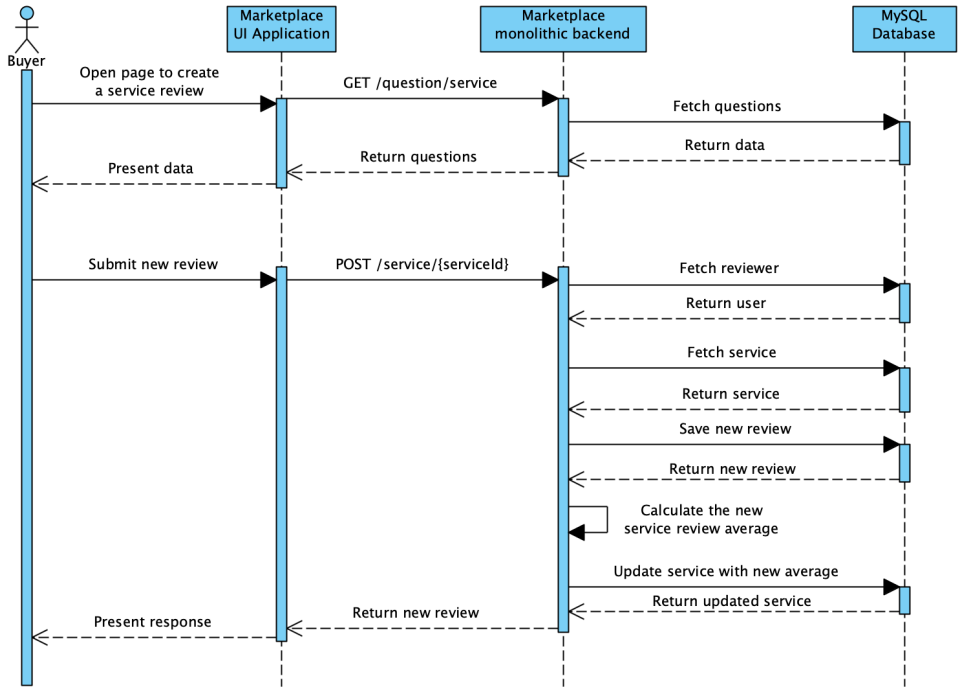


Figura 6.2: Ponto de vista de processo na granularidade de sistema do monólito.

Ponto de vista de implementação

Para o ponto de vista de implementação, existem então as duas aplicações de UI, que usam o monólito, o qual usa a base de dados, como se encontra representado na Figura 6.3.

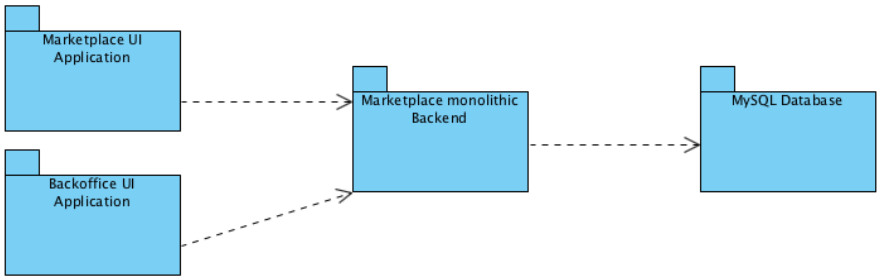


Figura 6.3: Ponto de vista de implementação na granularidade de sistema do monólito.

Ponto de vista de implantação

A implantação na nuvem foi feita usando a plataforma da DigitalOcean. Todas as aplicações estão *containerizadas* exceto a base de dados, que é uma solução gerida da DigitalOcean. Há a possibilidade de aumentar o número de instâncias do monólito, se a quantidade de tráfego assim o exigir, e a DigitalOcean gere também o *load balancer* usado para distribuir a carga entre as diferentes instâncias do monólito. Assim, é possível distinguir três *tiers*. Um que contém todas as aplicações de UI, outro com as instâncias do monólito e o respetivo *load balancer*, e um que contém a base de dados. A função de cada *tier* é distinta: interface com o utilizador, lógica de negócio, e, armazenamento de dados, respetivamente. A Figura 6.4 apresenta o diagrama de implantação.

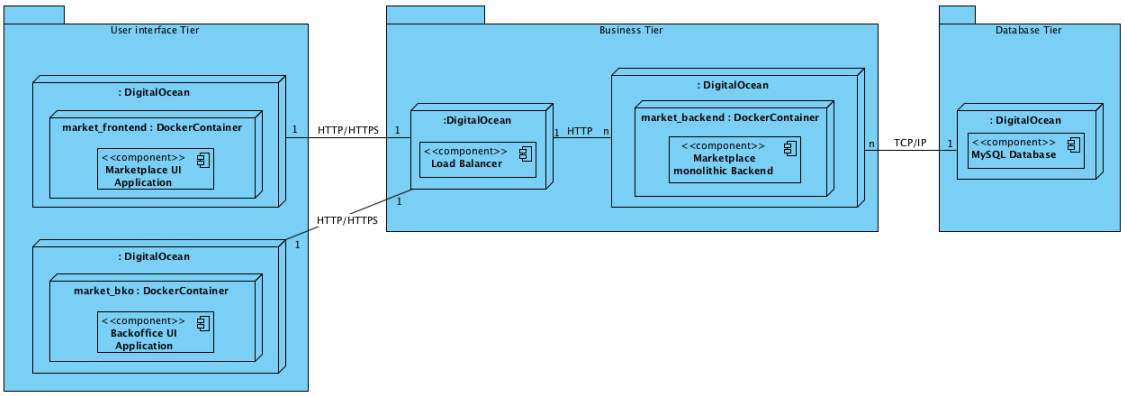


Figura 6.4: Ponto de vista de implantação na granularidade de sistema do monólito.

6.1.2 Granularidade de aplicação

Na granularidade de aplicação o objetivo é analisar os detalhes internos do componente de interesse, neste caso é analisado o monólito. Ou seja, serão analisadas as relações e interações dos componentes interno do monólito.

Ponto de vista lógico

A estrutura lógica interna do monólito é um pouco complexa, por isso é aqui introduzida de uma granularidade mais fina até uma que abrange toda a aplicação.

O monólito é constituído por vários pacotes em que cada um está focado em uma (ou apenas algumas) das entidades de domínio. Todos eles foram criados para ter uma estrutura interna semelhante, organizada por camadas, como no diagrama da Figura 6.5. O código foi organizado dessa forma precisamente para facilitar uma possível eventual divisão por micro-serviços.

Um destes pacotes não é verdadeiramente uma componente no sentido em que não pode ser compilados nem distribuídos de forma independente uns dos outros, no entanto, para esta análise é útil tratá-los como tal, por isso será feito esse abuso de linguagem.

Análise de um único componente da aplicação Na Figura 6.5 encontra-se representada a estrutura interna do componente User a título exemplificativo, a qual segue um estilo arquitetural por camadas.

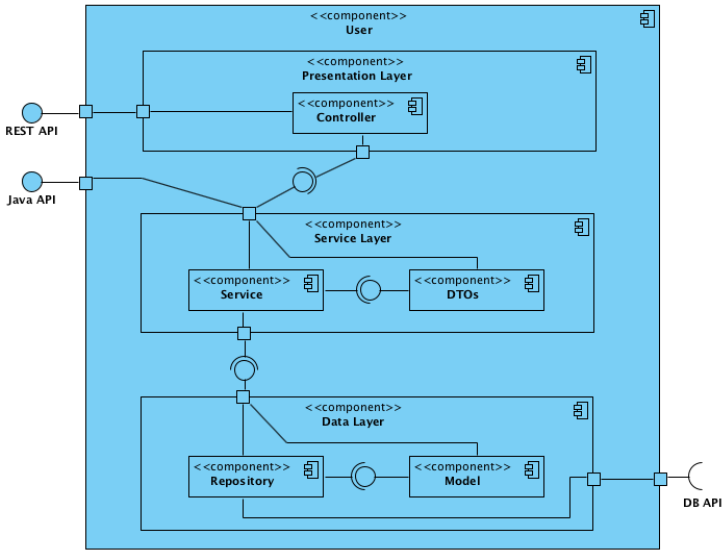


Figura 6.5: Ponto de vista de lógico do componente User, na granularidade de aplicação do monólito.

A primeira camada é a camada de apresentação, que contém o controlador e expõe uma API REST que contribui para a API exposta pelo monólito (e consumida pelas aplicações de UI).

A camada de apresentação consome a camada de negócio, onde se encontra o serviço que contém a lógica de negócio. O serviço mostra apenas ao controlador os DTOs (*Data Transfer Objects*) e isola as entidades do modelo. Além disso, o serviço contém a interface que pode ser consumida pelos outros componentes do monólito (para obter dados é sempre necessário passar pelo serviço do componente que tem acesso a esses dados).

Por último, a camada de dados é apenas consumida pela de negócio, e é a única que tem acesso à API da base de dados. Esse acesso é feito através do repositório, apenas. Aqui não existe uma divisão entre a entidade de dados e a entidade de domínio, o que em alguns casos sobrecarrega a entidade com algumas propriedades úteis para o modelo de dados mas não para o de domínio.

Análise com alguns componentes da aplicação Na Figura 6.6 encontram-se representados os componentes mencionados acima como contendo uma ou mais entidades de domínio. Quase todas contribuem para a REST API do monólito, e todas acedem à base de dados.

Alguns componentes apenas gerem uma entidade de domínio, tais como: User gere os utilizadores; Order regista os pedidos de contacto; Address gere as moradas; e, Payment gere os pagamentos das subscrições integrando com os serviços externos de pagamento como Paypal e Paystack.

Outros componentes gerem várias entidades de domínio, como Subscription que gere todos os tipos de subscrição, e Review todos os tipos de Revisão. Service é o componente mais complexo. Gere os serviços e as categorias, e contém as funcionalidades de pesquisa de serviços.

Sobre as suas relações, o User consome apenas Address, e todos os restantes componentes podem consumir User (nem todos precisam). O Service consome o User e a Address, e todas as restantes podem consumir Service. Existe uma referência cíclica entre Payment e Subscription, pois o Payment tem que saber qual subscrição está a ser paga, e o Subscription tem que integrar as subscrições com sistemas externos, e essa integração está apenas implementada no Payment.

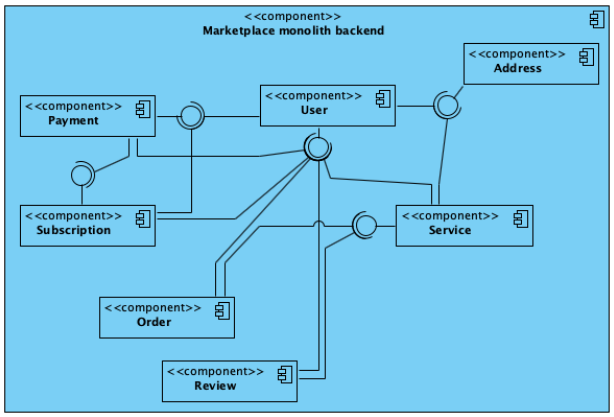


Figura 6.6: Ponto de vista de lógico com apenas alguns componentes, na granularidade de aplicação do monólito.

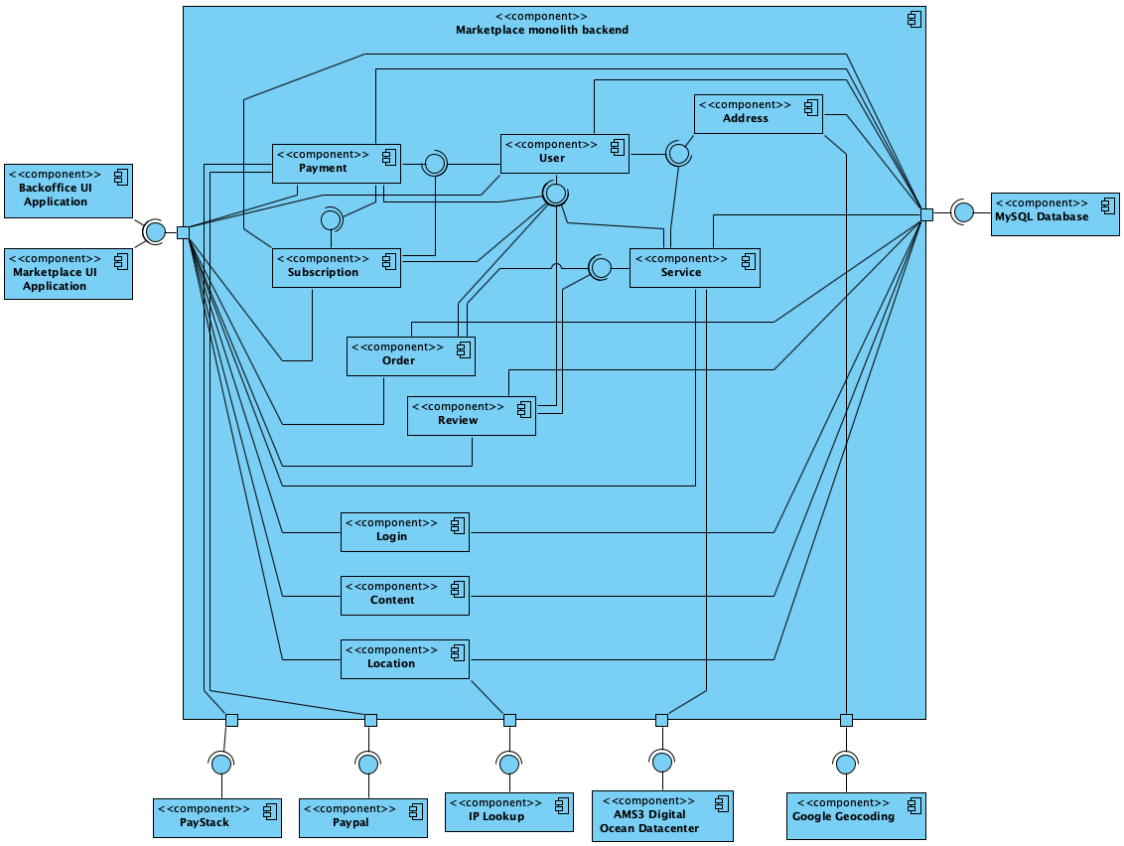


Figura 6.7: Ponto de vista de lógico com mais componentes, na granularidade de aplicação do monólito.

Análise introduzindo mais alguns componentes e relações Existem alguns outros componentes que foram introduzidos no diagrama da Figura 6.7. Foram também introduzidas as relações dos componentes com as aplicações de UI, a base de dados, e com serviços externos.

O componente Login gere o registo e autenticação dos utilizadores. Estas funcionalidades foram segregadas do componente User para facilitar o uso de OAuth2.0 no futuro. O componente Login seria a base para um *Authorization Server*, que precisa de existir em separado do servidor que contém os recursos protegidos [71].

O componente Content gere alguns conteúdos da plataforma que têm de ser guardados em base de dados, e o Location gere listas de localizações guardadas na base de dados, para os utilizadores poderem usar como filtros de pesquisa de serviço.

O Location usa serviços de localização de IP para a funcionalidade de procurar serviços que estão perto do utilizador. O Service integra com um *datacenter* para armazenar as imagens que os fornecedores carregam para os seus serviços. Por fim, o Address integra com a API de Geocoding da Google para descobrir as coordenadas geográficas das moradas introduzidas pelos utilizadores.

Os restantes componentes Os componentes que ainda se encontravam em falta são o Backoffice, o ActionService e o Email. Na Figura 6.8 encontra-se o diagrama onde se representa o relacionamento destes componentes com os restantes, e, por simplicidade, foram removidas as relações entre os componentes que já tinham sido mostradas anteriormente.

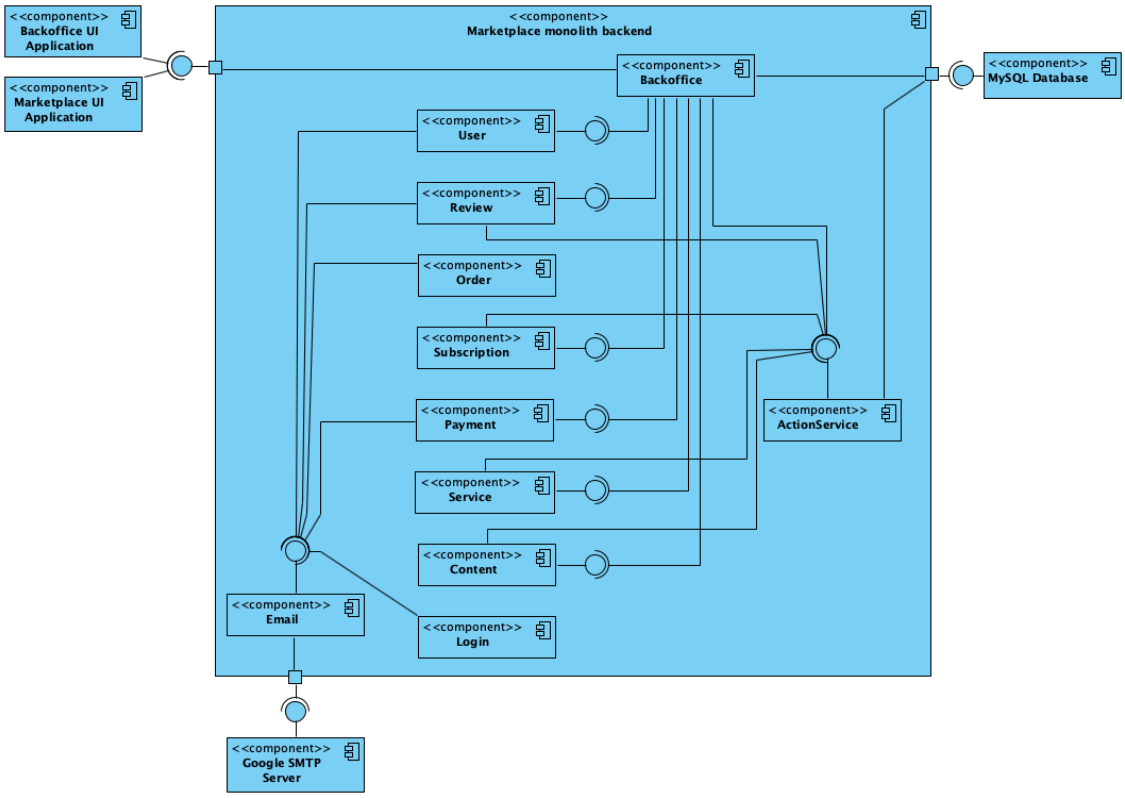


Figura 6.8: Ponto de vista de lógico com tudo incluído, na granularidade de aplicação do monólito.

O componente Backoffice contém as funcionalidades para os utilizadores de negócio gerirem a aplicação, e consome vários dos outros componentes. Content serve para gerir alguns conteúdos e configurações da plataforma. Só podem ser alterados por utilizadores a partir do Backoffice, mas estão disponíveis para leitura pela Marketplace UI Application. O componente Location apenas serve para fazer consultas sobre países e estados presentes na base de dados para a funcionalidade *autocomplete* da pesquisa.

O componente Email consome a API de um servidor SMTP para fazer o envio de emails, e, é consumido por vários componentes do monólito.

Por fim, o componente ActionService apenas regista certas ações que os utilizadores tomam na base de dados. Ele é consumido por vários outros para guardar na base de dados o acontecimento de certos eventos, que serve para guardar métricas de serviço.

Tudo junto No final, juntando todos os elementos, obtém-se o diagrama que se encontra na Figura A.1 do anexo A. É um diagrama extenso e já difícil de analisar.

Ponto de vista de processo

Tomando o mesmo exemplo da criação de uma nova revisão, é necessário obter primeiro o utilizador e o serviço. Isso é feito através dos seus respetivos serviços que chamam os repositórios. De seguida o serviço da ServiceReview chama o seu repositório para guardar a nova revisão, e por fim novamente o serviço dos Service para guardar a nova média. Este processo encontra-se na Figura 6.9.

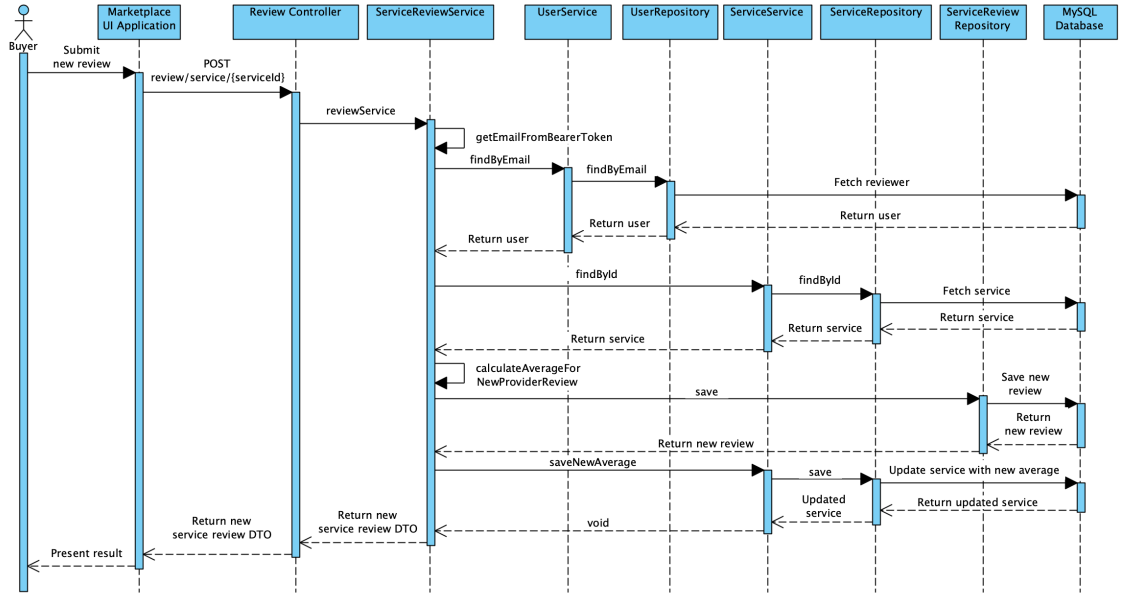


Figura 6.9: Ponto de vista de processo na granularidade de aplicação do monólito.

Ponto de vista de implementação

A parte mais fundamental para a lógica de negócio é:

- O pacote do User apenas consome o da Address.
- O Service consome o User e a Address.

- O Order e o Review consomem ambos o Service e o User.
- O Subscription consome o Service e o Payment.

O Email é consumido por vários outros sem consumir nenhum, e o Backoffice consome vários sem ser consumido por nenhum.

O diagrama de pacotes encontra-se na Figura 6.10.

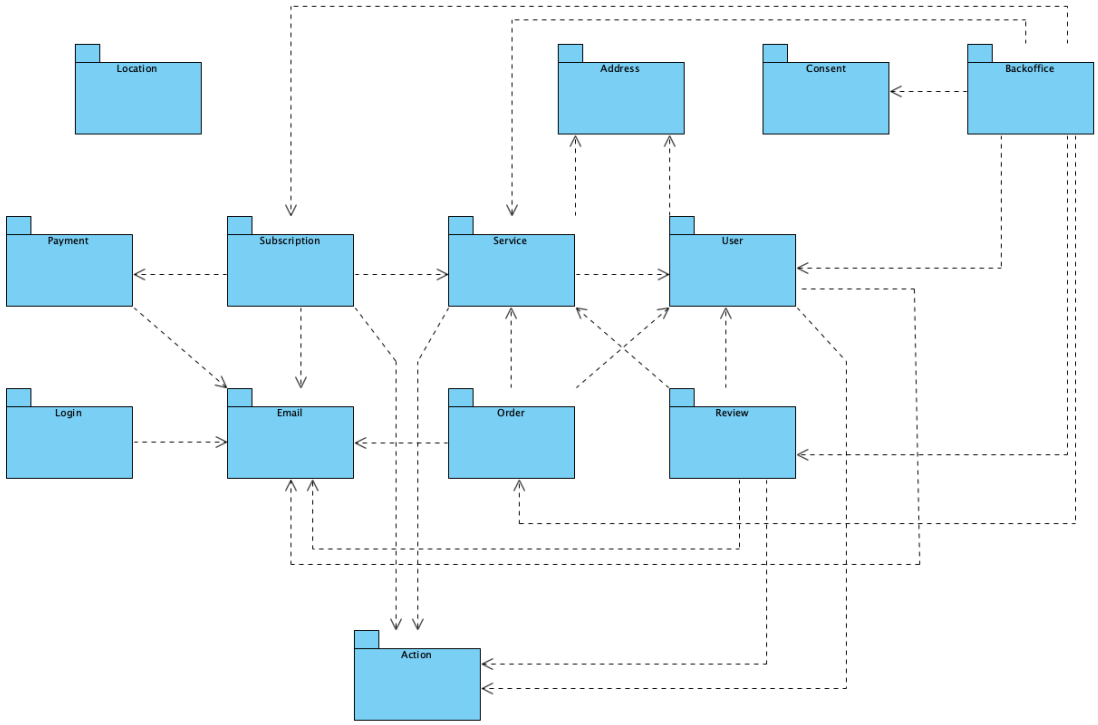


Figura 6.10: Ponto de vista de implementação na granularidade de aplicação do monólito.

Ponto de vista de implantação

O monólito encontra-se containerizado usando o Docker e implantado numa máquina da DigitalOcean, tal como se encontra na Figura 6.11.

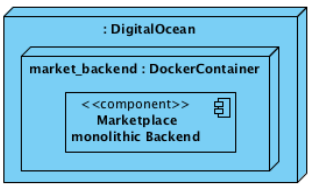


Figura 6.11: Ponto de vista de implantação na granularidade de aplicação do monólito.

6.2 Design da solução em arquitetura de Micro-serviços

Nesta secção é criado o desenho do sistema, agora dividido em micro-serviços. O monólito tem uma estrutura interna já construída de forma a facilitar uma eventual transformação em micro-serviços, por isso o trabalho de desenhar e implementar este sistema em micro-serviços poderá não ser representativo do trabalho envolvido para fazer o mesmo para qualquer monólito. No entanto, continua a poder fazer-se uma análise do comportamento dos dois sistemas que é completamente válida.

Abordagens para dividir um monólito em micro-serviços

Na Secção 2.4 encontra-se a revisão de quatro formas diferentes de decompor monólitos.

A decomposição por capacidades de negócio não é apropriada para este caso pois na Secção 3.4 foi identificado que este projeto se enquadra apenas na atividade de desenvolvimento tecnológico do negócio, não abrangendo assim mais que uma capacidade de negócio.

A decomposição por equipa também não é apropriada pois a equipa de desenvolvimento é pequena e funciona bem como uma equipa ágil. Subdividir uma equipa de quatro pessoas em dois pares ou em equipas de uma pessoa, deixa de fazer sentido chamar-lhe equipa.

As decomposições por subdomínio e por transações são ambas opções interessantes para este caso pois beneficiam a escalabilidade e a disponibilidade, respetivamente. Ambas são exploradas nas secções abaixo.

Decomposição por subdomínio

Para a decomposição por subdomínio é necessário dividir o modelo de domínio apresentado na Figura 5.1 em *bounded contexts* (BC) usando DDD, e cada um dos BCs dará origem a um micro-serviço.

DDD não explica como é possível identificar um BC, apenas o define como uma fronteira lógica dentro da qual um certo sub-domínio faz sentido [72].

Para as entidades do modelo é fundamental que seja mantida uma identidade associada a si. Para evitar impactos na performance é possível omitir as identidades em alguns objetos que não têm entidade conceptual, e apenas descrevem as características de uma entidade. Em DDD esses objetos são chamados *value objects* [49].

Analisando o modelo de domínio foram identificados os BCs identificados no diagramas da Figura 6.12. Apesar de todas as entidades no diagramas terem sido implementadas com identidade algumas delas poderiam alternativamente ser transformadas em *value objects*, tais como: BusinessHours, Price, Address e Geolocation.

O BC das revisões foi fácil de identificar pois há várias entidades relacionadas com as revisões com muitas relações entre si. As suas entidades relacionam-se também com ambos o User e o Service.

Em relação ao BC dos pagamentos é muito simples e contêm apenas uma entidade relacionada com o User, pois apenas precisa de identificar o utilizador que fez o pagamento.

Por fim, o BC dos pedidos de contacto, relaciona-se intimamente com ambos os utilizadores e os serviços, por isso poderia alternativamente ficar junto com uma dessas entidades. A sua separação pode ter no entanto vantagens. Permite extrair algumas propriedades dos

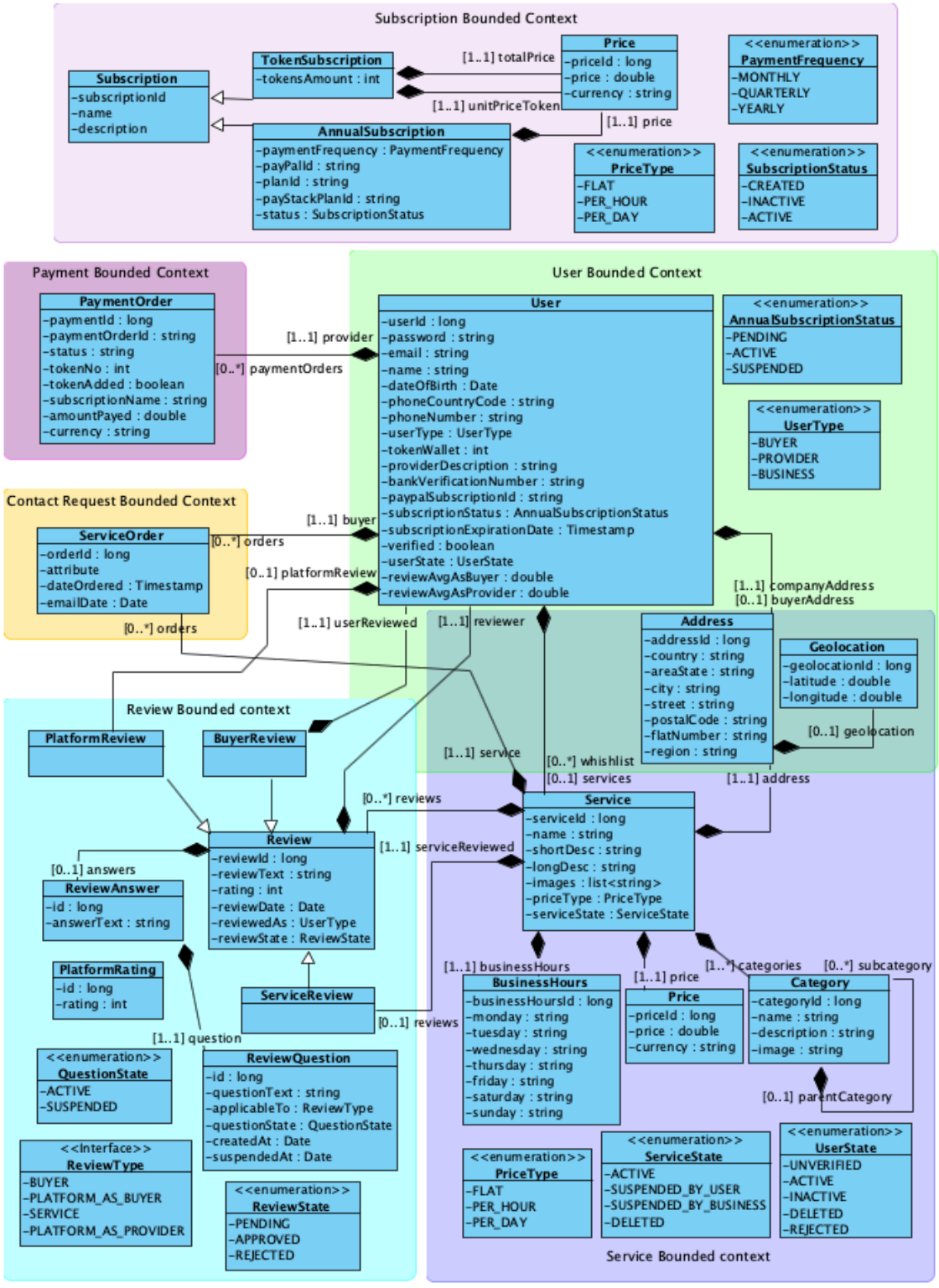


Figura 6.12: Diagrama de domínio DDD com identificação dos BCs.

utilizadores e serviços, ficando com uma versão de ambas mais especializada neste BC. Além disso, favorece o princípio da responsabilidade única: as regras que governam se um pedido de contacto pode ser feito podem mudar sem que mude a lógica que governa a gestão de utilizadores e serviços. Por isso, este BC torna-se num candidato interessante

para a separação.

A separação do User e do Service é complicada pelo facto de que eles partilham a entidade da Address. Separá-los obriga a repetir a morada em ambos os BCs. Isso oferece a flexibilidade de modelar a morada de negócios e de utilizadores de formas diferentes, mas obriga a repetir a lógica para chamar a API da Geolocation da Google. Uma alternativa poderia ser de criar um micro-serviço para implementar apenas essa chamada, ou até mudar a estratégia e ter essa funcionalidade num serviço *serverless*, como uma função AWS lambda. Como esta funcionalidade apenas é chamada quando um utilizador introduz ou modifica uma morada, a latência adicional de ter um serviço que chama outro serviço não cria um risco de falha ou atraso demasiado grande no sistema.

Então, propõem-se dois BCs para o utilizador e para o serviço. O BC do utilizador é bastante pequeno e centra-se em gerir os utilizadores, as suas moradas, e as suas *wishlists* (listas de desejos). O BC dos serviços é um pouco mais complexo gerindo os serviços, as suas moradas e categorias, e também a funcionalidade de pesquisa de serviços. Alternativamente, pelo princípio da responsabilidade única, poderia ter-se argumentos para separar as categorias deste BC. Um dos parâmetros de pesquisa permite filtrar serviços pela categoria, por isso, para evitar a latência originada por uma chamada subsequente, pode ser vantajoso manter esta categoria perto do serviço.

Existem outras componentes no sistema que esta abordagem não ajuda a decidir como separá-las do monólito. O Backoffice não gere as suas próprias entidades de dados, apenas chama outros componentes. Além disso, expõe os endpoints que são consumidos pela aplicação de backoffice, por isso faz sentido separá-lo para o seu próprio micro-serviço.

Componentes como Content e Location gerem uma entidade de dados cada uma que não se relaciona com as outras. Podem ficar em micro-serviços separados ou então juntos, pois a sua responsabilidade é mais ou menos a mesma, de servir conteúdo que está guardado e raramente é alterado.

O componente Login pode ser transformada no componente *identity provider* de um sistema SSO, como se encontra descrito na Secção 2.3.15.

O componente ServiceAction gere uma entidade de dados. O seu trabalho é recolher métricas de serviço e guardá-las na base de dados. Pode ser simplesmente separado para um micro-serviço, ou então completamente substituído por uma ferramenta de monitorização que permita fazer o mesmo.

Resta analisar o componente de Email. Este apenas serve para integrar com um serviço de envio de emails. A sua lógica pode ser repetida em todos os micro-serviços que necessitem, ou então, pode ser o seu próprio micro-serviço.

Assim, a estrutura dos micro-serviços acaba por imitar a estrutura interna do monólito, como se pode ver na Figura 6.13. Para simplificar o diagrama, foram deixadas de parte os componentes ActionService e Email. Sabe-se que ambas serão consumidas por vários dos micro-serviços e nesta fase inicial não precisam de muita mais análise que isso.

Foi introduzido um componente novo chamado Gateway que serve para consultar o Login e redirecionar o pedido para o micro-serviço pretendido com a informação da identidade do utilizador, tal como definido na Secção 2.3.15.

O Backoffice consome apenas a Gateway, e a partir daí é que tem acesso aos restantes micro-serviços. Já que o Backoffice necessita também de autorização, essa funcionalidade do Login

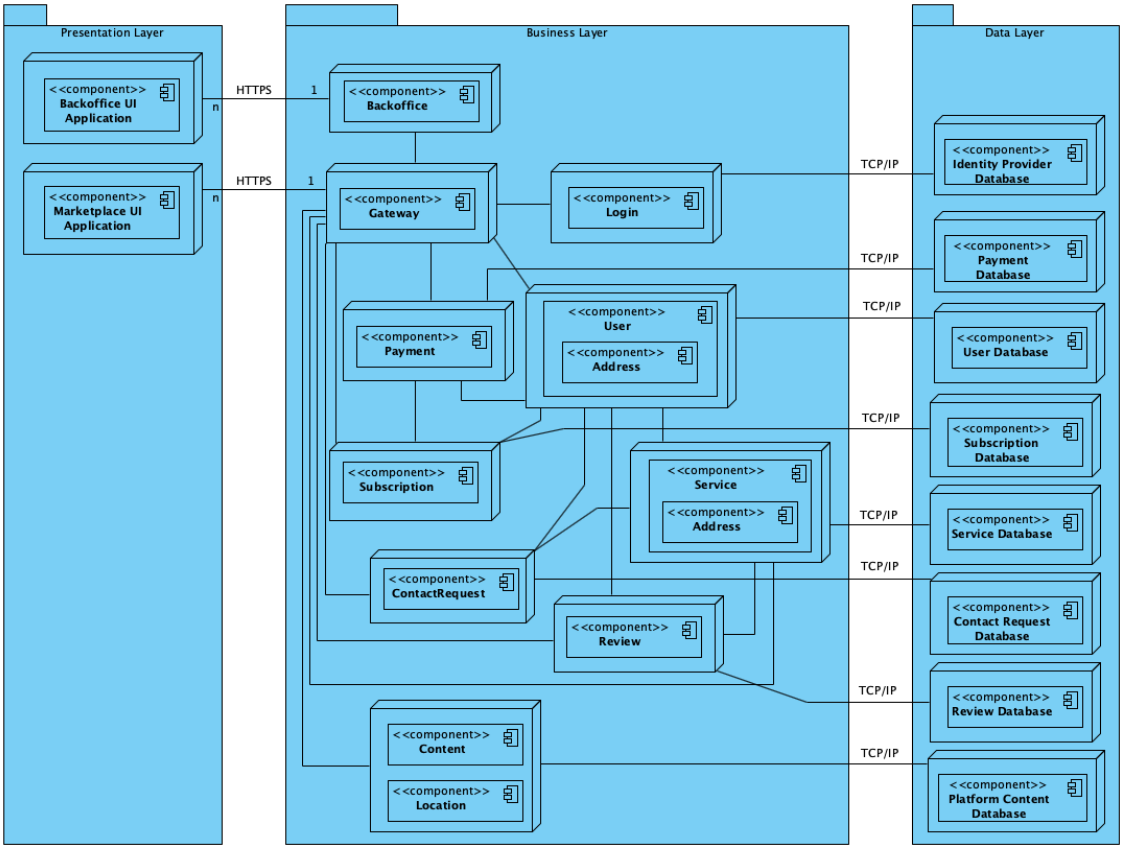


Figura 6.13: Diagrama de *deployment* do sistema de micro-serviços proposto, de acordo com a decomposição por subdomínio. Foram omitidos os micro-serviços ServiceAction e Email por simplicidade. Para não sobrecarregar o diagrama foram omitidos protocolos e número de nodos. Pode-se assumir TCP/IP e *n* em todas as omissões.

pode ser assim reutilizada. Alternativamente, os pedidos poderiam chegar ao sistema pela Gateway, a qual os passaria ao Backoffice que consumiria os restantes serviços diretamente. A vantagem da primeira abordagem é que ao expor a API do Backoffice num componente especializado torna possível expor a API apenas para a rede interna da organização, e assim melhor protegê-la.

O componente que no monólito era chamada Order, agora chama-se ContactRequest, pois foi esse o nome atribuído ao BC e o novo nome exprime melhor a sua funcionalidade. Será usado o novo nome de aqui em diante.

Decomposição por transações

Nesta abordagem o objetivo é analisar as transações entre serviços para cada chamada, e criar as divisões de maneira a minimizar o número de transações. Para esse propósito nesta subsecção toma-se vantagem da decomposição feita anteriormente, e o diagrama da Figura 6.13 é analisado como ponto de partida para determinar as transações entre micro-serviços.

Quando um fornecedor cria um novo Serviço, este pode ser criado apenas com recurso ao micro-serviço dos Serviços. Os outros micro-serviços que precisem de saber os Serviços que existem no sistema podem ser informados através de métodos de comunicação assíncrona.

Esse tipo de interação não será contado como uma transação para esta decomposição, e então, pode-se dizer que operações CRUD de um Serviço não criam transações subsequentes.

Da mesma forma as operações CRUD de revisões apenas registam os dados em base de dados sem terem que conectar com mais nenhum micro-serviço (assumindo que todos os utilizadores e serviços já foram introduzidos nas suas bases de dados).

Por outro lado, ao carregar a página de um Serviço é necessário carregar também algumas das revisões mais recentes do Serviço. De acordo com o modelo proposto o micro-serviço dos Serviços não conhece as revisões, por isso para completar a resposta terá que consultar o micro-serviço das Reviews. Em vez de uma chamada entre micro-serviços isso poderia ser feito através da Gateway. Ela poderia saber que para este tipo de chamadas deve chamar dois serviços e combinar as respostas, no entanto, delegar esse tipo de lógica para a Gateway deve ser evitado para que ela não se torne num ponto de acoplamento forte. Também poderiam ser feitas duas chamadas a partir da UI, mas além de ser evitável fazer múltiplas chamadas para carregar uma página, essa alteração também forçaria alterações em outras partes do sistema. A API deve manter-se inalterada, tanto o quanto possível. Outra alternativa poderia ser reintroduzir a Review no BC do Service, e assincronamente, registar todas as revisões também na base de dados do Service. Dado que um Serviço pode existir na plataforma durante vários anos, e durante esse tempo pode acumular centenas ou até milhares de revisões, isto estaria na origem de uma grande duplicação de dados. O ganho dessa estratégia nem seria assim tão grande, pois apenas são devolvidas algumas revisões mais recentes. Os restantes *endpoints* que permitem obter mais revisões estão já implementados no monólito na componente Review.

No caso das revisões recebidas por utilizadores, elas não são devolvidas na resposta do pedido que devolve utilizador na API do monólito. Existem é endpoints no componente User que devolvem as revisões. Então, esses endpoints poderiam ser simplesmente movidos para o micro-serviço Review e a situação seria resolvida sem ser necessário transações entre micro-serviços, nem quebrando a API.

Quando um utilizador de negócio criar uma nova subscrição que os fornecedores possam subscrever, ela tem também que ser criada em serviços externos, como no Paypal (para que o Paypal possa gerir os pagamentos das subscrições ele tem que as conhecer). A integração com o Paypal está implementada no componente dos pagamentos, então o micro-serviço das subscrições terá que chamar o dos pagamentos a cada vez que for recebida alguma operação CRUD nas subscrições. Esta é uma operação exclusiva para os utilizadores de negócio, por isso não haverá um volume grande de chamadas deste tipo.

Quando há novos pedidos de contacto é preciso verificar o estado da subscrição do fornecedor desse serviço, por isso haverá aí uma chamada ao serviço do pagamento. Este pedido terá que ser feito de forma síncrona, e o utilizador terá que ficar à espera do resultado, pois se for detetado algum problema com a subscrição do fornecedor, o utilizador terá que ser informado que o seu pedido de contacto não foi bem sucedido.

Quando um fornecedor subscreve a um plano de pagamentos (Subscription) ou compra *tokens*, isso é logo feito contra a componente Payment, por isso o micro-serviço dos pagamentos não terá que fazer chamadas subsequentes.

Não são detetados mais casos de uso que possam ter relevância arquitetural a nível de transações entre micro-serviços. Então parecem surgir dois núcleos de dependências em torno do micro-serviço dos pagamentos e dos serviços. Estranhamente não parecem surgir

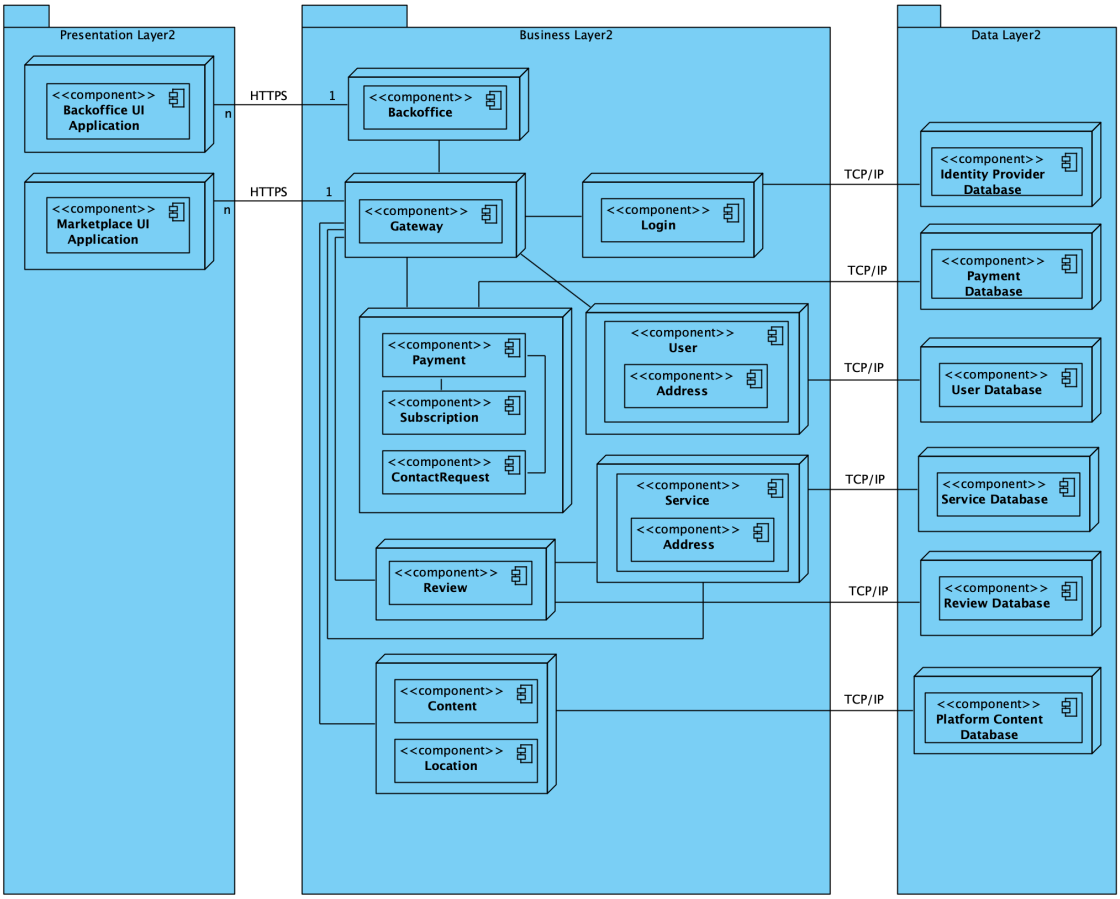


Figura 6.14: Diagrama de *deployment* do sistema de micro-serviços proposto, de acordo com a decomposição por transações. Para não sobrecarregar o diagrama foram omitidos protocolos e número de nodos. Pode-se assumir TCP/IP e *n* em todas as omissões.

dependências no micro-serviço dos utilizadores, quando em comparação com a análise do monólito, muitas componentes precisavam de consumir o componente User. Isso devia-se ao facto de que esses componentes precisavam de identificar utilizadores e por vezes de atualizar certas propriedades deles. Agora, assumiu-se que em cada BC há uma versão do utilizador com as propriedades relevantes que são geridas diretamente por cada micro-serviço.

Então, propõe-se a estrutura da Figura 6.14, onde se juntaram os componentes Payment, Subscription e ContactRequest no mesmo micro-serviço, devido ao seu relacionamento mais íntimo, permitindo assim evitar algumas transações. O componente Review continua separado do Service pois ambos já contêm um BC algo complexo. A junção iria resultar num micro-serviço já bastante complexo, quando a pequenez é um dos seus atributos mais importantes. Foi identificada apenas uma transação entre os dois, então o benefício da junção não ultrapassa o seu custo.

Escolha de uma abordagem

Têm-se então duas alternativas de como dividir este monólito em micro-serviços. A primeira alternativa, obtida a partir de decomposição por subdomínios originou um maior número de micro-serviços, e por isso haverão mais transações entre eles para resolver. Isso torna-a

também mais interessante como caso de estudo, permitindo ter mais situações onde aplicar os padrões de comunicação revistos na Secção 2.3, como composição de APIs, CQRS e Saga. No próximo capítulo mostra-se a implementação do sistema de micro-serviços de acordo com essa alternativa.

Capítulo 7

Implementação da solução

Neste capítulo é feita uma descrição da implementação do monólito e também dos passos tomados para a sua decomposição, e é descrito o sistema de micro-serviços obtido no final.

7.1 Implementação do monólito

Como já foi mencionado no Capítulo 4 foi feita uma implementação de raiz para o monólito. Essa implementação foi feita ao longo de cinco meses por uma equipa de quatro elementos, usando a *framework* Spring Boot e linguagem Java.

Inicialmente os requisitos foram divididos em pequenas tarefas (com a duração prevista de 1 a 2 dias), e o trabalho foi organizando usando a metodologia ágil Scrum [73]. A partir do segundo mês foram feitas demonstrações do sistema e dos progressos ao cliente no final de cada *sprint*.

Durante o desenvolvimento foi usada a estratégia de *branching* Gitflow [74]. Em cada tarefa foi criado um *Pull request* (PR) para fazer o *merge* para o ramo de desenvolvimento. Esse PR era manualmente revisto pelos outros elementos da equipa, e apenas podia ser feito o *merge* caso fosse aprovado. No momento que o PR é aberto eram lançados os testes unitários e de integração, e era lançada a verificação de código do Sonar Cloud [75]. Estas verificações são o primeiro passo de CI [76].

Os seguintes passos da prática de CI adoptada eram correr toda a *build* após o *merge* para o ramo de desenvolvimento (incluindo testes e sonar), e implantar a aplicação. A *build* de CI foi realizada usando as ferramentas de *build* do Bitbucket [77].

A *build* do projeto foi criada usando gradle [78], a virtualização do ambiente para correr a aplicação com Docker [79], e o versionamento da base de dados é feito usando Liquibase [80].

Para o ambiente de produção a implantação foi usada a plataforma DigitalOcean [81] devido aos baixos custos, e usando uma *elastic pool* para tentar garantir a escalabilidade.

Foram usadas algumas APIs externas. O envio de emails foi configurado para usar o servidor SMTP da Gmail [82]. Foram usadas as plataformas de pagamento Paypal [83] e Paystack [84]. A Geocoding API da Google [85] foi usada para obter as coordenadas geográficas das moradas introduzidas pelos utilizadores, e a API do Datacenter da Digital Ocean foi usada para armazenar as imagens que os fornecedores queiram adicionar aos seus serviços.

O Paypal foi integrado usando as bibliotecas para o efeito [86, 87], tal como a API da Digital Ocean [88]. O Paystack e a API de Geocoding foram integrados através da implementação das chamadas HTTPS usando a biblioteca Okhttp [89].

Os testes unitários foram criados usando a *framework* JUnit 5 [90] e as funcionalidades de mocking foram garantidas criadas usando Mockito [91]. Os de integração foram criados usando uma estratégia de lançar o servidor (com configurações de ambiente de teste) e executar pedidos HTTP contra o mesmo. Os pedidos HTTP são lançados usando as funcionalidades da biblioteca REST-assured [92], que permite não só lançar os pedidos como de correr verificações sobre as respostas. O envio de emails no ambiente de teste está configurado para usar as funcionalidades do servidor SMTP mock SubEtha Wiser [98], que permite simular o envio de emails e correr verificações contra os emails recebidos; e, é usada neste ambiente a base de dados em memória H2 [93].

7.2 Implementação do sistema de micro-serviços

O sistema de micro-serviços foi implementado apenas pelo autor deste documento, para o propósito da sua avaliação e comparação com o monólito. Todas as tecnologias usadas no monólito foram mantidas, tendo-se apenas dividido as suas funcionalidades por vários projetos, cada um no seu repositório.

A abordagem usada para a implementação do sistema de micro-serviços foi de em primeiro lugar apenas separar as funcionalidades do monólito, mantendo-as o mais parecidas possível ao que já existia, e fundamentalmente, mantendo a API exposta. O primeiro passo foi segregar cada um dos micro-serviços em isolamento dos outros, deixando a comunicação entre eles para um segundo passo, onde já se terá uma ideia mais completa das necessidades reais. Por fim, foram configuradas a virtualização do sistema e a *gateway*.

Nesta secção listam-se as componentes identificadas no capítulo anterior, e descreve-se a forma como evoluíram para o sistema de micro-serviços.

7.2.1 Login

O primeiro componente separado do monólito foi o Login, e originou um micro-serviço chamado de MicroAuth. Este contém as funcionalidades do login, registo de utilizadores, e recuperação de password. A sua principal responsabilidade é então de gerir as operações que necessitam de tratar da palavra-passe dos utilizadores. Mais nenhum micro-serviço terá conhecimento das palavas-passe.

O endpoint para alteração da palavra-passe encontrava-se no componente User do monólito, mas foi movido para o micro-serviço MicroAuth.

O diagrama de classes implementado no MicroAuth e a API REST exposta encontram-se no Anexo C.

7.2.2 Email

Foi tomada a decisão de não implementar um micro-serviço dedicado ao envio de emails. A componente Email contém apenas uma classe que contém a lógica de envio de emails, e o resto são apenas DTOs que servem para conter a informação a ser enviada no email. Se esses DTOs fossem mantidos do lado do hipotético micro-serviço Email, a cada vez que mudasse

algo relacionado com as funcionalidades que causam o envio de emails, poderia ser necessário mudar também o micro-serviço Email, tornando-o assim num ponto de acoplamento entre os demais. Alternativamente, distribuindo os DTOs pelos outros micro-serviços, deixaria o micro-serviço Email com pouco mais que uma classe, não tornando justificável o esforço de manter e implantar um mirco-serviço só para isso. Então foi tomada a decisão que cada um dos micro-serviços terá que manter a sua própria lógica de envio de emails, a qual poderá evoluir consoante as necessidades de cada um deles.

7.2.3 User

O micro-serviço que gere os utilizadores foi chamado de MicroUser. Existem alguns end-points que devolvem DTOs com informação muito completa sobre o utilizador, como as suas Subscriptions, Reviews e Services. Essa informação terá que ser obtida dos outros micro-serviços através do padrão de composição de APIs, como descrito na secção 2.3.8. Alternativamente, poderia ser apenas devolvido o link para obter cada um desses recursos através de HATEOAS, mas isso alteraria a API.

O diagrama de classes implementado no MicroUser e a API REST exposta encontram-se no Anexo D.

7.2.4 Service

O micro-serviço que foi chamado de MicroService gere os Services, a pesquisa de Services, as Whishlists e as Categories. Neste micro-serviço é necessário ter conhecimento da avaliação média das Reviews e do número de ContactRequests de cada Service, para que possam ser usados nos filtros da procura. Como esta funcionalidade deve ter o menor tempo de resposta possível não se pode considerar composição de APIs para obter esses dados. Dessa forma, quando houver uma alteração nas Reviews e ContactRequests, essa informação deve ser enviada para o MicroService e armazenada na sua base de dados, o que pode ser feito de forma assíncrona.

O MicroService implementa também uma versão menos complexa do User, apenas com as propriedades necessárias para suportar as funcionalidades do MicroService, tal como se encontra no BC do serviço na Figura 6.12. Então, quando os Users forem criados ou modificados, essa informação terá que ser integrada com o MicroService, o que também pode ser feito de forma assíncrona.

A integração assíncrona pode ser feita usando a estratégia de *event sourcing*. Dessa forma, o MicroUser terá que emitir eventos a cada vez que um utilizador for criado, modificado ou removido, para que os outros micro-serviços que estejam interessados possam estar à escuta.

O diagrama de classes implementado no MicroService e a API REST exposta encontram-se no Anexo E.

7.2.5 Review

O micro-serviço que gere as Reviews foi chamado de MicroReview, e, de forma similar ao MicroService, implementa versões mais simples de User e Service, e precisará de receber essa informação por *event sourcing*.

O diagrama de classes implementado no MicroReview e a API REST exposta encontram-se no Anexo F.

7.2.6 Content e Location

Estas duas componentes são muito simples, gerindo apenas algumas entidades de base de dados entre si, e foram implementadas juntas no mesmo micro-serviço chamado MicroPlatformContent. Conhecem apenas uma versão muito simples do User, apenas o suficiente para que possam reconhecer se o utilizador que está a fazer modificações tem realmente permissão para isso.

O diagrama de classes implementado no MicroPlatformContent e a API REST exposta encontram-se no Anexo G.

7.2.7 Subscription e Payment

Na Figura 6.12 é possível verificar que os BCs de Subscription e Payment não têm uma sobreposição notável, no entanto, ao tentar segregar o componente de Subscription verificou-se que quase todas as operações suportadas por Subscription acabam por chamar o componente de Payment (como por exemplo, criação de subscrições pelos utilizadores de negócio, utilização de subscrições e compra de tokens pelos fornecedores). Ao separá-los significa que quase sempre que Subscription recebesse um pedido, iria ter que estender um outro ao Payment. Isso iria criar um acoplamento mais forte entre esses dois serviços, e criar mais carga e mais latência no sistema.

Desta forma, apesar dos seus BCs não estarem fortemente relacionados, ao analisar deste ponto de vista mais funcional, pode ser vantajoso juntar estas duas componentes no mesmo micro-serviço, o qual foi simplesmente chamado de MicroSubscription.

Este micro-serviço, tal como os anteriores, implementa uma versão simples do User, apenas com as propriedades que suportam as funcionalidades deste micro-serviço, como o mapeamento entre Users e Subscriptions, e a carteira de tokens.

O diagrama de classes implementado no MicroSubscription e a API REST exposta encontram-se no Anexo H.

7.2.8 Contact Request

O micro-serviço que gere os pedidos de contacto, chamado MicroContactRequest, tem algumas peculiaridades.

Quando o utilizador faz um pedido de contacto é lhe enviado um email com a morada do serviço pedido e o contacto telefónico do fornecedor, e é enviado também um email ao fornecedor para o informar que um pedido de contacto foi feito. Para que um pedido de contacto seja efetuado com sucesso o fornecedor do serviço tem que ter uma subscrição ativa, ou será descontado um token da sua carteira (que é gerida pelo MicroSubscription).

Há muitas abordagens que se podem implementar para resolver este problema. Se a consistência do número de tokens for muito importante pode-se implementar uma estratégia bloqueante, como por exemplo, usando transações distribuídas. No entanto, esse não é o caso. Os tokens são um bem virtual, se em algum momento forem gastos mais do que os que existem, o único lesado é o negócio (não o fornecedor), e o valor de cada token é tão

baixo que não é uma preocupação para o negócio. Por isso podem-se explorar soluções que não garantem consistência imediata.

Uma solução simples para o problema poderia passar pelo mesmo tipo de solução mencionado acima, e aplicar *event sourcing* para o MicroSubscription enviar para o MicroContactRequest atualizações a cada vez que mudasse o estado da subscrição de um fornecedor, ou quando este comprasse mais tokens. Depois o MicroContactRequest poderia imediatamente verificar esses dados e decidir se enviaria os dados ao utilizador. A diferença principal em relação aos exemplos anteriores é que os dados recebidos por *event sourcing* eram necessários apenas para consulta. Neste caso o número de tokens precisa de ser modificado, e essa responsabilidade deve continuar do lado do MicroSubscription.

Então, foi implementada uma Saga, pois há mais que uma interação entre os dois micro-serviços. O MicroContactRequest informa o MicroSubscrition que há um pedido de contacto pendente. O MicroSubscription avalia e informa o outro de volta se há ou não condições para o pedido de contacto continuar. O MicroContactRequest pode então enviar os emails, mas se houver alguma falha que impossibilite o envio dos emails, então, se um token foi descontado, o MicroSubscription deve ser informado para o devolver.

O diagrama de classes implementado no MicroContactRequest e a API REST exposta encontram-se no Anexo I.

7.2.9 Backoffice e Action

A componente Backoffice poderia ter sido implementada em um micro-serviço. Este teria apenas a responsabilidade de reencaminhar pedidos para os micro-serviços relevantes. Acaba por apenas expor uma API diferente para ser consumida por uma aplicação diferente. Além disso a sua única responsabilidade seria assegurar que os pedidos que lhe chegam vêm com a autenticação de um utilizador de negócio. A sua baixa complexidade torna-a pouco relevante para arquitetura do sistema, e então foi decidido deixar a sua implementação de parte e manter o foco no núcleo do sistema.

Pelo mesmo motivo um possível micro-serviço para a componente Action foi deixado de fora. A sua única responsabilidade seria guardar em base de dados um log de ações que foram acontecendo no sistema.

Considera-se que deixar estas componentes de fora da implementação não terá um grande impacto na avaliação e comparação das duas alternativas (monólito e micro-serviços), no entanto, idealmente elas teriam sido incluídas.

7.3 Comunicação entre serviços

Optou-se por uma comunicação entre serviços por coreografia usando principalmente *event sourcing* e transações BASE para espalhar modificações de dados por todo o sistema. Assim, será minimizado o número de chamadas bloqueantes e a latência do sistema, abraçando assim o conceito da consistência eventual. Isso pode ser feito pois não existe nenhum requisito que necessite de consistência imediata.

A comunicação assíncrona tem as suas vantagens, no entanto não é útil para todos os cenários. Cada micro-serviço deve ter na sua base de dados a informação que necessita para o seu funcionamento, essa é a informação que é espalhada pelo sistema usando comunicação

assíncrona. Por exemplo, um novo utilizador regista-se. Essa operação encontra-se implementada no MicroAuth mas todos os outros precisam de ser informados que há um novo utilizador no sistema, e isso pode ser feito assincronamente.

Por outro lado, existem alguns endpoints que devolvem uma vista de um objeto que inclui informação que não existe no micro-serviço onde o endpoint está implementado. Nesse caso, essa informação pode ser obtida sincronamente dos outro micro-serviços através da estratégia de composição de APIs. Por exemplo, em MicroUser, existe uma vista dos utilizadores de tipo provider que inclui os seus serviços, um sumário das reviews que receberam, e um resumo do estado da sua subscrição. Toda essa informação é obtida sincronamente a partir dos respetivos micro-serviços, pois não existe na base de dados de MicroUser por não ser fundamental para o seu funcionamento. Alternativamente, todos esses dados poderiam ser enviados assincronamente para MicroUser e armazenados na base de dados, mas isso apenas contribuiria para a duplicação de informação, sem qualquer benefício nem necessidade.

7.3.1 Comunicação assíncrona

É necessário escolher uma ferramenta para implementar a comunicação por *event sourcing*. Existem algumas ferramentas *open source* no mercado tais como:

RabbitMQ É um *message broker* de propósito genérico que não suporta ordenação das mensagens nem as persiste por muito tempo. Uma vez consumida uma mensagem ela é apagada [94]. Por isso, não é tão apropriado para o propósito de *event sourcing*.

Apache Kafka É uma ferramenta de *event streaming* que permite ordenar as mensagens e guardá-las com um tempo de vida mais longo, como um log [94]. Pode ser usado para *event sourcing* com alguns ajustes e pode ter algumas limitações nesse aspeto [95, 96].

Axon É uma plataforma construida especificamente para *event sourcing* e suporta vários tipos de mensagens para esse propósito (comandos, *queries* e eventos) [95].

Foi escolhido o Kafka para a comunicação assíncrona pois é perfeitamente adequado para as necessidades de comunicação deste projeto, e é usado por um grande número de empresas [97], por isso tem uma grande comunidade na Internet, o que também são fatores interessantes para se ter em consideração ao escolher uma nova tecnologia.

Utilizou-se o serviço Confluent Cloud para criar um cluster de Kafka na nuvem e assim garantir a comunicação assíncrona no sistema. Podem ver-se os tópicos que foram criados nesse cluster na Figura 7.1. Cada micro-serviço tem a responsabilidade de criar os tópicos que usa (se ainda não existirem), e pode consumir e produzir para qualquer deles.

7.4 Base de dados

Existe apenas uma instância de Mysql, mas contém uma base de dados para cada micro-serviço. Cada micro-serviço usa o seu próprio utilizador, o qual tem acesso apenas à sua própria base de dados. Assim, efetivamente, cada micro-serviço tem a sua base de dados isolada dos outros, e, futuramente, elas podem ser separadas cada uma para a sua própria instância com apenas uma modificação nas configurações dos micro-serviços. Um excerto do script que é corrido ao iniciar o container da base de dados encontra-se na Listagem 7.1.

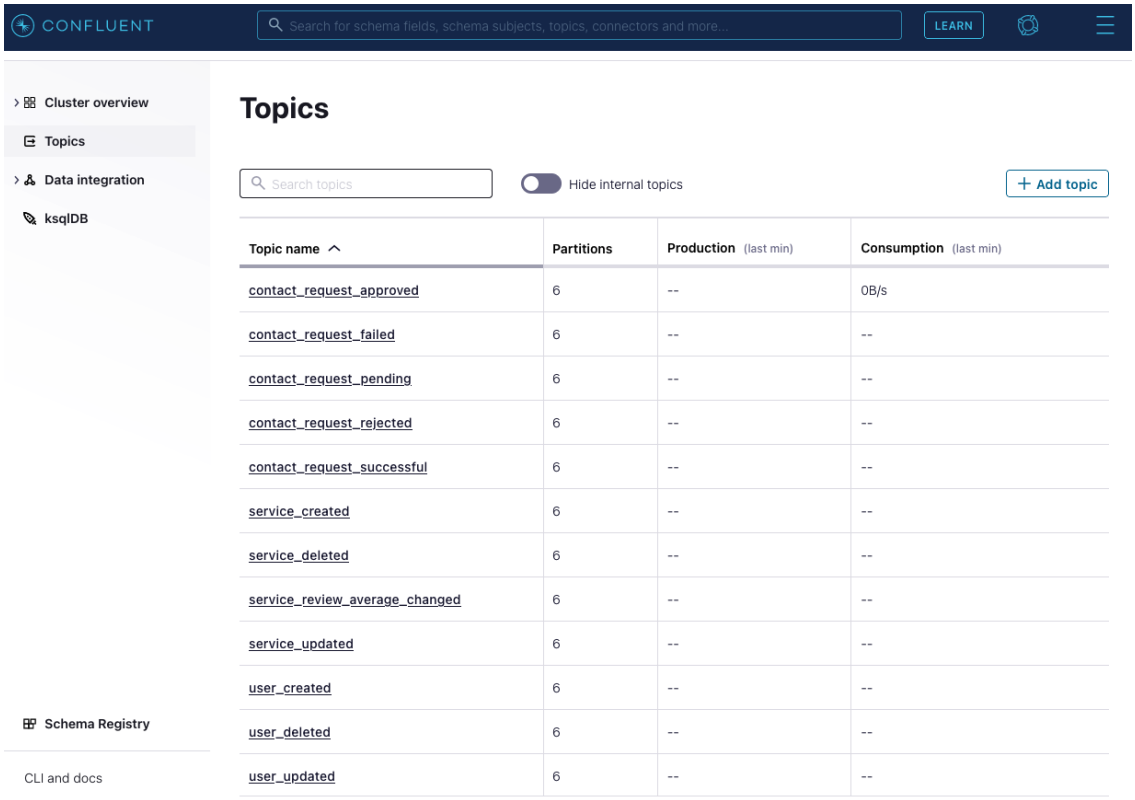


Figura 7.1: Vista dos tópicos de Kafka existentes no cluster do Confluent Cloud.

```sql
CREATE DATABASE IF NOT EXISTS micro_review_db;

CREATE USER 'micro_review_dbuser'@'%' IDENTIFIED BY 'pw';
GRANT ALL PRIVILEGES ON micro_review_db.* To 'micro_review_dbuser'@'%';

FLUSH PRIVILEGES;
```

Listing 7.1: Excerto do script de criação inicial de bases de dados e utilizadores

7.5 Gateway

A *gateway* foi implementada utilizando uma instância de Nginx que está configurada para redirecionar os pedidos que lhe chegam para os micro-serviços, passando-lhes o pedido sem alteração. Um excerto da configuração encontra-se na Listagem 7.2.

```
upstream micro-review-upstream {
    server micro-review:8087;
}

server {
    listen 8080;

    location ~ ^/api/review.*$ {
        proxy_pass http://micro-review-upstream;
        proxy_http_version 1.1;
```

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection keep-alive;
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Authorization $http_authorization;
        proxy_pass_header  Authorization;
    }
}
```

Listing 7.2: Excerto da configuração do Nginx

7.6 Virtualização

Foram virtualizados todos os micro-serviços, a base de dados e a *gateway* utilizando o Docker. Foi criado um ficheiro docker-compose para poder levantar e baixar todo o sistema de uma vez, onde se definem todos os *containers* e as suas propriedades, tais como as portas que expõem e os seus hostnames; bem como as suas dependências e a network do Docker que usarão para comunicar. Um excerto desse docker-compose encontra-se na Listagem 7.3.

```
  micro-review:
    build:
      context: ../micro_review
      dockerfile: ../micro_review/dockerfile
    container_name: micro-review
    image: micro-review:latest
    hostname: micro-review
    ports:
      - 8087:8087
    expose:
      - "8087"
    networks:
      - local
    depends_on:
      - mysql
```

Listing 7.3: Excerto do docker-compose

7.7 Testes

O monólito implementava já testes unitários e de integração para todas as suas funcionalidades. Em cada micro-serviço mantiveram-se aqueles que testam as funcionalidades desse micro-serviço.

Em adição a isso, foram criados alguns testes End-to-end que testam algumas funcionalidades chamando a API REST pela *gateway*, e permitem assim testar a integração entre os serviços, tal como descrito na Secção 2.3.13.

Não foram implementados testes de UI pois a interface do utilizador não está no âmbito deste trabalho.

7.8 Solução final

No final, foram implementados a *gateway* e os sete micro-serviços: Auth, User, Service, Contact Request, Review, Subscription, e Platform Content. De seguida feita a análise do sistema.

7.8.1 Granularidade de aplicação

Todos os micro-serviços têm uma estrutura interna parecida, por isso a análise na granularidade de aplicação pode ser aplicada a todos eles.

Ponto de vista lógico

A estrutura interna dos micro-serviços é a mesma estrutura interna das componentes do micro-serviço, e o diagrama da Figura 6.5 é aplicável.

Ponto de vista processo

A simples estrutura por camadas dos micro-serviços leva a um diagrama de sequência também simples, e encontra-se na Figura 7.2.

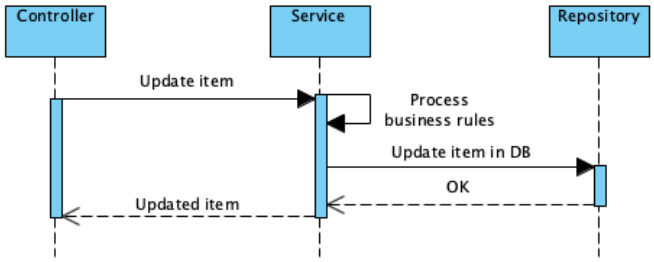


Figura 7.2: Diagrama de processo na granularidade de aplicação dos micro-serviços.

Ponto de vista implementação

O ponto de vista de implementação acaba por ser igualmente simples, e encontra-se na Figura 7.3.

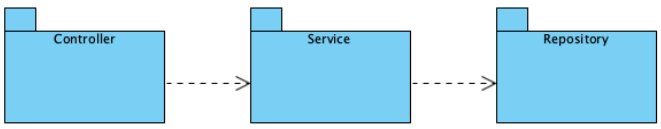


Figura 7.3: Diagrama de pacotes na granularidade de aplicação dos micro-serviços.

Ponto de vista implantação

Cada micro-serviço está implantado dentro de um *container* de Docker, tal como se encontra na Figura 7.4.

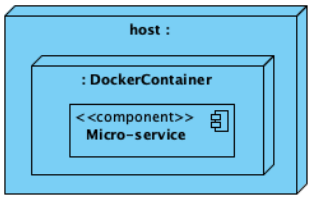


Figura 7.4: Diagrama de implantação na granularidade de aplicação dos micro-serviços.

7.8.2 Granularidade de sistema

Nesta secção é feita a análise na granularidade de sistema.

Ponto de vista lógico

Existem então no sistema os sete micro-serviços mencionados no início desta secção. A *gateway* consome todos eles e por sua vez é consumida pelas aplicações de frontend. Cada micro-serviço consome a sua própria base de dados, e têm também algumas inter-dependências: MicroUser consome MicroSubscription, MicroReview, MicroService e Micro-ContactRequest; e, MicroService consome também o MicroReview.

Estas dependências entre micro-serviços são implementadas através das chamadas bloqueantes para que os micro-serviços possam aceder a informação gerida por outros. Para a comunicação assíncrona todos os micro-serviços consomem o Kafka, mas isso foi deixado de fora do diagrama na Figura 7.5, pois a sua implementação e implantação não é da nossa responsabilidade, e pode ser tratado como um serviço externo. Os restantes serviços externos foram também deixados de fora do diagrama pois são os mesmos que já tinham sido mencionados no diagrama da Figura 6.7.

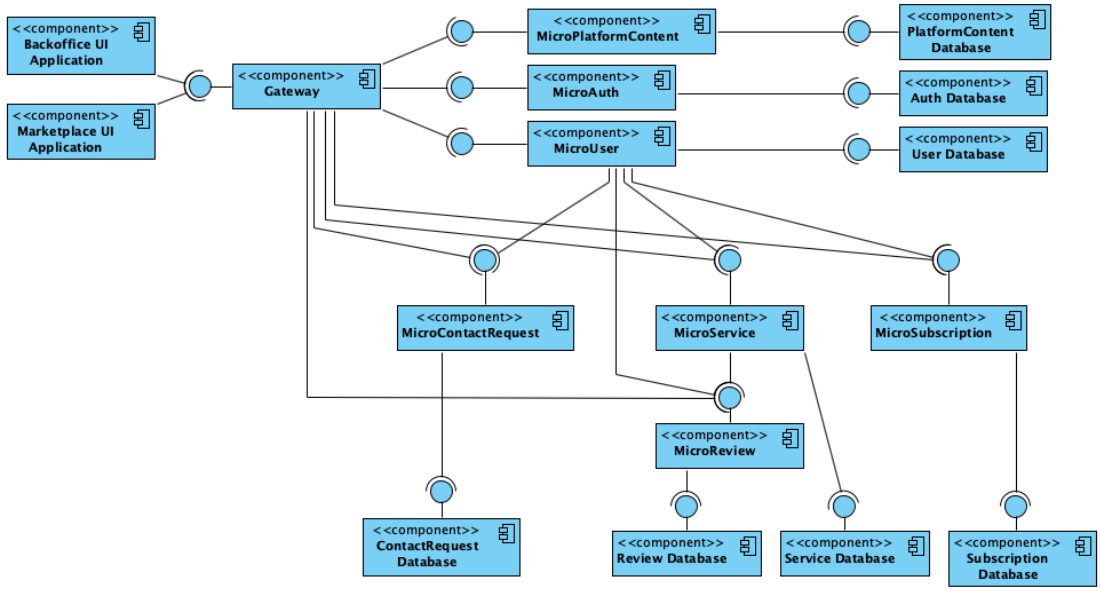


Figura 7.5: Diagrama de componentes na granularidade de sistema dos micro-serviços.

Ponto de vista processo

Na Figura 7.6 é possível ver um exemplo interessante de um processo que usa comunicação assíncrona que é a saga para a criação de pedidos de contacto, entre os micro-serviços MicroContactRequest e MicroSubscription, mencionada na Secção 7.2.8.

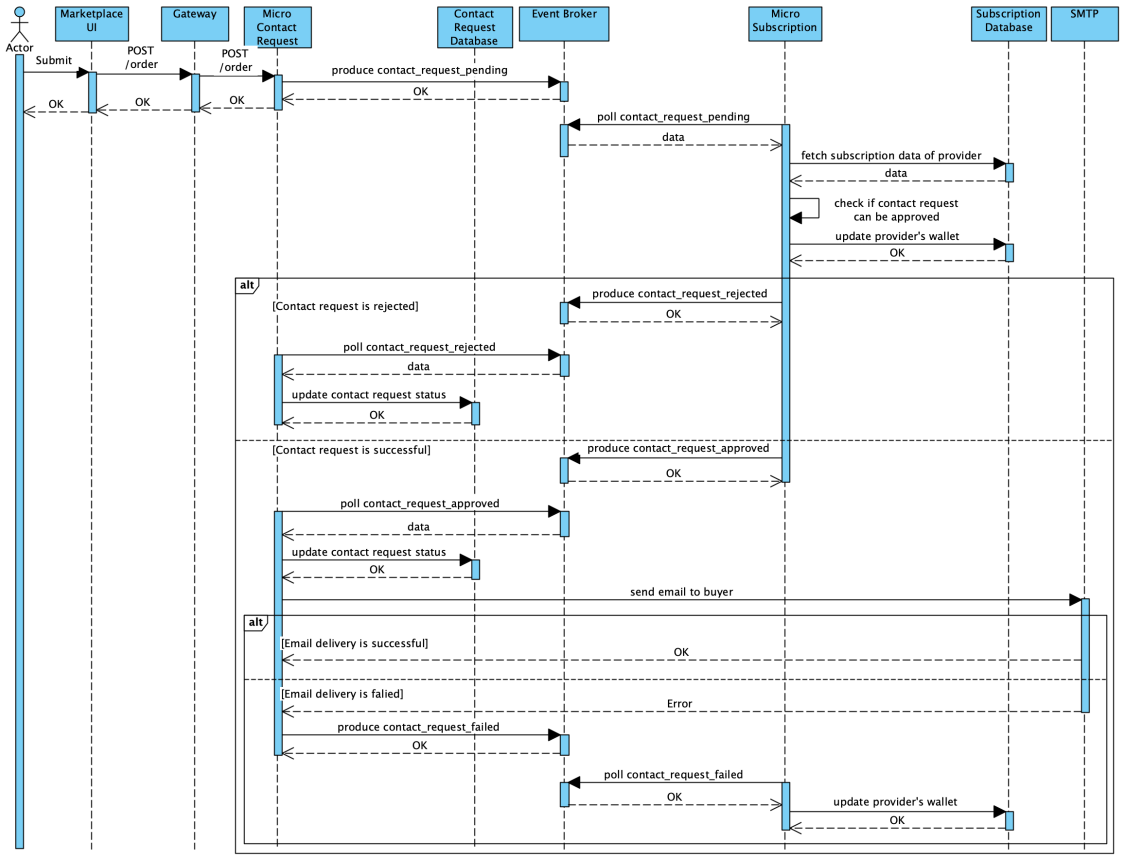


Figura 7.6: Diagrama de sequência na granularidade de sistema dos micro-serviços, mostrando a saga para a criação de pedidos de contacto. As palavras *poll* e *produce* significam que um tópico de Kafka está a ser consultado ou se está a produzir uma nova mensagem para esse tópico.

Na Figura 7.7 encontra-se um diagrama de sequência onde se mostra a composição de APIs usada para completar todos os dados do perfil do utilizador, quando este pretende ver o seu próprio perfil. É necessário consultar o MicroSubscription para obter o estado da sua subscrição, o MicroContactRequest para obter a lista dos seus pedidos de contacto, e o MicroReview para obter um resumo das revisões criadas por si e as que recebeu.

A diferença entre as duas estratégias é aqui notável. Na composição de APIs o utilizador tem que ficar a espera que todos os pedidos se completem para poder finalmente ver a informação. Tanto a Marketplace UI, como a Gateway e o MicroUser têm que ficar bloqueados à espera da resposta. Usando comunicação assíncrona, o utilizador recebe uma resposta muito mais rápida ao seu pedido inicial, mas ainda não sabe o resultado. Pode ir consultando o estado do pedido de contacto, e será eventualmente informado do resultado do pedido, mas não há nenhum componente que fique bloqueado a espera de respostas de mais que uma chamada.

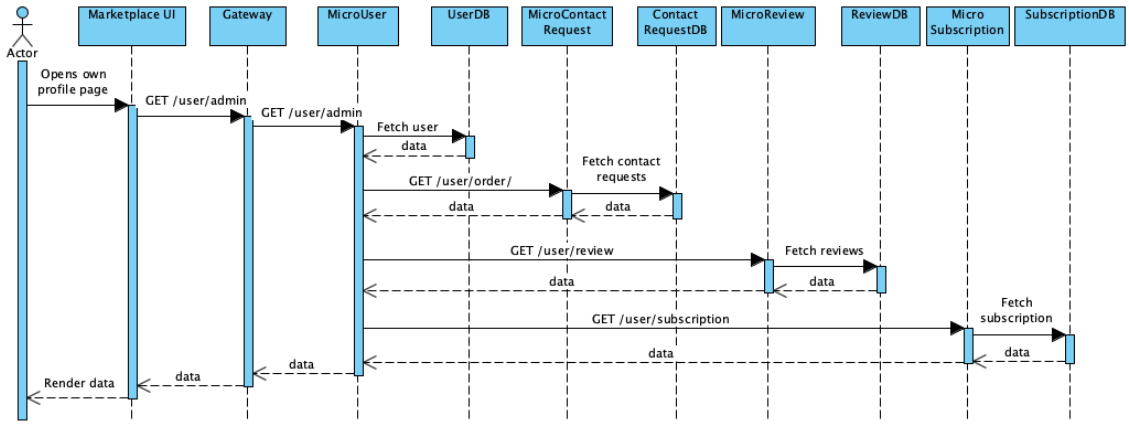


Figura 7.7: Diagrama de sequência na granularidade de sistema dos micro-serviços, mostrando a composição de APIs usada para um utilizador ver o seu próprio perfil.

Ponto de vista implementação

O diagrama da Figura 7.8 contém o diagrama de pacotes que mostra as dependências entre os micro-serviços.

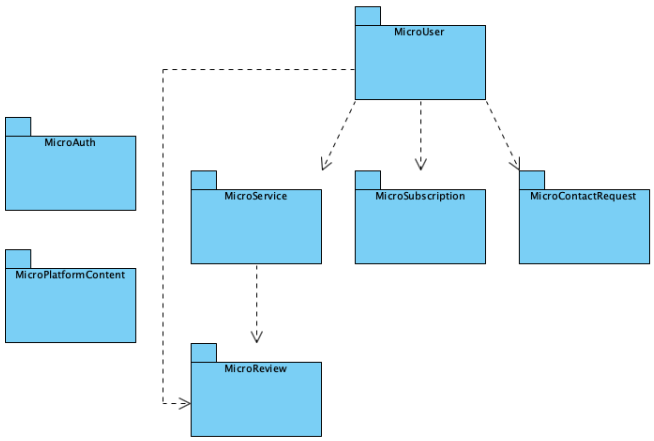


Figura 7.8: Diagrama de pacotes na granularidade de sistema dos micro-serviços.

Ponto de vista implantação

Não foi configurada a implantação do projeto de micro-serviços. Apenas existem as configurações para o ambiente local, onde todos os micro-serviços correm na mesma máquina. Esse ambiente encontra-se representado no diagrama da Figura 7.9.

Num ambiente de produção, idealmente ter-se-ia uma instância de cada micro-serviço por máquina, e cada base de dados poderia estar também na sua própria instância. Fazendo a experiência mental de como poderia ser a implantação do ambiente de produção propõe-se o diagrama da Figura B.1, que se encontra no Anexo B.

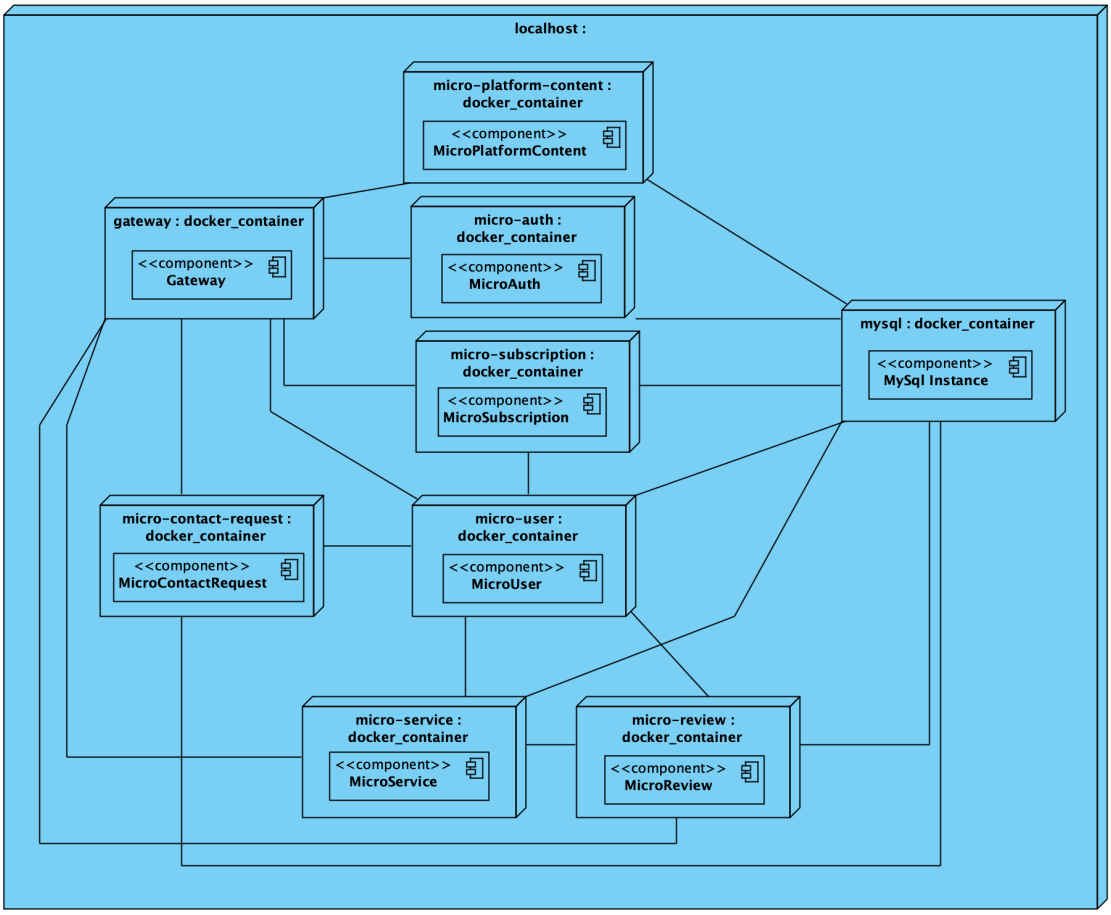


Figura 7.9: Diagrama de implantação na granularidade de sistema dos micro-serviços, para o ambiente local. Os protocolos e número de máquinas foram omitidos por simplicidade, em todos os casos existe apenas um *container* Docker de cada tipo e pode assumir-se o protocolo TCP/IP para as comunicações com a base de dados e HTTP para as restantes.

7.9 Melhorias/trabalho futuro

A *gateway* apenas redireciona o pedido para os micro-serviços, enviando o Header de autorização junto com o pedido. Cada micro-serviço então tem que descodificar o token de autorização para descobrir qual utilizador fez o pedido. Assim, a estratégia de codificação do token torna-se num ponto de acoplamento entre os micro-serviços. Se a equipa que gere o MicroAuth decidir alterar essa estratégia isso tem um impacto em todo o sistema, e a implantação da alteração teria que ser feita ao mesmo tempo em todos os micro-serviços, pois atrasos poderão resultar em indisponibilidade para os utilizadores autenticados. A forma de melhorar a *gateway* seria implementá-la como uma *gateway* de SSO, como descrito na Secção 2.3.15.

As chamadas HTTP entre micro-serviços são feitas sem qualquer tipo de autenticação, o que é um padrão a evitar. A segurança nessa comunicação poderia ser assegurada usando por exemplo Chaves de API, como foi descrito na Secção 2.3.15.

O MicroUser implementa o padrão de composição de APIs para complementar a informação em alguns dos DTOs que devolve nos seus endpoints. Em um desses DTOs precisa de

fazer chamadas a ambos o MicroSubscription, o MicroReview e o MicroService. Por sua vez, o MicroService, para complementar o DTO que precisa de devolver consulta também o MicroReview. Esta situação surge pois foi dada prioridade a manter a API que era exposta no monólito, e então cada micro-serviço localmente constrói os seus DTOs, indo retirar a informação que necessita aos outros. Assim, acabou por surgir no panorama geral esta dupla consulta ao MicroReview, que deveria ser otimizada criando endpoints específicos para este caso de uso.

A Saga implementada para o pedido de contacto, entre o MicroContactRequest e o MicroSubscription não implementa qualquer mitigação das anomalias. Uma ou mais estratégias descritas na Secção 2.3.10.

O MicroService é o micro-serviço mais complexo do sistema, e o uso potencialmente assimétrico das suas operações de busca e de modificação torna-o interessante para a aplicação de CQRS, como descrito na Secção 2.3.9, e a sua implementação poderia ser dessa forma melhorada.

Não foi implementada qualquer monitorização para o sistema de micro-serviços, sendo que esta é fundamental para um sistema deste tipo, tal como foi descrito na Secção 2.3.14.

Existe apenas configurações para o ambiente local. Isso deveria ser ampliado para a criação de mais ambientes de forma aplicar as práticas de CI e CD, tal como descrito na Secção 2.3.12.

Foram implementados alguns testes End-to-end, mas o seu número deveria ser ampliado, bem como deveria existir um ambiente de teste com uma base de dados de teste para correr esses testes End-to-end. Como há apenas o ambiente local, os testes apenas podem correr contra esse ambiente o que é limitativo. Por exemplo, torna-se impossível testar o envio de emails, o que impossibilita executar a validação de novos utilizadores (é enviado um email a novos utilizadores com um link que devem seguir para validar o perfil). Tal como é feito nos testes de integração (de cada micro-serviço e do monólito), os micro-serviços no ambiente de teste poderiam ser configurados para usar um servidor SMTP mock, tal como o SubEtha Wiser, e assim possibilitar novamente a verificação do envio de emails.

A organização de certos elementos do sistema pode também ser melhorada. Foi criado um repositório chamado marketplace_gateway que reúne não só as configurações da *gateway*, mas também o ficheiro docker-compose, o script mysql de criação de utilizadores e bases de dados, os testes End-to-end e até um shell script (presente na Listagem 7.4) para correr as tasks gradle de todos os micro-seviços para gerar os jars (para serem copiados para os *containers* do docker). Além disso, o docker-compose e o bash script dependem dos outros micro-serviços estarem todos na mesma pasta mãe junto com o marketplace_gateway para conseguirem usar os dockerfiles e tasks gradle, respetivamente, o que não é de todo uma dependência que se deva ter.

```
cd "../micro_auth" && ./gradlew clean assemble
cd "../micro_user" && ./gradlew clean assemble
cd "../micro_service" && ./gradlew clean assemble
cd "../micro_contact_request" && ./gradlew clean assemble
cd "../micro_subscription" && ./gradlew clean assemble
cd "../micro_review" && ./gradlew clean assemble
cd "../micro_platform_contents" && ./gradlew clean assemble
```

Listing 7.4: Shell script que corre as tasks gradle.

Finalmente, e não menos importante, falta ainda implementar micro-serviços para o Backoffice e o Action e assim completar os requisitos funcionais da aplicação.

Capítulo 8

Avaliação

Neste capítulo é feita a análise do sistema de micro-serviços e este é comparado com o monólito. Pretende-se avaliar o aumento total de tamanho e complexidade, o ganho de performance, e, o impacto do uso de métodos de comunicação assíncronos.

Para o aumento do tamanho do sistema é avaliada a evolução do número de linhas de código. Para avaliar o aumento da complexidade é feita a contagem das classes com elevada complexidade ciclomática e é avaliada a evolução da dívida técnica. O ganho de performance é avaliado usando testes de carga em ambiente local e na nuvem. Por fim, o impacto dos métodos de comunicação assíncronos é avaliado medindo o atraso entre a criação de um item em um micro-serviço e o seu aparecimento em outro.

8.1 Aumento do tamanho

A passagem para micro-serviços deixou o sistema como um todo com mais linhas de código para serem mantidas. Isto deve-se à duplicação de classes como DTOs que precisam de existir em mais que um micro-serviço, e também devido à duplicação de configurações. Desde a *buid* gradle, aos *scripts* da base de dados, as configurações de CORS, de segurança, do Kafka, do Liquibase, entre outros.

Usando um *plugin* de análise no IntelliJ chamado Statistic [99] foram identificados os dados da Tabela 8.1. Na tabela é possível ver o número de linhas de código de várias linguagens no monólito, em todos os micro-serviços, e a soma das linhas de código de todos os micro-serviços.

O código java implementa a lógica de negócio, o código properties serve para configurar todos os ambientes, o código sql é usado para introduzir dados de teste na base de dados para os testes de integração, o código xml serve para as definir a estrutura das tabelas para o liquibase e os ficheiros yaml são usados para introduzir algumas configurações para a aplicação e para o Docker.

	java	properties	sql	xml	yaml
Monólito	33784	133	522	3690	1198
MicroAuth	4727	138	82	278	242
MicroUser	5988	166	89	264	406
MicroService	8611	160	337	1551	522
MicroContactRequest	3052	158	203	666	429
MicroReview	7235	158	123	417	537
MicroSubscription	10566	158	149	348	444
MicroPlatformContent	1729	159	13	1591	178
Soma dos micro-serviços	41908	1097	996	5115	2758

Tabela 8.1: O número de linhas de código de cada linguagem em todos os projetos (monólito e micro-serviços), em comparação com a soma de todos os valores para os micro-serviços.

O monólito é bastante maior do que qualquer um dos micro-serviços, tendo na maioria dos casos mais uma ordem de grandeza do número de linhas, no entanto, juntando todos os micro-serviços, o número de linhas supera em todos os casos o monólito. Então conclui-se que cada micro-serviço é bem mais simples e por isso tem menos código para manter, mas no total, o sistema ficou maior e tem mais código para ser mantido. Essa tendência encontra-se mais graficamente exposta na Figura 8.1.

Então, um número de 33784 linhas de código java do monólito foram transformadas num total de 41908 no sistema de micro-serviços, o que representa um aumento de 24% no volume de código java que tem que ser mantido.

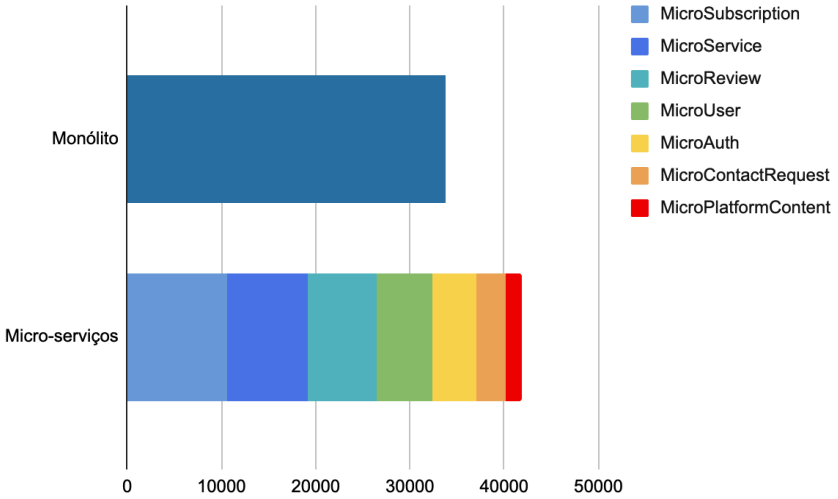


Figura 8.1: Gráfico que compara o número de linhas de código java do monólito com a combinação de todos os micro-serviços.

8.2 Aumento da complexidade

A complexidade pode ser um conceito um pouco abstrato de se medir, mas é importante conseguir-se medi-la pois a complexidade pode afetar fortemente a eficiência de um projeto. Assim, foram criadas algumas métricas que permitem estimar a complexidade de um projeto,

tais como o número de linhas de código (que já foi analisado na Secção anterior), ou a complexidade ciclomática, entre outros [100].

A complexidade ciclomática mede o número de caminhos independentes que uma *thread* pode tomar. Esta métrica assenta-se na ideia de que quantos mais forem os possíveis caminhos mais difícil o programa será de compreender e maior será o risco de mudança [101].

Nesta secção é feita uma análise da complexidade ciclomática usando o plugin CodeMR para o IntelliJ [102]. Esta ferramenta não dá uma medição numérica da complexidade, mas, a cada classe java, dá uma classificação da complexidade que pode ser: muito alta, alta, média-alta, média-baixa ou baixa. O relatório completo da análise encontra-se no Anexo J.

Contando o número de classes de cada um tipo de classificação de complexidade obtém-se o resultado na Tabela 8.2. Naturalmente, cada micro-serviço tem menos classes que o monólito em todas as classificações de complexidade pois têm bastantes menos classes que o monólito. Ao somar os valores de cada classe dos micro-serviços é possível verificar que há mais classes no sistema de micro-serviços, mas aquelas que tendem a estar mais vezes replicadas são as que têm menores complexidades. Isto deve-se ao facto de que a maioria das classes replicadas em mais que um micro-serviço são DTOs, que não têm grande complexidade. Existe uma exceção notável no caso das classes de classificação muito alta pois uma das classes com essa classificação no monólito é o controlador do Backoffice, a qual não foi implementada no sistema de micro-serviços.

	Muito alta	Alta	Média-alta	Média-baixa	Baixa	Número total de classes
Monólito	2	2	16	74	229	323
MicroPlatformContent	0	0	1	7	40	48
MicroSubscription	0	1	6	41	165	213
MicroReview	1	3	4	12	93	113
MicroContactRequest	0	0	2	15	59	76
MicroService	0	0	5	18	98	121
MicroUser	0	1	3	14	108	126
MicroAuth	0	0	8	21	77	106
Soma micro-serviços	1	5	29	128	640	803

Tabela 8.2: Comparação do número de classes em cada classificação de complexidade para o monólito, para cada micro-serviço, e para a soma de todos os micro-serviços.

Na Figura 8.2 é possível ver-se uma comparação do número de classes em cada classificação entre o monólito e o sistema de micro-serviços como um todo. O aumento do número total de classes é notável, passando para mais do dobro, causado principalmente pelo aumento do número de classes de baixa complexidade. Pode assim concluir-se que o tamanho do sistema aumentou, mas a complexidade aumentou numa porção menor, tendo esta ficado mais diluída num conjunto de classes menos complexas.

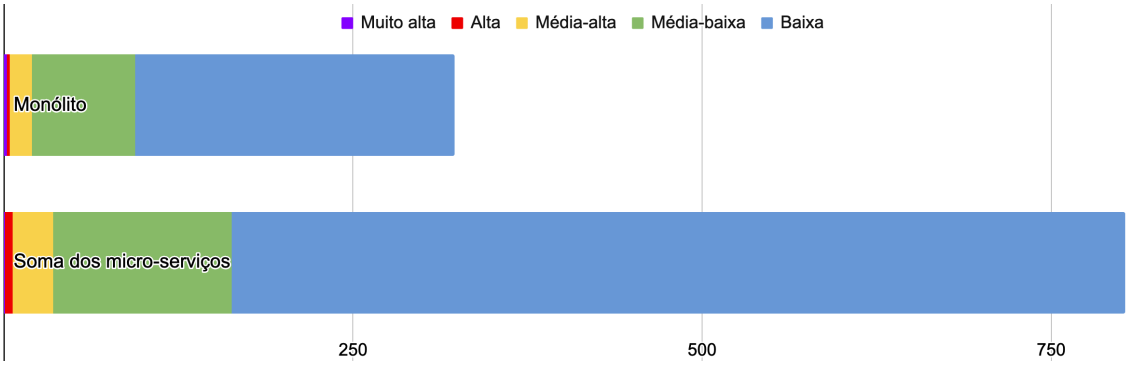


Figura 8.2: Gráfico que compara o número de classes em cada intervalo de complexidade para o monólito, e para a soma de todos os micro-serviços.

Este é um aumento significativo de 149% do número total de classes com a transformação do monólito em micro-serviços. As classes de complexidades diferentes foram afetadas de formas diferentes. O aumento do número de classes de complexidade alta foi de 150%, o de média-alta foi de 81%, o de média-baixa foi de 73%, e, o de baixa foi 179%.

8.3 Dívida técnica

Dívida técnica é um conceito um pouco abstrato, mas prende-se com o custo temporal de re-trabalhar o software devido a decisões tomadas no passado. É mais fácil de definir com um exemplo. Quando uma equipa hipotética está a trabalhar num software sob pressão, pode tomar decisões no sentido de deixar esse software a funcionar mais rapidamente. Está a poupar tempo agora, mas esse tempo terá que ser pago mais tarde quando chegar a altura de implementar modificações e surge a necessidade de refatorizar.

Ao replicar algumas classes ao longo do sistema de micro-serviços é normal replicar também alguma dívida técnica. Pode ser também interessante descobrir o aumento de dívida técnica total ao mover este projeto para micro-serviços. Essa análise foi feita usando SonarQube [103] que permite estimar o valor da dívida técnica de um projeto. Os resultados dessa análise encontram-se na Tabela 8.3.

	Dívida técnica
Monólito	3h 53min
MicroPlatformContent	2h 6min
MicroSubscription	2h 56min
MicroReview	2h 11min
MicroContactRequest	1h 54min
MicroService	4h 14min
MicroUser	3h 18min
MicroAuth	2h 3min
Soma dos micro-serviços	18h 42min

Tabela 8.3: Comparação dos tempos de dívida técnica entre o monólito, cada micro-serviço, e a soma de todos os micro-serviços.

Como seria de esperar, geralmente a dívida técnica de cada micro-serviço é menor que a do monólito em quase todos os casos. Por outro lado, é notável que a dívida técnica do sistema de micro-serviços como um todo é superior à do monólito, sendo mais de quatro vezes maior. Isto deve-se a parte da dívida técnica do monólito ser replicada por cada micro-serviço, mas também foi introduzida nova dívida técnica ao implementar a comunicação entre os micro-serviços. No caso do MicroService essa nova dívida técnica é até responsável pela dívida técnica deste micro-serviço ser ainda superior à do monólito.

A subida geral de dívida técnica encontra-se de forma mais visual no gráfico da Figura 8.3.

Tem-se assim um aumento de 381% do aumento da dívida técnica, principalmente alimentado pelo aumento do número de classes.

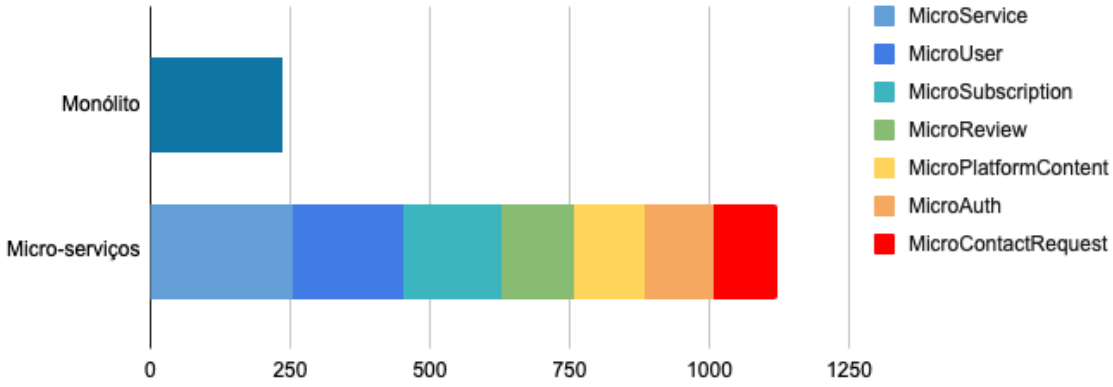


Figura 8.3: Gráfico que compara o número minutos de dívida técnica do monólito com o sistema de micro-serviços.

8.4 Comportamento sob carga

Nesta secção é analisado o comportamento sob carga dos dois sistemas com o objetivo de compreender as alterações comportamentais induzidas pela transformação do monólito em micro-serviços.

Para analisar o comportamento sob carga foi criado um plano de testes usando a ferramenta Apache JMeter [104]. Esta ferramenta pode ser usada para definir conjuntos de pedidos para serem lançados contra o sistema, e definir verificações contra as suas respostas. Os pedidos podem ser agrupados em conjuntos de *threads* (cada *thread* representa um utilizador). As *threads* podem ir aumentando em número ao longo do tempo e cada uma repete um determinado número de vezes o conjunto de chamadas.

Para comparar o monólito e os micro-serviços foi criado um plano de testes com quatro grupos de *threads* chamados: compradores, fornecedores, registos e serviços. O grupo dos compradores serve para simular algumas ações dos compradores: login, ver o próprio perfil, procurar serviços, adicionar à *wishlist*, criar pedido de contacto, criar revisão sobre o serviço, e ver revisões. O grupo dos fornecedores serve para simular algumas ações dos fornecedores: login, ver o próprio perfil, criar serviço, modificar serviço, ver revisões e subscrições. O grupo dos registos serve apenas para fazer alguns pedidos de login e registo de compradores e fornecedores. No caso dos micro-serviços estes registos são espalhados assincronamente por todo o sistema por isso porão carga em todo o sistema. O grupo dos serviços apenas

serve para gerar pedidos GET para obter serviços e fazer pesquisas (com e sem filtros), de forma a simular uma carga mais pesada sobre a parte do sistema que gere os serviços.

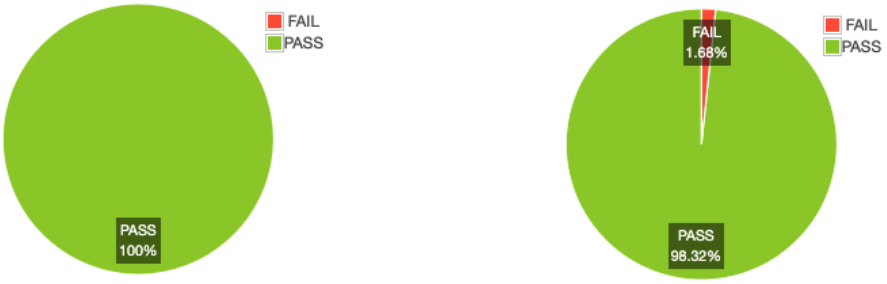
Cada chamada feita contém uma asserção para verificar que o código da resposta é o esperado.

8.4.1 Em ambiente local

Os testes foram executados em ambiente local, onde o monólito, o sistema de micro-serviços, a base de dados e a *gateway* se encontram implantados em *containers* Docker. Neste ambiente os micro-serviços têm que partilhar os recursos da mesma máquina, e assim poderá haver um maior *overhead* por ter todas essas aplicações a correr.

O grupo dos compradores foi utilizado com 800 *threads* concorrentes, o dos fornecedoras com 400, o dos registos com 200 e o dos serviços com 1000. Cada grupo teve um tempo de 240 segundos até criar todas as suas *threads* e cada uma executou as suas chamadas por duas vezes.

Sob estas condições surgiram várias diferenças no comportamento dos dois sistemas. Começando pelo número de respostas de erro, que se encontra na Figura 8.4. No caso do monólito não foram registados quaisquer erros, enquanto que no caso dos micro-serviços houve uma pequena percentagem.

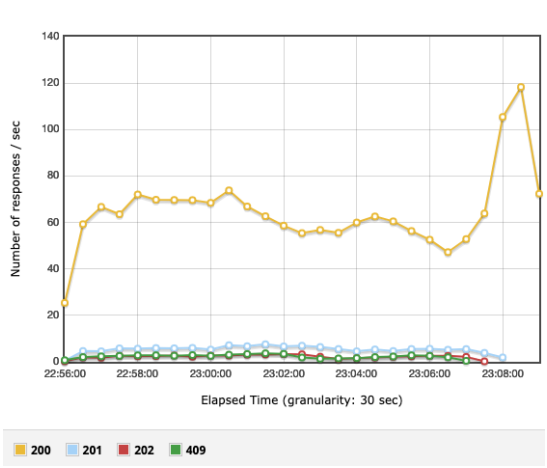


(a) No caso do monólito os pedidos tiveram 100% de sucesso.

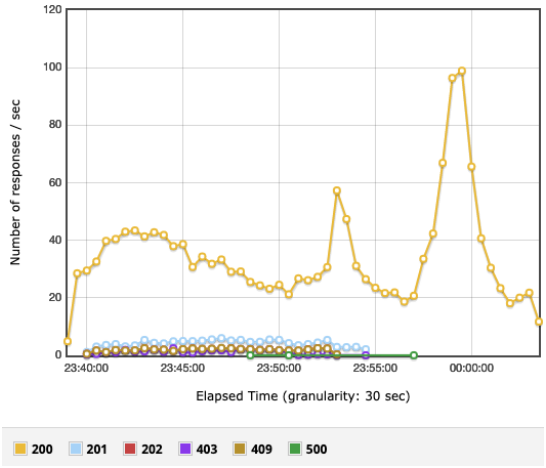
(b) No caso dos micro-serviços houve uma percentagem de 1,68% de erros.

Figura 8.4: Comparação entre percentagem de respostas de erro.

Os códigos de erro recebidos no caso dos micro-serviços foram principalmente erros 403 (Forbidden) e 500 (Internal Server Error), como se pode verificar na Figura 8.5. Já os erros 409 (Conflict) encontram-se presentes em ambos o monólito e os micro-serviços, por isso a sua presença não foi introduzida pela passagem para micro-serviços. Na Figura 8.6 encontra-se, mais pormenorizadamente o número de códigos de erro ao longo do tempo.

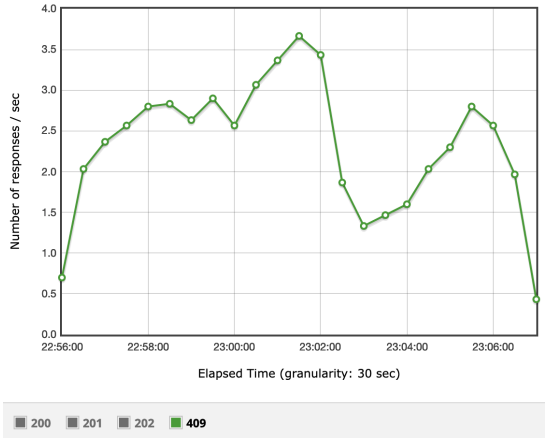


(a) Número de códigos HTTP das respostas ao longo do tempo, no caso do monólito.

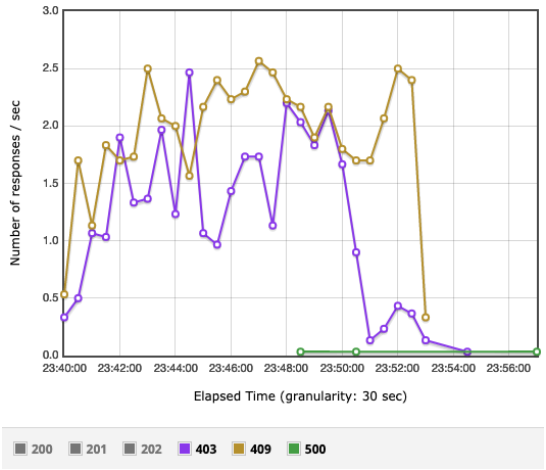


(b) Número de códigos HTTP das respostas ao longo do tempo, no caso dos micro-serviços.

Figura 8.5: Comparação do número de códigos HTTP das respostas.



(a) Número de códigos HTTP de erro ao longo do tempo, no caso do monólito.



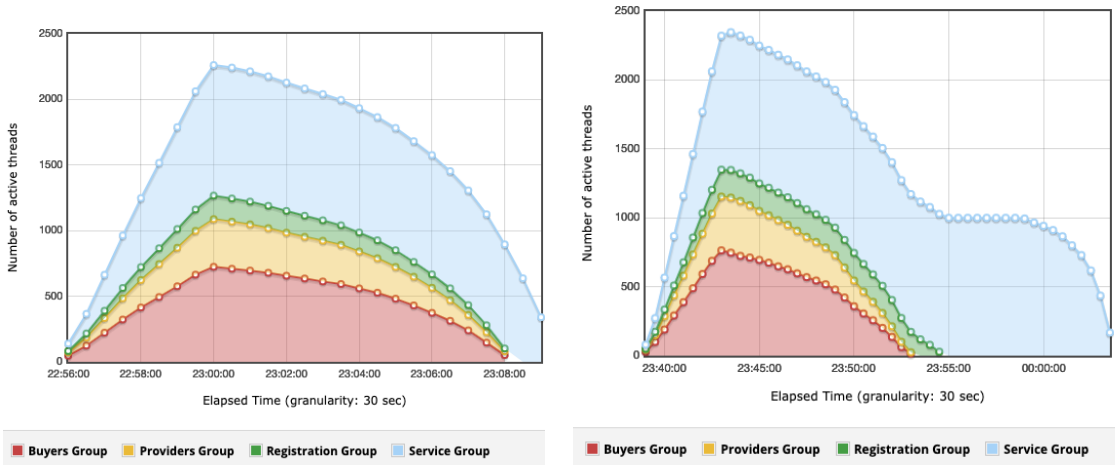
(b) Número de códigos HTTP de erro ao longo do tempo, no caso dos micro-serviços.

Figura 8.6: Número de códigos HTTP de erro ao longo do tempo.

É de notar que estes erros surgem nos momentos de maior concorrência. Na Figura 8.7 encontra-se o número de *threads* ativas ao longo do tempo. É possível verificar que a maioria dos códigos de erro da 8.6 surgem quando o número de *threads* ativas ultrapassa os mil.

Algo notável nos gráficos da Figura 8.7 é o facto de que, no caso do monólito o número de *threads* de cada grupo sobe e desce mais ou menos em simultâneo; enquanto que no caso dos micro-serviços, no momento em que os restantes grupos terminam o seu trabalho, o grupo dos serviços ainda continua mais algum tempo para terminar o seu trabalho. Seria de esperar que o facto de ter um micro-serviço dedicado apenas aos serviços conseguisse despachar essa carga com mais facilidade, no entanto, em ambiente local isso não se verificou.

Em termos do número de pedidos processados por segundo, é possível ver na Figura 8.8 que ambos têm aproximadamente o mesmo comportamento. O seu número sobe rapidamente até atingir um máximo local e baixa gradualmente a partir daí. Perto do final volta a



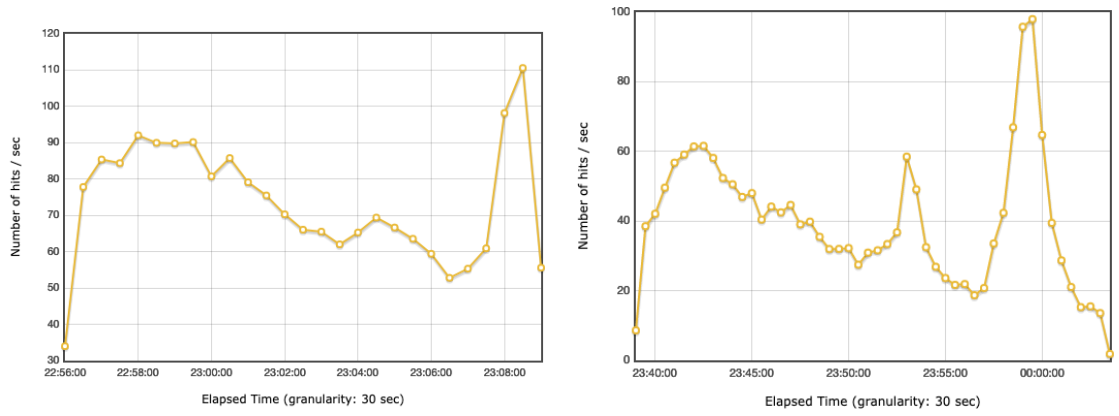
(a) Número de *threads* ativas ao longo do tempo, no caso do monólito.

(b) Número *threads* ao longo do tempo, no caso dos micro-serviços.

Figura 8.7: Comparação do número de *threads* ativas.

haver um pico na altura em que apenas o grupo dos serviços se encontra ativo. Tanto num pico como no outro, o número de pedidos por segundo do monólito é superior ao dos micro-serviços, bem como durante a maior parte dos pontos do gráfico apresentam um maior número de pedidos por segundo. Assim pode ser concluído que, em ambiente local, o monólito apresenta um maior *throughput*.

Isso pode também ser observado analisando o eixo dos *x* dos gráficos da Figura 8.8. Para responder ao mesmo número de chamadas o monólito demorou cerca de 13 minutos, e os micro-serviços cerca de 25 minutos.



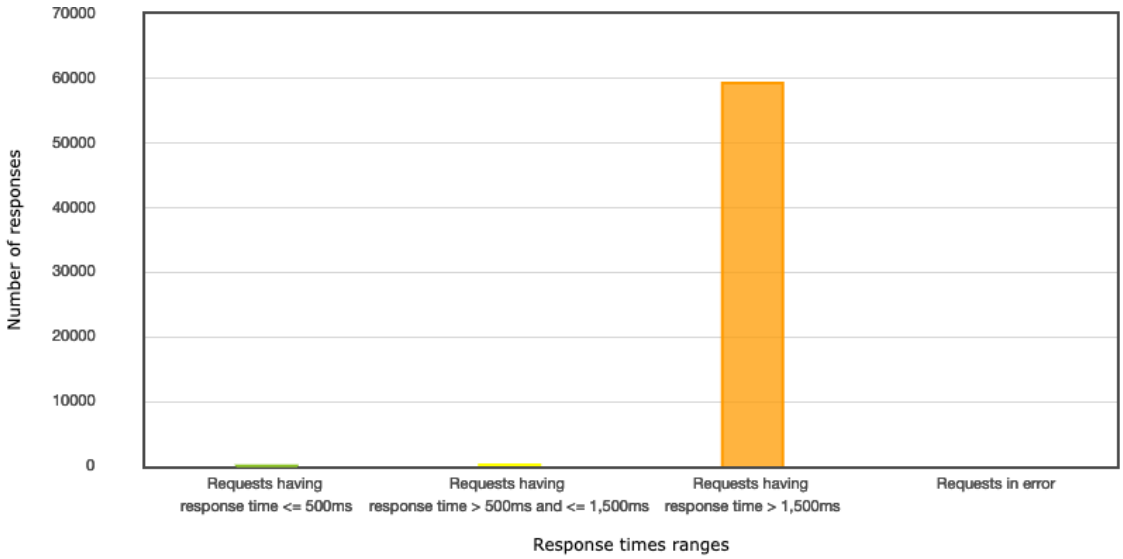
(a) Número de pedidos por segundo ao longo do tempo, no caso do monólito.

(b) Número de pedidos por segundo ao longo do tempo, no caso dos micro-serviços.

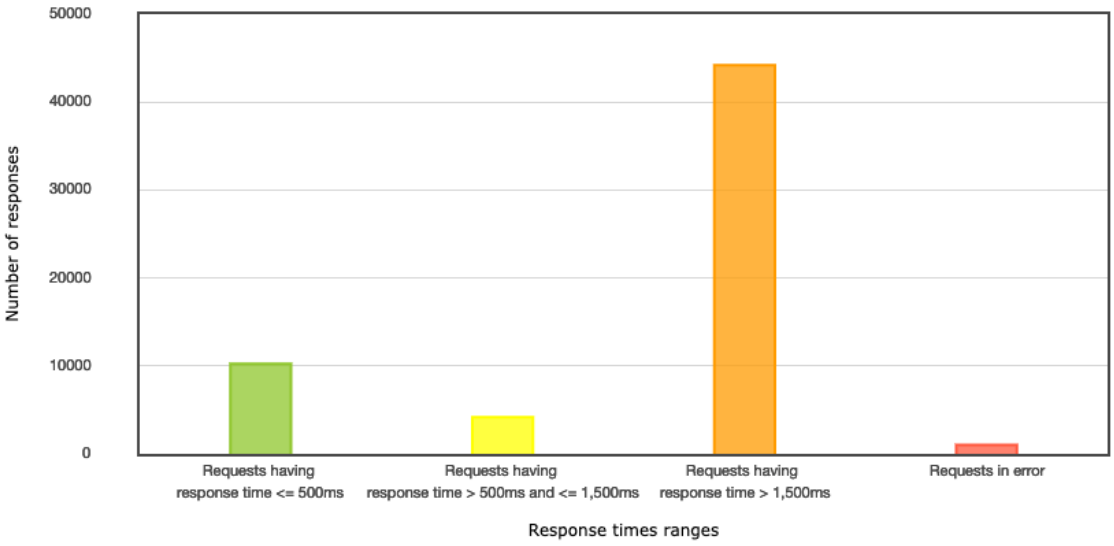
Figura 8.8: Comparação do número de pedidos por segundo ao longo do tempo.

Os tempos de resposta são também significativamente diferentes entre um sistema e outro. Os gráficos da Figura 8.9 mostram que a esmagadora maioria das chamadas do monólito tiveram um tempo de resposta superior a 1500 ms, numerando em quase 60 mil. Por outro lado, no caso dos micro-serviços, foram menos de 50 mil aquelas que tiveram esse tempo de resposta.

Visto isso, o sistema de micro-serviços parece responder mais rapidamente, o que pode ser difícil de reconciliar com o *throughput* mais baixo observado na Figura 8.8. Isto é, até se analisar mais pormenorizadamente os tempos de resposta de cada chamada, cujo gráfico se encontra na Figura 8.10. Aí é possível observar que os tempos de resposta de todas as chamadas do monólito evolui mais ou menos da mesma forma, e atinge um pico máximo de 35 s.

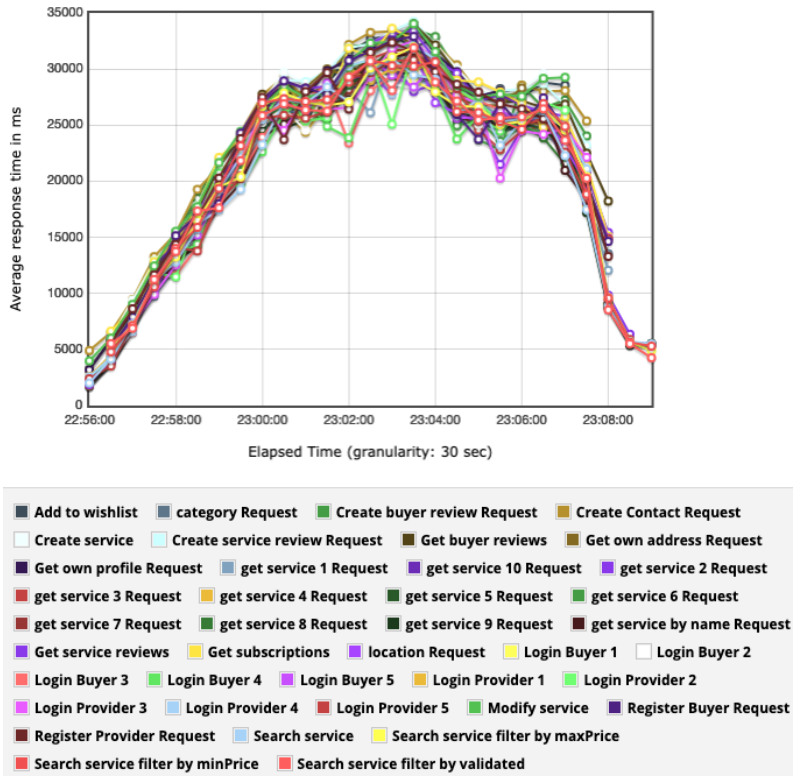


(a) Número de chamadas em cada classe de tempo de resposta, no caso do monólito.

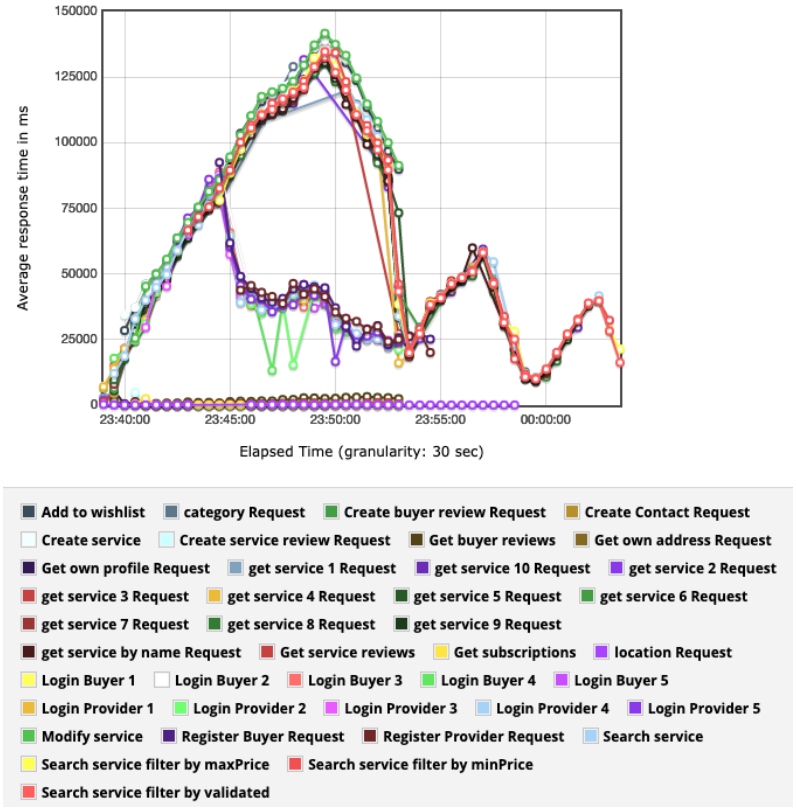


(b) Número de chamadas em cada classe de tempo de resposta, no caso dos micro-serviços.

Figura 8.9: Comparação dos números de chamadas em cada classe de tempo de resposta: menor que 500 ms, entre 500 ms e 1500 ms; e, superiores a 1500 ms. É também apresentado o número de respostas de erro.



(a) Tempos de respostas médios ao longo do tempo para cada tipo de chamada, no caso do monólito.



(b) Tempos de respostas médios ao longo do tempo para cada tipo de chamada, no caso dos micro-serviços.

Figura 8.10: Comparação dos tempos médios de resposta ao longo do tempo.

No caso dos micro-serviços é possível observar três curvas distintas. A curva inferior, com os valores mais próximos do zero, contém as chamadas que contribuem número de chamadas abaixo de 1500 ms observado na Figura 8.9b. Nessa curva estão as chamadas feitas contra os micro-serviços MicroUser, MicroReview, MicroSubscription, MicroContactReuest e MicroPlatformContent. Na curva superior encontram-se as chamadas feitas contra o MicroService, pois esse estava sob uma grande carga, e no seu máximo, os pedidos chegaram a demorar cerca de 140 s a serem respondidos. Na curva intermédia encontram-se os pedidos feitos ao MicroAuth, que também estava sob carga do grupo de registo. Então, o número de chamadas com tempos de resposta superiores a 1500 ms pode ser menor para os micro-serviços, no entanto, as respostas mais demoradas são muito mais demoradas que as do monólito, e daí surge o menor *throughput* dos micro-serviços.

Assim é possível verificar que, dada a sua natureza, o sistema de micro-serviços permite que as suas partes sob menos carga continuem a funcionar normalmente com bons tempos de resposta. Numa situação de uma carga assimétrica como esta sobre um ambiente de produção, bastaria escalar os micro-serviços afetados, enquanto que o monólito teria que ser escalado como um todo.

Em conclusão, no ambiente local obteve-se um sistema de micro-serviços que é mais dado a erros e tem menos *throughput* que o monólito. Mais realisticamente, para tirar partido dos micro-serviços pretende-se espalhar a carga por várias máquinas, e não tê-los todos a correr na mesma. Na próxima secção é feito esse estudo.

8.4.2 Na nuvem

Por forma de obter um teste um pouco mais realístico, em que cada micro-serviço está implantado na sua própria máquina, optou-se por fazer uma implantação na nuvem. O monólito e os micro-serviços encontram-se implantados em Heroku, cada um deles com acesso a recursos similares. A base de dados encontra-se alojada na Azure, e a *gateway* encontra-se implantada localmente em Docker.

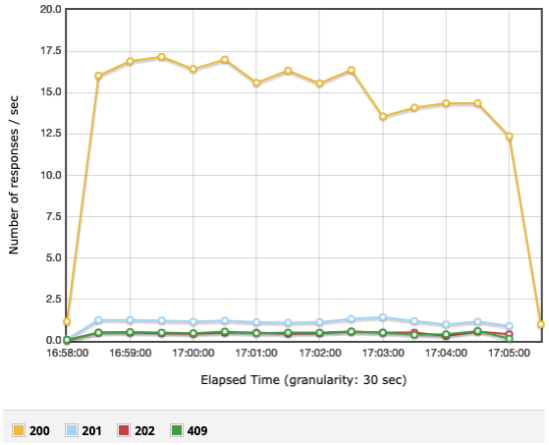
Como os recursos disponibilizados pelo Heroku no nível grátis são baixos [105], os testes foram lançados com menos carga. O grupo dos compradores foi utilizado com 75 *threads* concorrentes, o dos fornecedoras com 25, o dos registos com 40 e o dos serviços com 100. O tempo de *ramp up* foi mantido em 240 segundos e a repetição da execução também.

O monólito continua com 100% de sucesso, e o sistema de micro-serviços continua com uma pequena percentagem de erros, como se encontra na Figura 8.11.

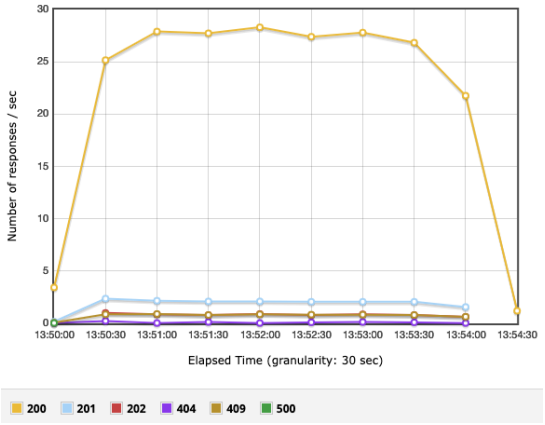
(a) No caso do monólito os pedidos tiveram 100% de sucesso.

(b) No caso dos micro-serviços houve uma percentagem de 0,35% de erros.

Figura 8.11: Comparação entre percentagem de respostas de erro.

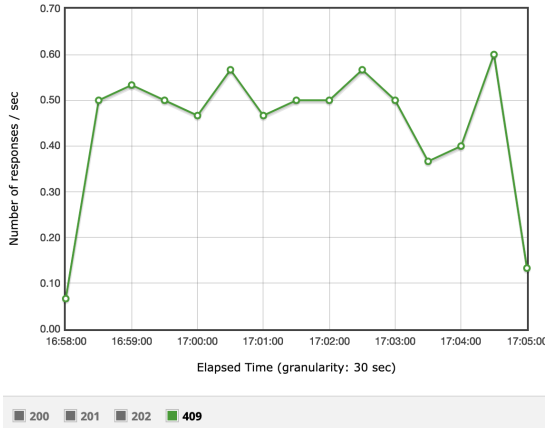


(a) Número de códigos HTTP das respostas ao longo do tempo, no caso do monólito.

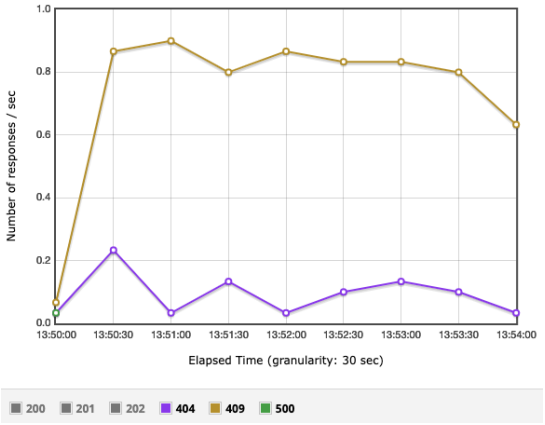


(b) Número de códigos HTTP das respostas ao longo do tempo, no caso dos micro-serviços.

Figura 8.12: Comparação do número de códigos HTTP das respostas.



(a) Número de códigos HTTP de erro ao longo do tempo, no caso do monólito.



(b) Número de códigos HTTP de erro ao longo do tempo, no caso dos micro-serviços.

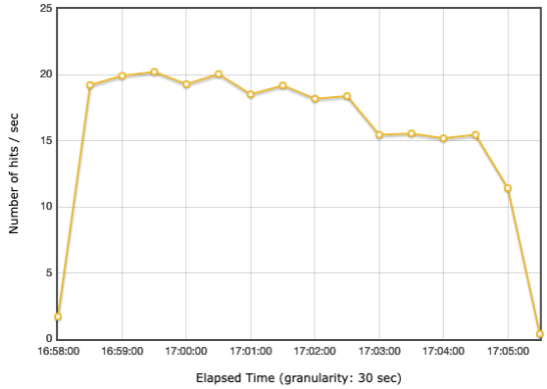
Figura 8.13: Número de códigos HTTP de erro ao longo do tempo.

Os erros 409 continuam presentes em ambos, no entanto os restantes erros diminuíram em número, tal como se encontra nas Figuras 8.12 e 8.13.

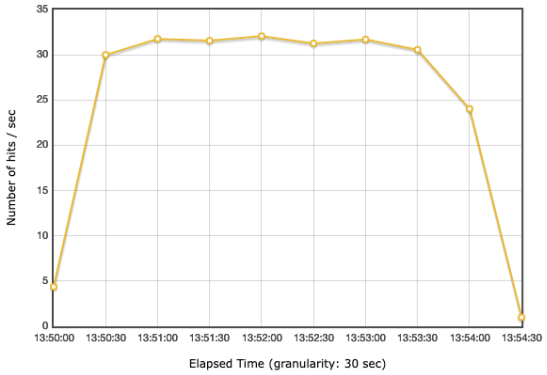
O número de chamadas por segundo manteve-se mais estável ao longo do teste, não apresentando picos visíveis como tinha no caso do ambiente local.

Os micro-serviços atingem valores de chamadas por segundo mais elevados, encontrando-se entre os 30 e os 35, enquanto que o monólito apenas consegue processar entre 15 e 20 pedidos por segundo, tal como se encontra na Figura 8.14. O monólito demorou cerca de sete minutos e meio a responder a todos os pedidos, enquanto que os micro-serviços demoraram cerca de quatro e meio. Pode concluir-se que os micro-serviços demonstram um maior *throughput*.

Esse *throughput* é sustentado por tempos de respostas mais curtos por parte dos micro-serviços. Na Figura 8.15 verifica-se que a maior parte das chamadas ao monólito têm um tempo de resposta superior a 1500 ms, enquanto que as dos micro-serviços têm tempos de resposta inferiores.

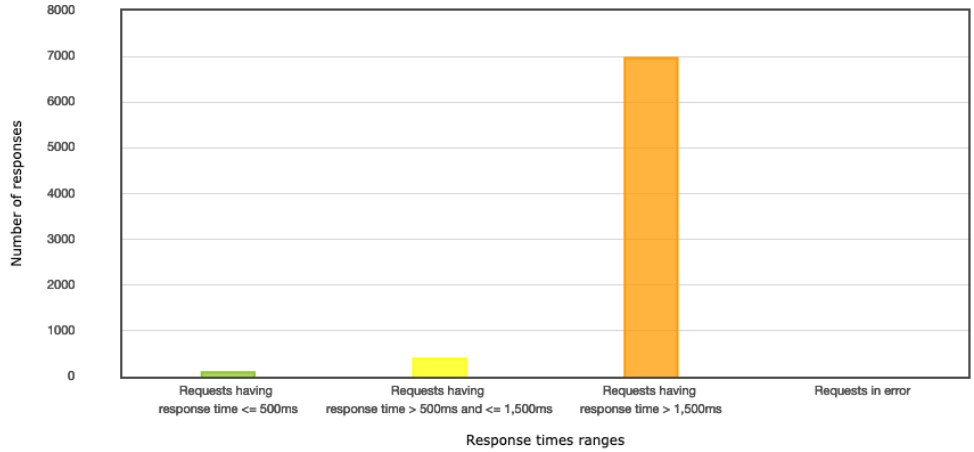


(a) Número de *threads* ativas ao longo do tempo, no caso do monólito.

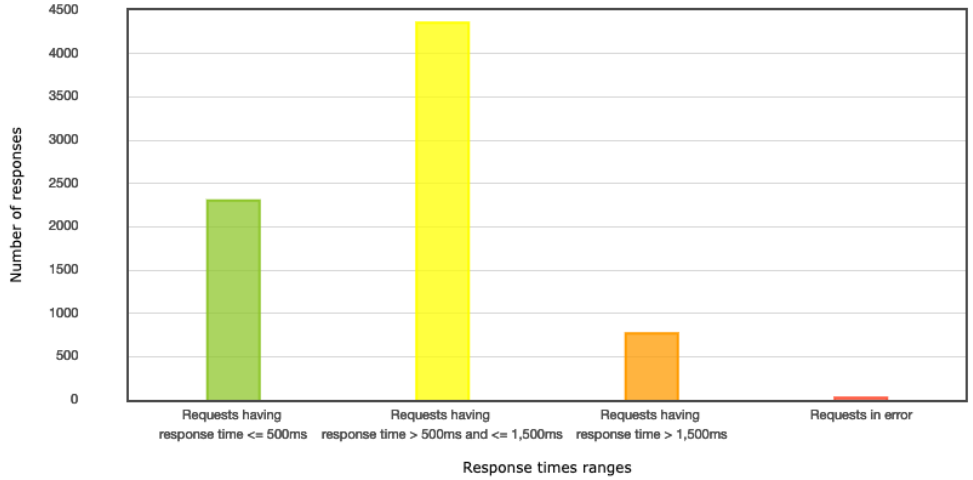


(b) Número *threads* ao longo do tempo, no caso dos micro-serviços.

Figura 8.14: Comparação do número de *threads* ativas.



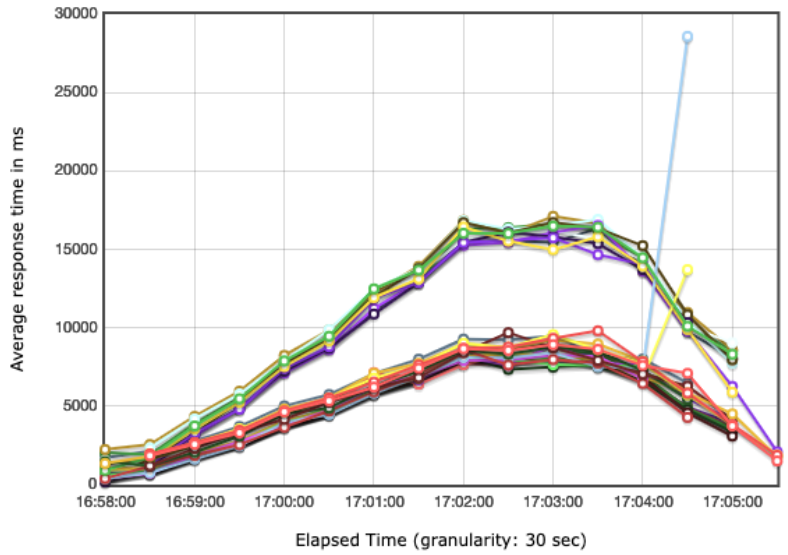
(a) Número de chamadas em cada classe de tempo de resposta, no caso do monólito.



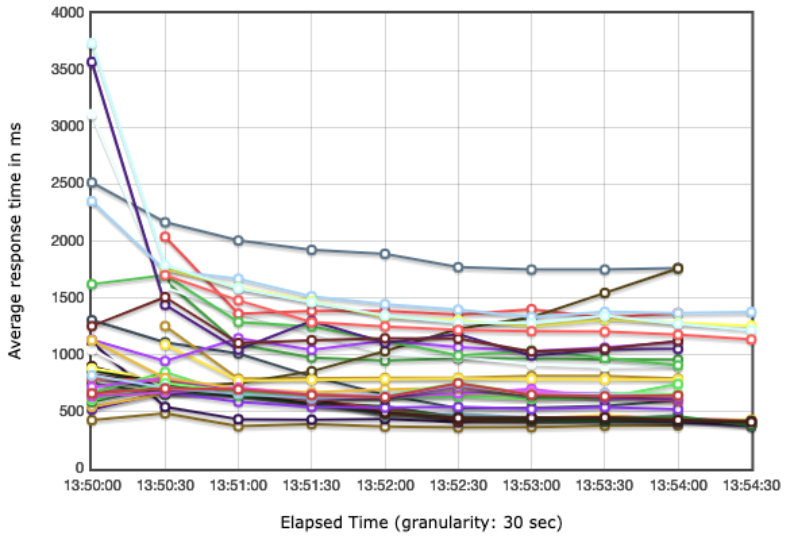
(b) Número de chamadas em cada classe de tempo de resposta, no caso dos micro-serviços.

Figura 8.15: Comparação dos números de chamadas em cada classe de tempo de resposta: menor que 500 ms, entre 500 ms e 1500 ms; e, superiores a 1500 ms. É também apresentado o número de respostas de erro.

O tempo de resposta de cada tipo de pedido, que se encontra na Figura 8.16, mostra agora um comportamento bastante diferente. No caso do monólito, tal como em ambiente local, o tempo de resposta vai aumentando com o aumento de carga. No caso dos micro-serviços, o tempo de resposta até começa elevado, mas rapidamente diminui, e a tendência de diminuição mantém-se até ao fim do teste. Esta quantidade de carga, que estava a ter um impacto real na latência do monólito, deixou o sistema de micro-serviços a continuar a funcionar normalmente.



(a) Tempos de respostas médios ao longo do tempo para cada tipo de chamada, no caso do monólito.



(b) Tempos de respostas médios ao longo do tempo para cada tipo de chamada, no caso dos micro-serviços.

Figura 8.16: Comparação dos tempos médios de resposta ao longo do tempo.

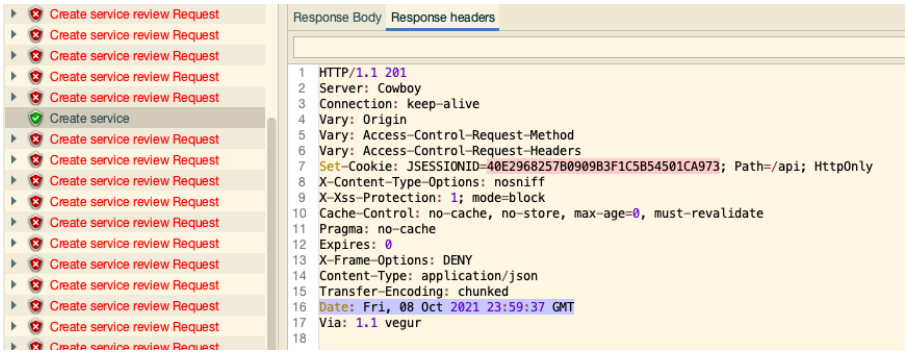
Assim, é possível compreender-se um pouco melhor a natureza destes sistemas. Por virtude de simplesmente ter mais máquinas (por conseguinte, mais recursos) para disponibilizar o sistema, ele consegue melhor lidar com a carga e manter um *throughput* elevado e latência baixa. Assim o monólito precisa de ser escalado mais cedo à medida que a carga aumenta.

8.5 Impacto dos métodos de comunicação assíncronos

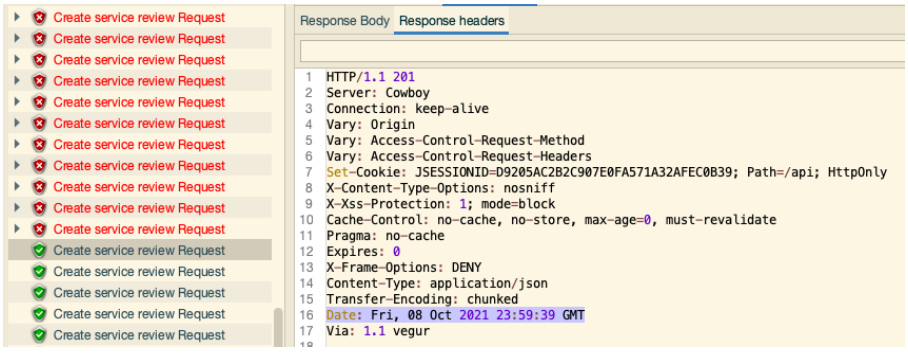
Os métodos de comunicação assíncronos introduzem atrasos na propagação da informação pelo sistema. Para tentar medir esse atraso foi elaborado um plano de testes em JMeter que visa criar um serviço e continuamente tentar criar uma revisão sobre ele. A informação sobre o serviço a ser criado viaja entre o MicroService e o MicroReview usando Kafka, por isso será de esperar que o MicroReview não conheça a existência do serviço durante algum tempo após a sua criação.

Este caso de uso em particular não é relevante para a experiência do utilizador, pois um serviço não estará a receber revisões imediatamente depois de ser criado. Apesar disso, é um caso de estudo interessante para analisar a propagação de informação usando Kafka.

Na Figura 8.18 é possível ver que o plano de testes está continuamente a tentar criar a revisão. Mesmo depois do serviço ser criado, os pedidos de criação da revisão continuam a falhar durante algum tempo com o estado 404 (Not Found), pois o MicroReview ainda não recebeu o serviço. É de notar que apesar de ter vários pedidos falhados entre a criação do serviço, e a primeira criação da revisão com sucesso, a diferença de tempo entre estes dois pedidos é de apenas dois segundos.



(a) Chamada de criação do serviço.

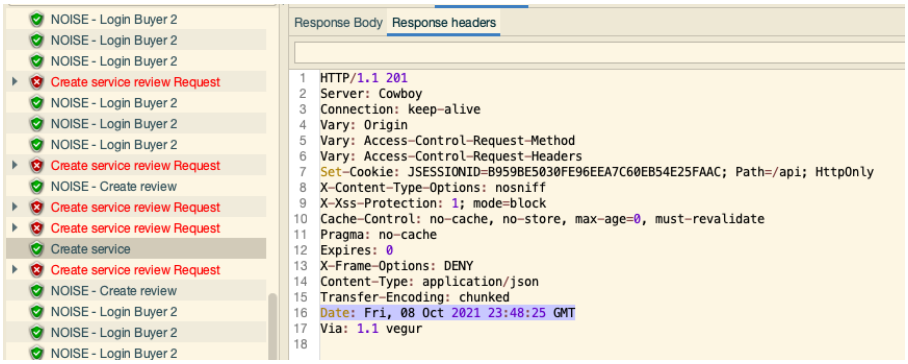


(b) Primeira chamada com sucesso de criação da revisão.

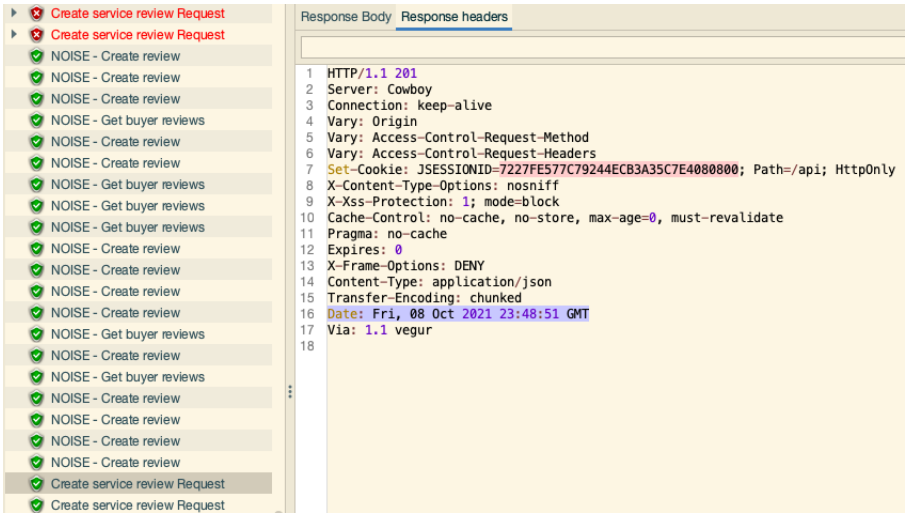
Figura 8.17: Análise do atraso na propagação da informação por vias de comunicação assíncronas, entre o MicroService e o MicroReview.

O atraso de dois segundos surge numa altura em que o MicroReview não está sob carga. Para testar o seu comportamento sob carga foi criado outro grupo de 80 threads que continuamente criam revisões sobre outros serviços e consultam revisões. O resultado desse teste encontra-se na Figura 8.18. O intervalo de tempo entre a criação do serviço e a

primeira criação de revisão com sucesso é agora de 26 segundos. Este intervalo é muito maior que aquele que foi verificado anteriormente.



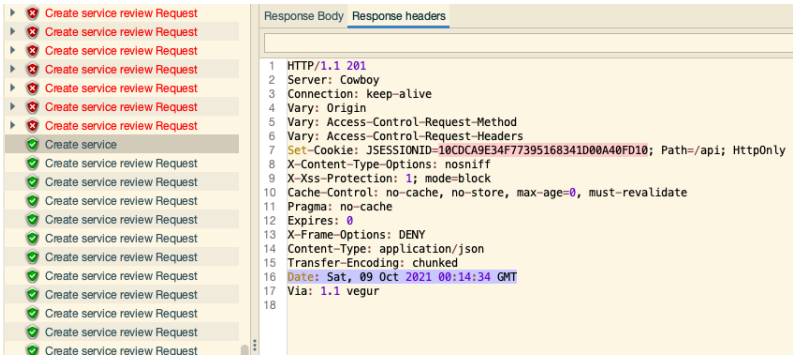
(a) Chamada de criação do serviço.



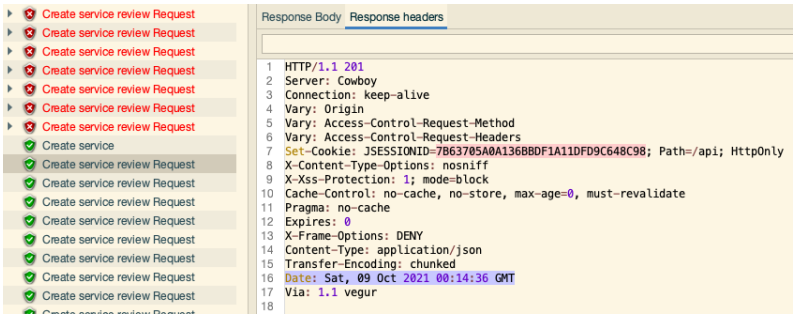
(b) Primeira chamada com sucesso de criação da revisão.

Figura 8.18: Análise do atraso na propagação da informação por vias de comunicação assíncronas, entre o MicroService e o MicroReview, com mais carga sobre o MicroReview. Chamadas com "NOISE" no nome são aquelas que foram introduzidas para simular essa carga.

Os mesmos testes foram também lançados sobre o monólito, e, em ambos os casos (com e sem carga) o primeiro pedido para criar a revisão feito depois da criação do serviço foi imediatamente aceite, tal como pode ser verificado nas Figuras 8.19 e 8.20.

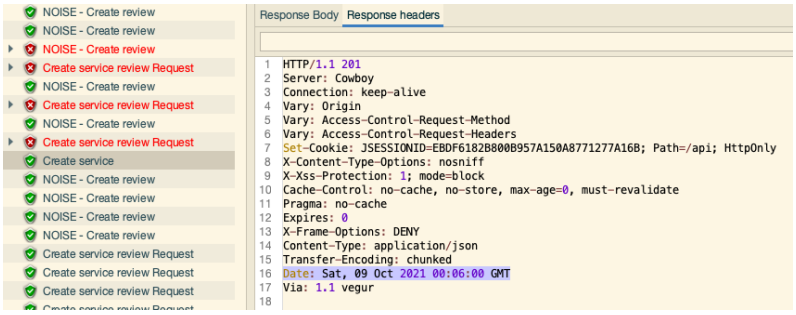


(a) Chamada de criação do serviço.

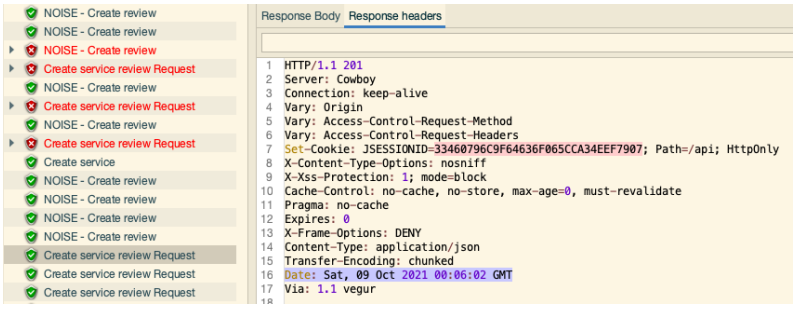


(b) Primeira chamada com sucesso de criação da revisão.

Figura 8.19: Análise do atraso entre a criação de um serviço e criação da revisão, para o caso do monólito.



(a) Chamada de criação do serviço.



(b) Primeira chamada com sucesso de criação da revisão.

Figura 8.20: Análise do atraso entre a criação de um serviço e criação da revisão, para o caso do monólito, com mais carga. Chamadas com "NOISE" no nome são aquelas que foram introduzidas para simular a carga.

Um intervalo de quase trinta segundos entre uma ação do utilizador e essa ação ser visível pode ter um impacto real na experiência do utilizador, e levar a um certo grau de frustração e incerteza se o sistema está realmente a funcionar. Para evitar isso é necessário alterar a forma como esta informação é mostrada ao utilizador de forma a tornar claro que o sistema está a processar o pedido e ficará disponível em breve.

8.6 Análise global

Foi verificado que o sistema resultante é bastante maior e mais complexo que o inicial. Isso deve-se à repetição de algum código (como por exemplo o envio de emails, e alguns DTOs), e também à introdução de algumas ferramentas novas, como o Kafka. Foram introduzidos alguns defeitos que levam ao aparecimento de códigos de erro nas respostas do sistema, quando este se encontra sob carga. Para determinar a sua causa é necessário adicionar meios de monitorização ao sistema.

O comportamento sob carga dos dois sistemas é também diferente, sendo que o sistema de micro-serviços tem a capacidade de processar mais carga, tendo um *throughput* superior. Também é notável que quando a carga não está espalhada simetricamente sobre o sistema, os micro-serviços com menos carga continuam a conseguir funcionar normalmente, e apenas os que estão sob mais carga demonstram um maior tempo de resposta.

O comportamento sob carga dos dois sistemas é também diferente, sendo que o sistema de micro-serviços tem a capacidade de processar mais carga, tendo um *throughput* superior. Também é notável que quando a carga não está espalhada simetricamente sobre o sistema, os micro-serviços com menos carga continuam a conseguir funcionar normalmente, e apenas os que estão sob mais carga demonstram um maior tempo de resposta.

Tem-se assim um sistema que responde melhor às necessidades de escalabilidade do cliente, com o custo de ter um sistema mais extenso e mais complexo para se manter e implantar.

Em termos de decidir qual destes dois sistemas é realmente o melhor para o cliente, isso dependerá sempre apenas do cliente. Neste caso, do quão forte será a necessidade dessa escalabilidade adicional, e do quão extensivo será o orçamento do cliente para desenvolver e manter um sistema que custará certamente mais.

Capítulo 9

Conclusão

Neste capítulo é resumido aquilo que foi feito no trabalho, os objetivos que foram atingidos, e, o trabalho que ainda falta fazer no sistema de micro-serviços.

9.1 Trabalho realizado

Neste trabalho foi feita uma revisão da literatura sobre micro-serviços. Foram estudadas as suas propriedades, as boas práticas a adotar para um sistema com sucesso, e as armadilhas a evitar.

Foi analisada a arquitetura do monólito para o dividir em micro-serviços, e foram propostos dois *designs* diferentes para essa divisão. Um deles foi adotado e implementado. Neste caso de estudo em particular, o trabalho envolvido nessa implementação foi simplificado pelo facto da estrutura interna do monólito ser já parecida com a estrutura dos micro-serviços, o que não será o caso da maioria dos monólitos.

A comunicação entre micro-serviços foi assegurada por *event sourcing* para informar os restantes micro-serviços do acontecimento de certos eventos; por composição de APIs para obter informação de outros micro-serviços no imediato; e também usando uma Saga, para concluir uma tarefa que envolve mais que um micro-serviço.

O sistema de micro-serviços foi estudado e comparado com o monólito principalmente pela sua complexidade e comportamento sob carga.

9.2 Objetivos atingidos

Os objetivos propostos foram atingidos estudando a transformação do monólito em micro-serviços.

O aumento de tamanho e complexidade foram estudados analisando o número de linhas de código, a complexidade ciclomática das classes java e a dívida técnica do monólito e dos micro-serviços. Foi verificado que houve um aumento significativo de todos esses parâmetros.

O ganho de performance foi estudado utilizando testes de carga. Foi verificado que, quando os dois sistemas estão implantados na nuvem, em maquinaria semelhante, os micro-serviços atingem um *throughput* superior. Isso deve-se principalmente ao facto que os micro-serviços estão espalhados por várias máquinas, enquanto que o monólito está em apenas uma.

O impacto dos métodos de comunicação assíncronos foi testado enviando uma atualização para um micro-serviço e esperando que essa atualização apareça em outro. Quando o segundo micro-serviço se encontra sobre uma carga significativa, o atraso verificado é notável, o que demonstra a necessidade de adotar práticas que levem estes atrasos em consideração.

9.3 Trabalho futuro

A implementação do sistema de micro-serviços encontra-se ainda incompleta, faltando implementar o Backoffice e o ActionService.

É também ainda necessário implementar certas boas práticas importantes em micro-serviços, tais como: monitorização, CI/CD, segurança na comunicação entre serviços, e diferentes ambientes (como por exemplo um ambiente para correr os testes end-to-end).

Há também algumas melhorias que poderiam ser aplicadas, tais como a implementação de OAuth 2.0 no MicroAuth, de uma *gateway* de SSO, mitigação de anomalias na saga e CQRS no MicroService.

Bibliografia

[1] *History of Ecommerce*. https://www.ecommerce-land.com/history_ecommerce.html. Acedido em 11 de Fev. de 2021.

[2] Jan Betzing et al. «Exploring customers' likeliness to use e-service touchpoints in brick and mortar retail». Em: *Electronic Markets* (nov. de 2020). doi: 10.1007/s12525-020-00445-0.

[3] Chris Richardson. *Introduction to Microservices*. https://www.nginx.com/blog/introduction-to-microservices/. Acedido em 10 de Fev. de 2021. Mai. de 2015.

[4] Moaid Hathot. *To Microservices or not to Microservices?* https://codevalue.com/to-microservices-or-not-to-microservices/. Acedido em 11 de Fev de 2021.

[5] Sandeep Jagtap. *Microservices Powered By Domain-Driven Design*. https://dzone.com/articles/microservices-powered-by-domain-driven-design. Acedido em 12 de Fev. de 2021. Nov. de 2020.

[6] Martin Fowler. *Event Sourcing*. https://martinfowler.com/eaaDev/EventSourcing.html. Acedido em 4 de Fev. de 2021. Dez. de 2005.

[7] *CAP Theorem*. https://www.ibm.com/cloud/learn/cap-theorem. Acedido em 30 de Jan. de 2021.

[8] *Action research*. https://en.wikipedia.org/wiki/Action_research. Acedido em 16 de Set. de 2021.

[9] *What is Ecommerce?* https://www.shopify.com/encyclopedia/what-is-ecommerce. Acedido em 4 de Fev. de 2021.

[10] *Software Framework vs Library*. https://www.geeksforgeeks.org/software-framework-vs-library/. Acedido em 23 de Fev. de 2021. Set. de 2020.

[11] *How useful are web application frameworks? And How do I know which framework would suit me?* https://www.cuelogic.com/blog/how-useful-are-web-application-frameworks-how-do-i-know-which-framework-would-suit-me. Acedido em 23 de Fev. de 2021.

[12] *Specifying Components as Properties*. https://docs.oracle.com/cd/E69533_01/Platform.11-3/ATGPlatformProgGuide/html/s0204specifyingcomponentsasproperties01.html. Acedido em 23 de Fev. de 2021.

[13] *Oracle Commerce Platform Documentation*. https://www.oracle.com/technical-resources/documentation/atg-web-commerce-doc.html. Acedido a 23 de Fev. de 2021.

[14] Curtis Johnson. *Java Basics: What Is SAP Hybris?* https://www.jrebel.com/blog/sap-hybris. Acedido em 23 de Feb. de 2021. Ago. de 2020.

[15] Steve Susina. *Demandware Ecommerce Solution*. https://www.lyonscg.com/2013/05/24/demandware-partner/. Acedido em 23 de Feb. de 2021. Mai. de 2013.

[16] *HCL Commerce*. https://www.hcltechsw.com/wps/portal/products/commerce/home. Acedido em 23 de Fev. de 2021.

[17] *Magento Commerce*. https://magento.com/. Acedido em 23 de Fev. de 2021.

[18] *What is SaaS?* https://www.salesforce.com/eu/learning-centre/tech/saas/. Acedido em 23 de Fev. de 2021.
[19] *Oracle CX Commerce 21A*. https://docs.oracle.com/en/cloud/saas/cx-commerce/21a/index.html. Acedido em 23 de Fev. de 2021.
[20] *Salesforce Commerce Cloud*. https://www.salesforce.com/eu/products/commerce-cloud/overview/. Acedido em 23 de Fev. de 2021.
[21] *SAP Commerce Cloud*. https://www.sap.com/portugal/products/commerce-cloud/technical-information.html. Acedido em 23 de Fev. de 2021.
[22] *Our Microservices architecture*. https://commercetools.com/microservices. Acedido em 23 de Fev. de 2021.
[23] *commercetools*. https://commercetools.com/commerce-platform. Acedido em 23 de Fev. de 2021.
[24] *Innovative Commerce, Without the Risk*. https://www.elasticpath.com/product. Acedido em 23 de Fev. de 2021.
[25] *10 Best Ecommerce Platforms Compared and Rated For 2021*. https://www.ecommerceceo.com/ecommerce-platforms. Acedido em 23 de Fev. de 2021.
[26] *Shopify*. https://www.shopify.com/. Acedido em 23 de Fev. de 2021.
[27] *Squarespace eCommerce website builder*. https://www.squarespace.com/ecommerce-website. Acedido em 23 de Fev. de 2021.
[28] *Wix eCommerce*. https://www.wix.com/ecommerce/website. Acedido em 23 de Fev. de 2021.
[29] *BigCommerce Headless Commerce for Greater Flexibility*. https://www.bigcommerce.com/solutions/headless-commerce/. Acedido em 23 de Fev. de 2021.
[30] *Shift4shop Best e-Commerce platform of 2021*. https://www.shift4shop.com/. Acedido em 23 de Fev. de 2021.
[31] *Magento – Hybris – Oracle ATG: What to Choose?* https://neklo.com/magento-hybris-oracle-atg-what-to-choose/. Acedido em 29 de Jan. de 2021.
[32] *Magento vs. Demandware*. https://forixcommerce.com/magento-vs-demandware/. Acedido em 29 de Jan. de 2021. Mar. de 2018.
[33] Shaneil Lafayette. *eCommerce Platform TCO Comparison*. https://www.elasticpath.com/blog/ecommerce-platform-tco-comparison. Acedido em 28 de Jan. de 2021. Jan. de 2021.
[34] *Oracle Commerce Cloud vs. Demandware vs. Magento*. https://www.realdecoy.com/2016/07/12/oracle-commerce-cloud-vs-demandware-vs-magento-comparison-table/. Acedido em 29 de Jan. de 2021. Jul. de 2016.
[35] Yuliana H. *Enterprise Ecommerce Platform Comparison: How to Select Between Magento, Salesforce, Hybris, Shopify, BigCommerce, and Oracle*. https://elogic.co/blog/enterprise-ecommerce-platform-comparison-how-to-select-between-magento-salesforce-hybris-and-shopify/. Acedido em 29 de Jan. de 2021.
[36] *Wix vs Squarespace vs Shopify — Which is The Best Home For Your Website?* https://litextension.medium.com/wix-vs-squarespace-vs-shopify-which-is-the-best-home-for-your-website-701aa3d71083. Acedido em 29 de Jan. de 2021. Jul. de 2020.
[37] *The Perfect Fit for Every Shop*. https://www.shift4shop.com/plans.html?country=notUS. Acedido em 23 de Fev. de 2021.
[38] Sam Newman. *Building Microservices*. 1st. O'Reilly Media, Inc., 2015. isbn: 1491950358.
[39] Kevlin Henney. *97 Things Every Programmer Should Know: Collective Wisdom from the Experts*. 1st. O'Reilly Media, Inc., 2010. isbn: 0596809484.

[40] *SOA (Service-Oriented Architecture)*. https://www.ibm.com/cloud/learn/soa. Acedido em 28 de Jan. de 2021. Jul. de 2019.

[41] Ima Miri. *Microservices vs. SOA*. https://dzone.com/articles/microservices-vs-soa-2. Acedido em 23 de Jan. de 2021. Jan. de 2017.

[42] *SOA vs. Microservices: What's the Difference?* https://www.ibm.com/cloud/blog/soa-vs-microservices. Acedido em 28 de Jan. de 2021. Set. de 2020.

[43] *The Value of Correlation IDs*. https://blog.rapid7.com/2016/12/23/the-value-of-correlation-ids/. Acedido em 30 de Jan. de 2021. Dez. de 2016.

[44] Chris Richardson. *Pattern: Database per service*. https://microservices.io/patterns/data/database-per-service.html. Acedido em 2 de Fev. de 2021.

[45] C. Richardson. *Microservices Patterns: With examples in Java*. Manning Publications, 2018. isbn: 9781617294549. url: https://books.google.pt/books?id=UeK1swEACAAJ.

[46] Martin Fowler. *CommandQuerySeparation*. https://martinfowler.com/bliki/CommandQuerySeparation.html. Acedido em 2/2/2021. Dez. de 2005.

[47] Martin Fowler. *CQRS*. https://martinfowler.com/bliki/CQRS.html. Acedido em 2/2/2021. Jul. de 2011.

[48] Hugo Rocha. *What they don't tell you about event sourcing*. https://medium.com/@hugo.oliveira.rocha/what-they-dont-tell-you-about-event-sourcing-6afc23c69e9a. Acedidio em 4 de Fev. de 2021. Ago. de 2018.

[49] Evans. *Domain-Driven Design: Tacking Complexity In the Heart of Software*. USA: Addison-Wesley Longman Publishing Co., Inc., 2003. isbn: 0321125215.

[50] Hugo Rocha. *The Perils of Event-Driven: Eventual Consistency*. https://medium.com/swlh/handling-eventual-consistency-11324324aec4. Acedido em 4 de Fev. de 2021. Jun. de 2020.

[51] Dilfuruz Kizilpinar. *Data Consistency in Microservices Architecture*. https://medium.com/garantibbva-teknoloji/data-consistency-in-microservices-architecture-5c67e0f65256. Acedido em 21 de Jul. de 2021. Abr. de 2021.

[52] Isaac Eldridge. *What Is Container Orchestration?* https://newrelic.com/blog/best-practices/container-orchestration-explained. Acedido em 11 de Jun. de 2021. Jul. de 2018.

[53] Martin Fowler. *Microservice Trade-Offs*. https://martinfowler.com/articles/microservice-trade-offs.html. Acedido em 7 de Fev. de 2021. Jul. de 2015.

[54] Ran Ilany. *Microservices vs. Monoliths: Which is Right for Your Enterprise?* https://devops.com/microservices-vs-monoliths-which-is-right-for-your-enterprise/. Acedido em 7 de Fev. de 2021. Jun. de 2020.

[55] Phil Wittmer. *Microservices Disadvantages and Advantages*. https://www.tiempodev.com/blog/disadvantages-of-a-microservices-architecture/. Acedido em 7 de Fev. de 2021. Dez. de 2019.

[56] Bartosz Jedrzejewski. *Microservices and cross cutting concerns*. https://www.e4developer.com/2018/09/09/microservices-and-cross-cutting-concerns/. Acedido em 7 de Fev. de 2021. Set. de 2018.

[57] Dmitry Gulin Hari Ohm Prasath Rajagopal Tabby Ward. *AWS Prescriptive Guidance: Decomposing monoliths into microservices*. Rel. téc. Amazon Web Services, Inc. an, 2021.

[58] João Carlos Ribeiro Dias Neves. «Technical Challenges of Microservices Migration». Tese de mestrado. Instituto Superior de Engenharia do Porto, 2019.

[59] Meng Wang. «Service Integration Design Patterns in Microservices». Tese de mestrado. South Dakota State University, 2018.

[60] Tiago Costa Santos. «Adopting Microservices». Tese de mestrado. Instituto Superior Técnico, 2018.

[61] Tony Woodall. «Conceptualising 'value for the customer': an attributional, structural and dispositional analysis». Em: *Academy of marketing science review* 12.1 (2003), pp. 1–42.

[62] *3 ways of thinking about client value*. `https://legal.thomsonreuters.com/blog/3-ways-of-thinking-about-client-value/`. Acedido em 20 de Jan. de 2021. Dez. de 2019.

[63] Valarie A Zeithaml. «Consumer perceptions of price, quality, and value: a means-end model and synthesis of evidence». Em: *Journal of marketing* 52.3 (1988), pp. 2–22.

[64] Jozee Lapierre. «Customer-perceived value in industrial contexts». Em: *Journal of Business and Industrial Marketing* 15.2/3 (2000), pp. 122–145.

[65] *Porters value chain*. `https://www.mindtools.com/pages/article/newSTR_66.htm`. Acedido em 21 de Jan. de 2021.

[66] Monire Jalili e Kamran Rezaie. «Quality principles deployment to achieve strategic results». Em: *International Journal of Business Excellence* 3.2 (2010), pp. 226–259.

[67] *Creating a Value Proposition*. `https://www.mindtools.com/CommSkll/ValueProposition.htm`. Acedido em 21 de Jan. de 2021.

[68] Thomas L. Saaty. «Decision making with the analytic hierarchy process». Em: *International Journal of Services Sciences* 1.1 (2008).

[69] Technology stack. `https://devdocs.magento.com/guides/v2.4/architecture/tech-stack.html`. Acedido em 14 de Fev. de 2021. Out. de 2020.

[70] Thomas L. Saaty e Liem T. Tran. «On the invalidity of fuzzifying numerical judgments in the Analytic Hierarchy Process». Em: *Mathematical and Computer Modelling* 46.7 (2007). Decision Making with the Analytic Hierarchy Process and the Analytic Network Process, pp. 962–975. issn: 0895-7177.

[71] *Which OAuth 2.0 Flow Should I Use?* `https://auth0.com/docs/authorization/which-oauth-2-0-flow-should-i-use`. Acedido em 18 de Fev. de 2021.

[72] Vadim Samokhin. *DDD Strategic Patterns: How To Define Bounded Contexts*. `https://codeburst.io/ddd-strategic-patterns-how-to-define-bounded-contexts-2dc70927976e`. Acedido em 23 de Jun. de 2021. Out. de 2017.

[73] *What is scrum?* `https://www.scrum.org/resources/what-is-scrum`. Acedido em 20 de Set. de 2021.

[74] *Gitflow Workflow*. `https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`. Acedido em 20 de Set. de 2021.

[75] *Save time and effort with higher Code Quality*. `https://sonarcloud.io/code-quality`. Acedido em 20 de Set. de 2021.

[76] *What is Continuous Integration?* `https://www.atlassian.com/continuous-delivery/continuous-integration`. Acedido em 20 de Set. de 2021.

[77] *Bitbucket Pipelines and Deployments*. `https://bitbucket.org/product/features/pipelines`. Acedido em 20 de Set. de 2021.

[78] *Accelerate developer productivity*. `https://gradle.org/`. Acedido em 20 de Set. de 2021.

[79] *Docker: Empowering App Development for Developers*. `https://www.docker.com/`. Acedido em 20 de Set. de 2021.

[80] *Track, version, and deploy database changes*. `https://www.liquibase.org/`. Acedido em 20 de Set. de 2021.

[81] *App platform*. `https://www.digitalocean.com/products/app-platform/`. Acedido em 20 de Set. de 2021.

[82] *IMAP, POP, and SMTP*. https://developers.google.com/gmail/imap/imap-smtp. Acedido em 20 de Set. de 2021.
[83] *Build Modern Commerce with PayPal*. https://developer.paypal.com/home. Acedido em 20 de Set. de 2021.
[84] *A complete API suite to build amazing payments experiences*. https://paystack.com/developers. Acedido em 20 de Set. de 2021.
[85] *Geocoding API*. https://developers.google.com/maps/documentation/geocoding/overview. Acedido em 20 de Set. de 2021.
[86] *checkout-sdk 1.0.4 API*. https://javadoc.io/doc/com.paypal.sdk/checkout-sdk/latest/index.html. Acedido em 20 de Set. de 2021.
[87] *rest-api-sdk 1.14.0 API*. https://javadoc.io/doc/com.paypal.sdk/rest-api-sdk/latest/index.html. Acedido em 20 de Set. de 2021.
[88] *AWS SDK for Java API Reference - 1.12.70*. https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/. Acedido em 20 de Set. de 2021.
[89] *OkHttp*. https://square.github.io/okhttp/. Acedido em 20 de Set. de 2021.
[90] *JUnit 5*. https://junit.org/junit5/. Acedido em 20 de Set. de 2021.
[91] *Mockito*. https://site.mockito.org/. Acedido em 20 de Set. de 2021.
[92] *REST-assured*. https://rest-assured.io/. Acedido em 20 de Set. de 2021.
[93] *H2 Database Engine*. https://www.h2database.com/html/main.html. Acedido em 20 de Set. de 2021.
[94] Eran Levy. *Kafka vs. RabbitMQ: Architecture, Performance and Use Cases*. https://www.upsolver.com/blog/kafka-versus-rabbitmq-architecture-performance-use-case. Acedido em 21 de Jul. de 2021. Mai. de 2019.
[95] Vijay Nair. *Axon and Kafka - How does Axon compare to Apache Kafka?* https://axoniq.io/blog-overview/axon-and-kafka-how-does-axon-compare-to-apache-kafka. Acedido em 21 de Jul. de 2021.
[96] Jesper Hammarback. *Apache Kafka Is Not for Event Sourcing*. https://dzone.com/articles/apache-kafka-is-not-for-event-sourcing. Acedido em 21 de Jul. de 2021. Ago. de 2020.
[97] *Companies using Apache Kafka*. https://enlyft.com/tech/products/apache-kafka. Acedido em 28 de Jul. de 2021.
[98] *SubEtha Wiser*. https://github.com/voodoodyne/subethasmtp/blob/master/Wiser.md. Acedido em 13 de Set. de 2021.
[99] *Statistic*. https://plugins.jetbrains.com/plugin/4509-statistic. Acedido em 21 de Set. de 2021.
[100] *A Simple Understanding of Code Complexity*. https://www.codegrip.tech/productivity/a-simple-understanding-of-code-complexity/. Acedido em 21 de Set. de 2021.
[101] *Cyclomatic complexity*. https://www.ibm.com/docs/en/raa/6.1?topic=metrics-cyclomatic-complexity. Acedido em 21 de Set. de 2021.
[102] *CodeMR*. https://plugins.jetbrains.com/plugin/10811-codemr. Acedido em 21 de Set. de 2021.
[103] *Your teammate for Code Quality and Code Security*. https://www.sonarqube.org/. Acedido em 21 de Set. de 2021.
[104] *Apache JMeter*. https://jmeter.apache.org/. Acedido em 21 de Fev. de 2021.
[105] *Heroku Pricing*. https://www.heroku.com/pricing. Acedido em 7 de Out. de 2021.
[106] *UML Lab Modeling IDE*. https://marketplace.eclipse.org/content/uml-lab-modeling-ide. Acedido em 13 de Out. de 2021.

[107] *OpenAPI Specification*. `https://swagger.io/specification/`. Acedido em 13 de Out. de 2021.

[108] *Swagger2Markup*. `https://github.com/Swagger2Markup/swagger2markup`. Acedido em 13 de Out. de 2021.

[109] *Asciidoctor PDF*. `https://asciidoctor.org/docs/asciidoctor-pdf/`. Acedido em 13 de Out. de 2021.

Apêndice A

Diagrama de componentes do monólito

Neste anexo é mostrado o diagrama de componentes que mostra o ponto de vista lógico do monólito na granularidade de aplicação.

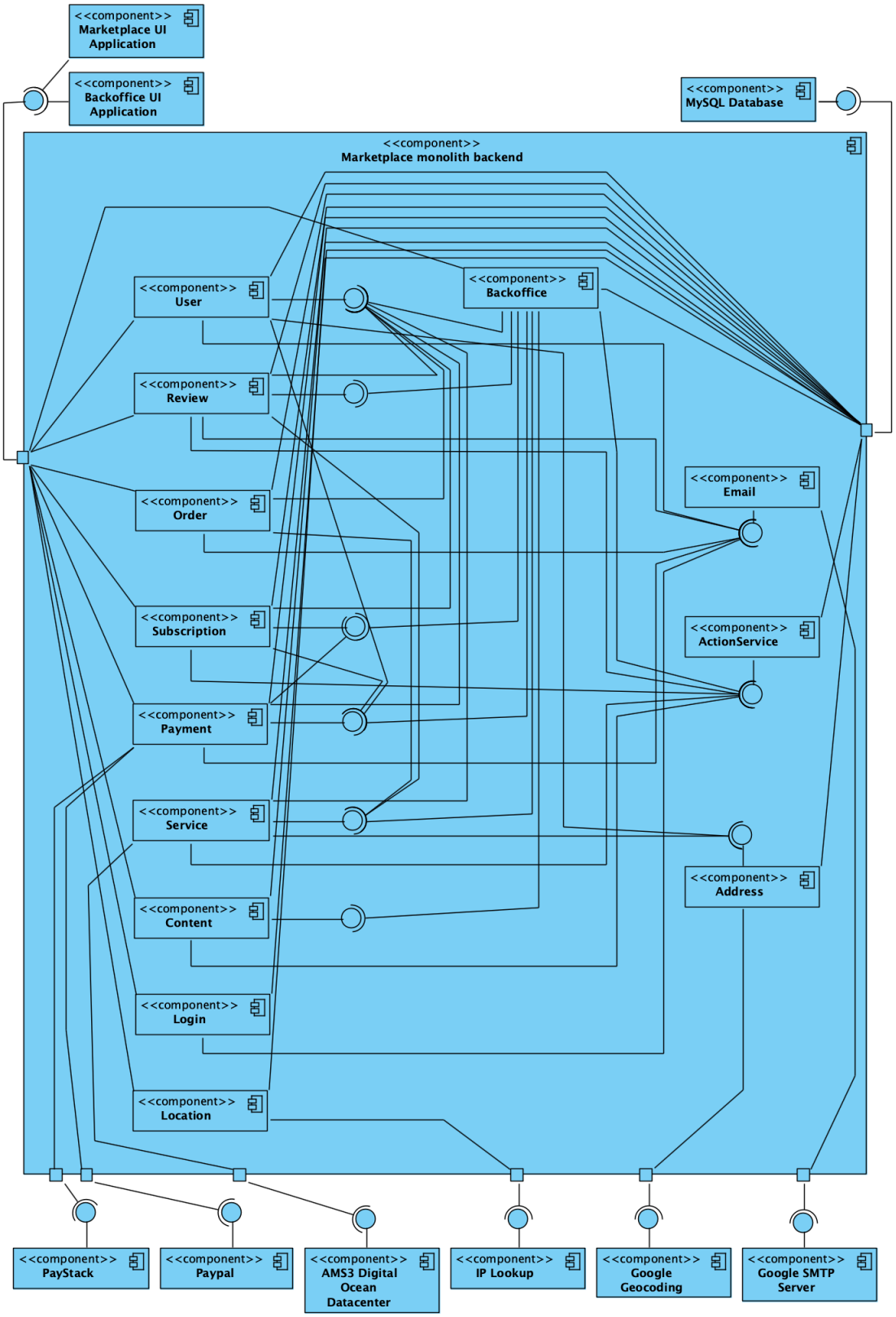


Figura A.1: Ponto de vista lógico na granularidade de aplicação do monólito, com todos os pormenores incluídos.

Apêndice B

Proposta de implantação dos micro-serviços em ambiente de produção

Neste anexo encontra-se um diagrama de implantação na granularidade de sistema dos micro-serviços, que serve como uma proposta para o ambiente de produção. Isto não se encontra configurado nem implantado.

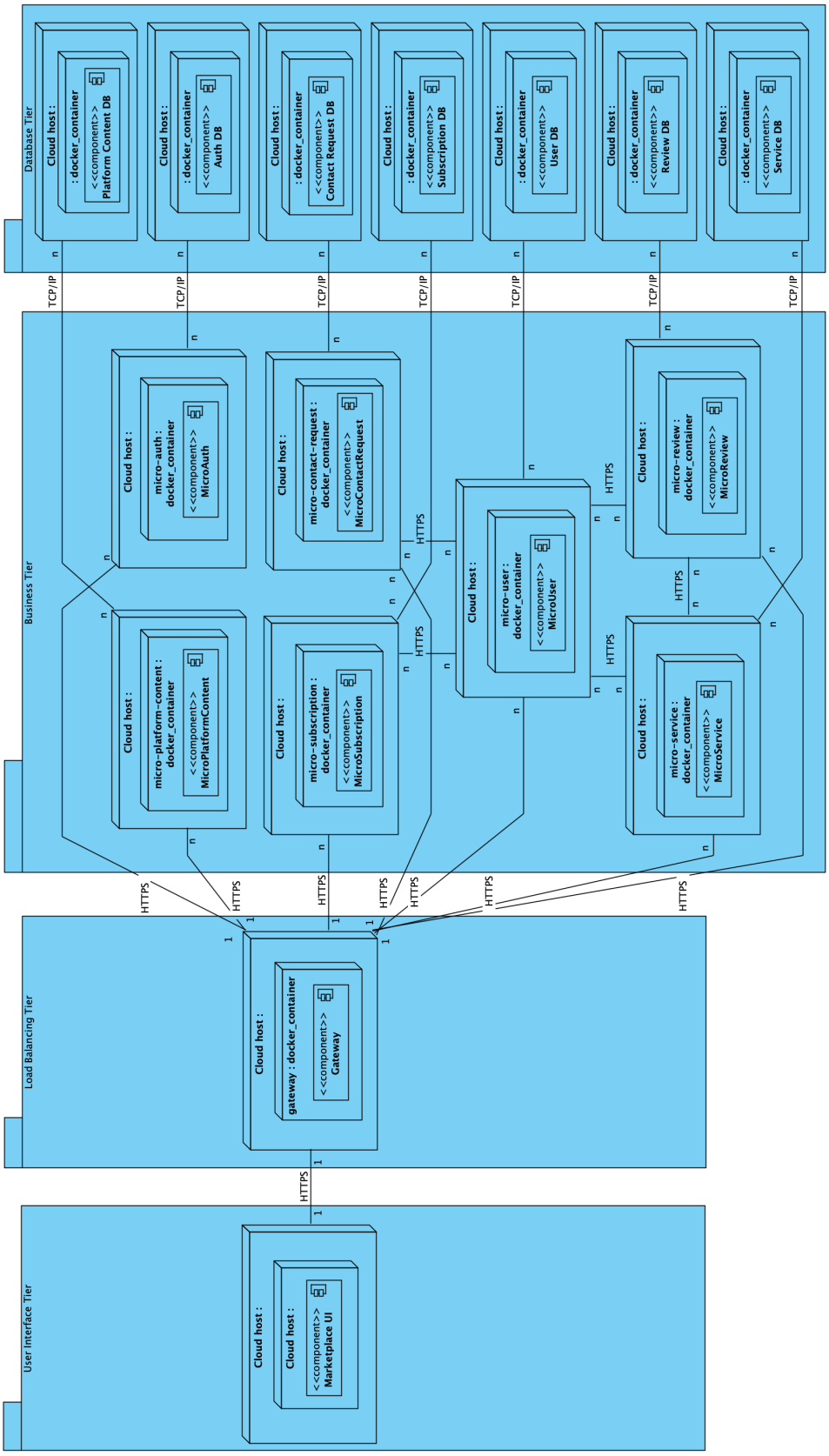


Figura B.1: Diagrama de implantação na granularidade de sistema dos micro-serviços, para o ambiente de produção.

Apêndice C

Especificações sobre o MicroAuth

Neste anexo encontram-se os diagramas de classes do MicroAuth, gerados usando UML Lab [106]; e, especificações sobre a API REST extraída a partir da OpenAPI [107] usando Swagger2Markup [108] e Asciidoctor PDF [109].

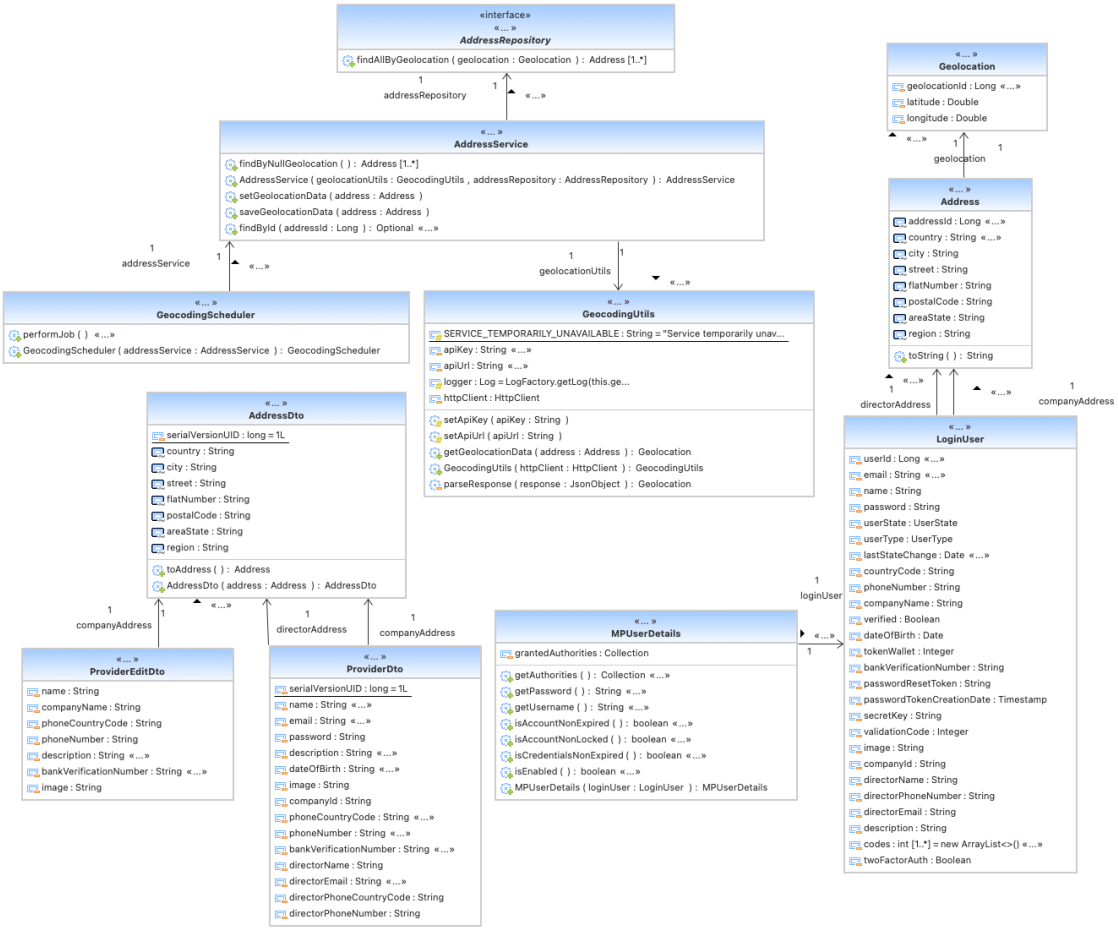


Figura C.1: Modelo de domínio implementado no MicroAuth, parte 1 de 2.

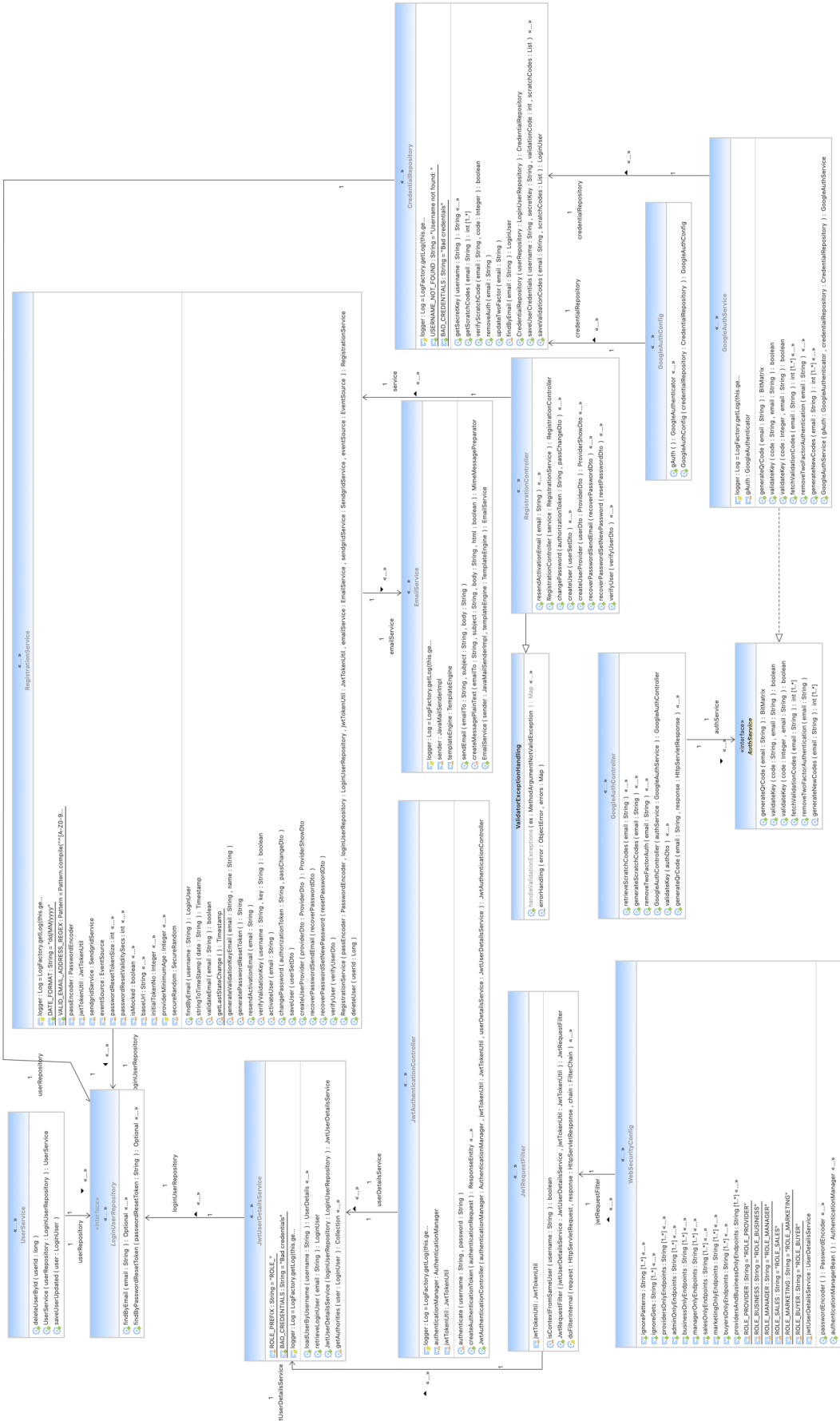


Figura C.2: Modelo de domínio implementado no MicroAuth, parte 2 de 2.

MicroAuth REST API

Overview

Documentation for MicroAuth API

Version information

Version : 1.0.0

URI scheme

Host : localhost:8082
BasePath : /api

Tags

- GeocodingAPIMock : Geocoding API Mock
- LoginController : Jwt Authentication Controller
- RegistrationController : Registration Controller
- the Authentication API : Google Auth Controller

Produces

- `application/json`

Paths

generateQrCode

```
GET /code/generate
```

Parameters

Type	Name	Description	Schema
Query	**email** *required*	email	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- the Authentication API

retrieveScratchCodes

```
GET /code/scratch
```

Parameters

Type	Name	Description	Schema
Query	**email** *required*	email	string

Responses

HTTP Code	Description	Schema
200	OK	ScratchCodesDto
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- the Authentication API

generateScratchCodes

```
PATCH /code/scratch
```

Parameters

Type	Name	Description	Schema
Query	**email** *required*	email	string

Responses

HTTP Code	Description	Schema
200	OK	ScratchCodesDto
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- the Authentication API

removeTwoFactorAuth

```
PATCH /code/user
```

Parameters

Type	Name	Description	Schema
Query	**email** *required*	email	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- the Authentication API

validateKey

```
POST /code/validate
```

Parameters

Type	Name	Description	Schema
Body	**authDto** *required*	authDto	AuthenticationDto

Responses

HTTP Code	Description	Schema
200	OK	ValidationDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- the Authentication API

resendActivationEmail

```
POST /email/activation
```

Parameters

Type	Name	Description	Schema
Query	**to** *required*	to	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- RegistrationController

getUserServicesFromToken

```
GET /geocodemock
```

Parameters

Type	Name	Description	Schema
Query	**address** *required*	address	string
Query	**key** *required*	key	string

Responses

HTTP Code	Description	Schema
200	OK	string
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- GeocodingAPIMock

createAuthenticationToken

```
POST /login
```

Parameters

Type	Name	Description	Schema
Body	**authenticationRequest** *required*	authenticationRequest	JwtRequest

Responses

HTTP Code	Description	Schema
200	OK	JwtResponse
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- LoginController

changePassword

```
PUT /password/change
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**passChangeDto** *required*	passChangeDto	PasswordChangeDto

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- RegistrationController

recoverPasswordSendEmail

```
POST /password/forgotten
```

Parameters

Type	Name	Description	Schema
Body	**recoverPasswordDto** *required*	recoverPasswordDto	RecoverPasswordDto

Responses

HTTP Code	Description	Schema
201	Created	No Content

HTTP Code	Description	Schema
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- RegistrationController

recoverPasswordSetNewPassword

```
POST /password/recover
```

Parameters

Type	Name	Description	Schema
Body	**resetPasswordDto** *required*	resetPasswordDto	ResetPasswordDto

Responses

HTTP Code	Description	Schema
201	Created	No Content
204	No Content	No Content

HTTP Code	Description	Schema
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- RegistrationController

createUser

```
POST /register
```

Parameters

Type	Name	Description	Schema
Body	**userSetDto** *required*	userSetDto	UserSetDto

Responses

HTTP Code	Description	Schema
201	Created	ShowLoginUserDto
401	Unauthorized	No Content
403	Forbidden	No Content

HTTP Code	Description	Schema
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- RegistrationController

createUserProvider

```
POST /register/provider
```

Parameters

Type	Name	Description	Schema
Body	**userDto** *required*	userDto	ProviderDto

Responses

HTTP Code	Description	Schema
201	Created	ProviderShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- RegistrationController

verifyUser

```
POST /user/verified
```

Parameters

Type	Name	Description	Schema
Body	**verifyUserDto** *required*	verifyUserDto	VerifyUserDto

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- RegistrationController

Apêndice D

Especificações sobre o MicroUser

Neste anexo encontram-se os diagramas de classes do MicroUser, gerados usando UML Lab [106]; e, especificações sobre a API REST extraída a partir da OpenAPI [107] usando Swagger2Markup [108] e Asciidoctor PDF [109].

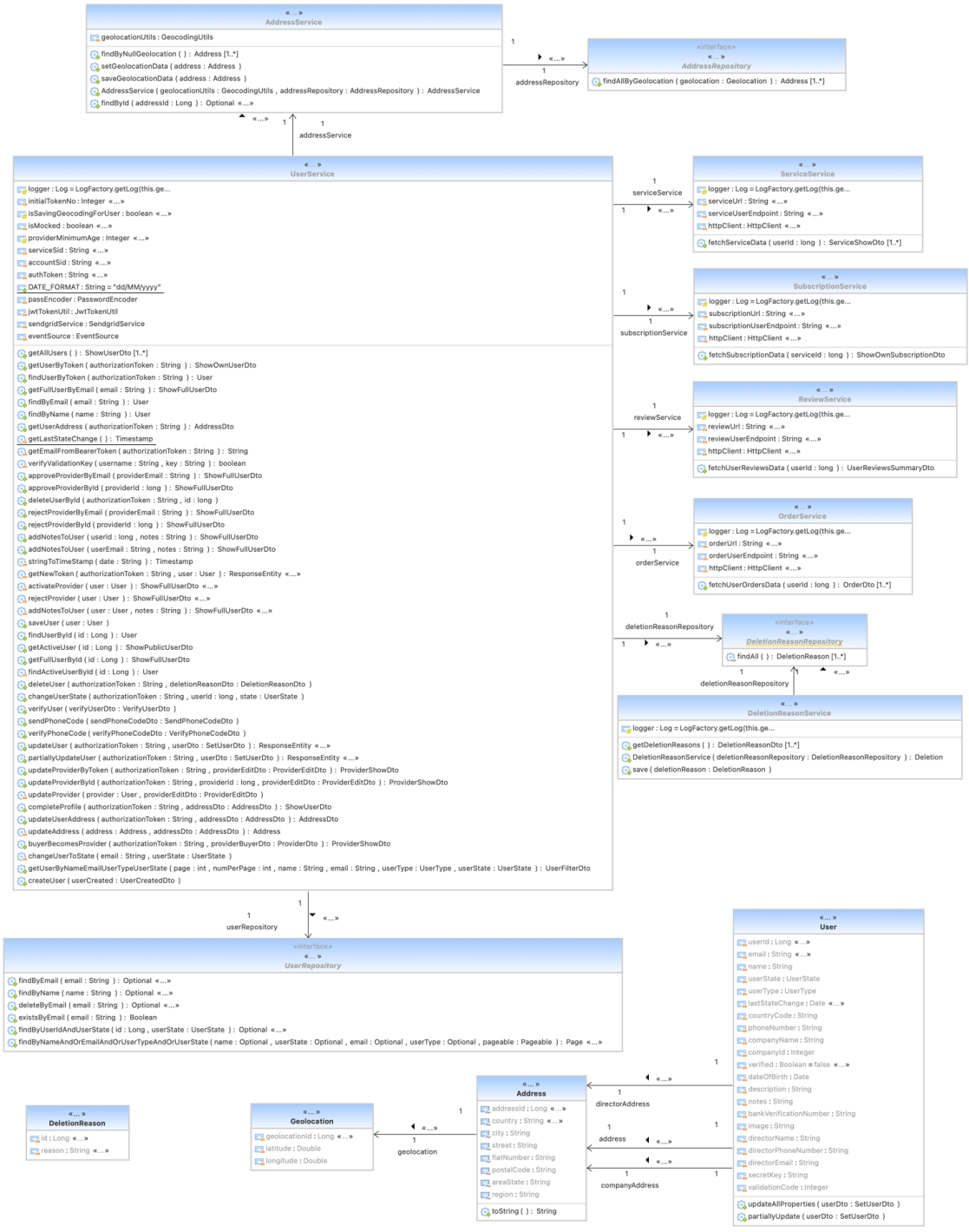


Figura D.1: Modelo de domínio implementado no MicroUser.

MicroUser REST API

Overview

Documentation for MicroUser API

Version information

Version : 1.0.0

URI scheme

Host : localhost:8083
BasePath : /api

Tags

- GeocodingAPIMock : Geocoding API Mock
- UserController : Admin User Controller

Produces

- `application/json`

Paths

getUserServicesFromToken

```
GET /geocodemock
```

Parameters

Type	Name	Description	Schema
Query	**address** *required*	address	string
Query	**key** *required*	key	string

Responses

HTTP Code	Description	Schema
200	OK	string
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- GeocodingAPIMock

getUserByToken

```
GET /user/admin
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	ShowOwnUserDto
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

updateUser

```
PUT /user/admin
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**userDto** *required*	userDto	SetUserDto

Responses

HTTP Code	Description	Schema
200	OK	JwtResponse
201	Created	No Content
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

deleteUser

```
DELETE /user/admin
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**deletionReasonDto** *required*	deletionReasonDto	DeletionReasonDto

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

partiallyUpdateUser

```
PATCH /user/admin
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**userDto** *required*	userDto	SetUserDto

Responses

HTTP Code	Description	Schema
200	OK	JwtResponse
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

getUserAddress

```
GET /user/admin/address
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	AddressDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

updateUserAddress

```
PUT /user/admin/address
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**addressDto** *required*	addressDto	AddressDto

Responses

HTTP Code	Description	Schema
200	OK	AddressDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

becomeProvider

```
POST /user/admin/become/provider
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**providerDto** *required*	providerDto	ProviderDto

Responses

HTTP Code	Description	Schema
200	OK	ProviderShowDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

completeProfile

```
POST /user/admin/complete
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**addressDto** *required*	addressDto	AddressDto

Responses

HTTP Code	Description	Schema
200	OK	ShowUserDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

updateProvider

```
PUT /user/admin/provider
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**providerEditDto** *required*	providerEditDto	ProviderEditDto

Responses

HTTP Code	Description	Schema
200	OK	ProviderShowDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

getMadeBuyerReviews

```
GET /user/admin/review/buyer/made
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

getReceivedBuyerReviews

```
GET /user/admin/review/buyer/received
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

getReceivedProviderReviews

```
GET /user/admin/review/provider/received
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

getMadeServiceReviews

```
GET /user/admin/review/service/made
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

getOwnSubscription

```
GET /user/admin/subscription
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	ShowOwnSubscriptionDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

getUserById

```
GET /user/id/{id}
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	ShowPublicUserDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

sendPhoneCode

```
POST /user/phone/code
```

Parameters

Type	Name	Description	Schema
Body	**sendPhoneCodeDto** *required*	sendPhoneCodeDto	SendPhoneCodeDto

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

verifyPhoneCode

```
POST /user/phone/verify
```

Parameters

Type	Name	Description	Schema
Body	**verifyPhoneCodeDto** *required*	verifyPhoneCodeDto	VerifyPhoneCodeDto

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

verifyUser

```
POST /user/verified
```

Parameters

Type	Name	Description	Schema
Body	**verifyUserDto** *required*	verifyUserDto	VerifyUserDto

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserController

Apêndice E

Especificações sobre o MicroService

Neste anexo encontram-se os diagramas de classes do MicroService, gerados usando UML Lab [106]; e, especificações sobre a API REST extraída a partir da OpenAPI [107] usando Swagger2Markup [108] e Asciidoctor PDF [109].

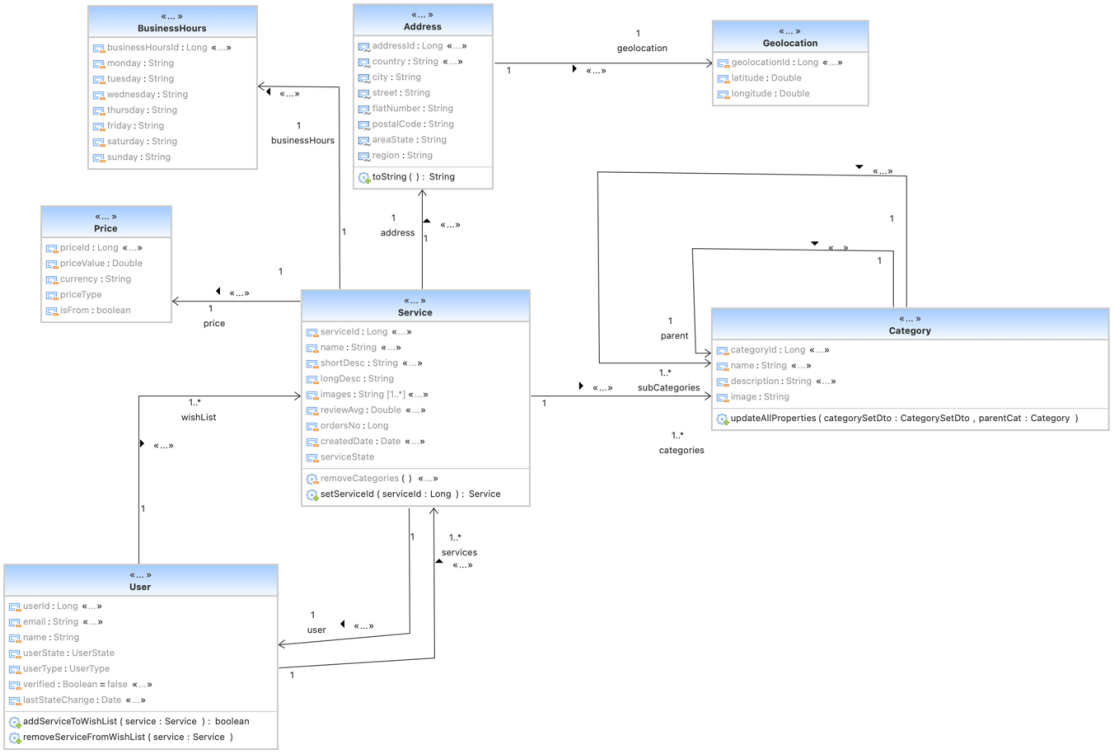


Figura E.1: Modelo de domínio implementado no MicroService, parte 1 de 2.

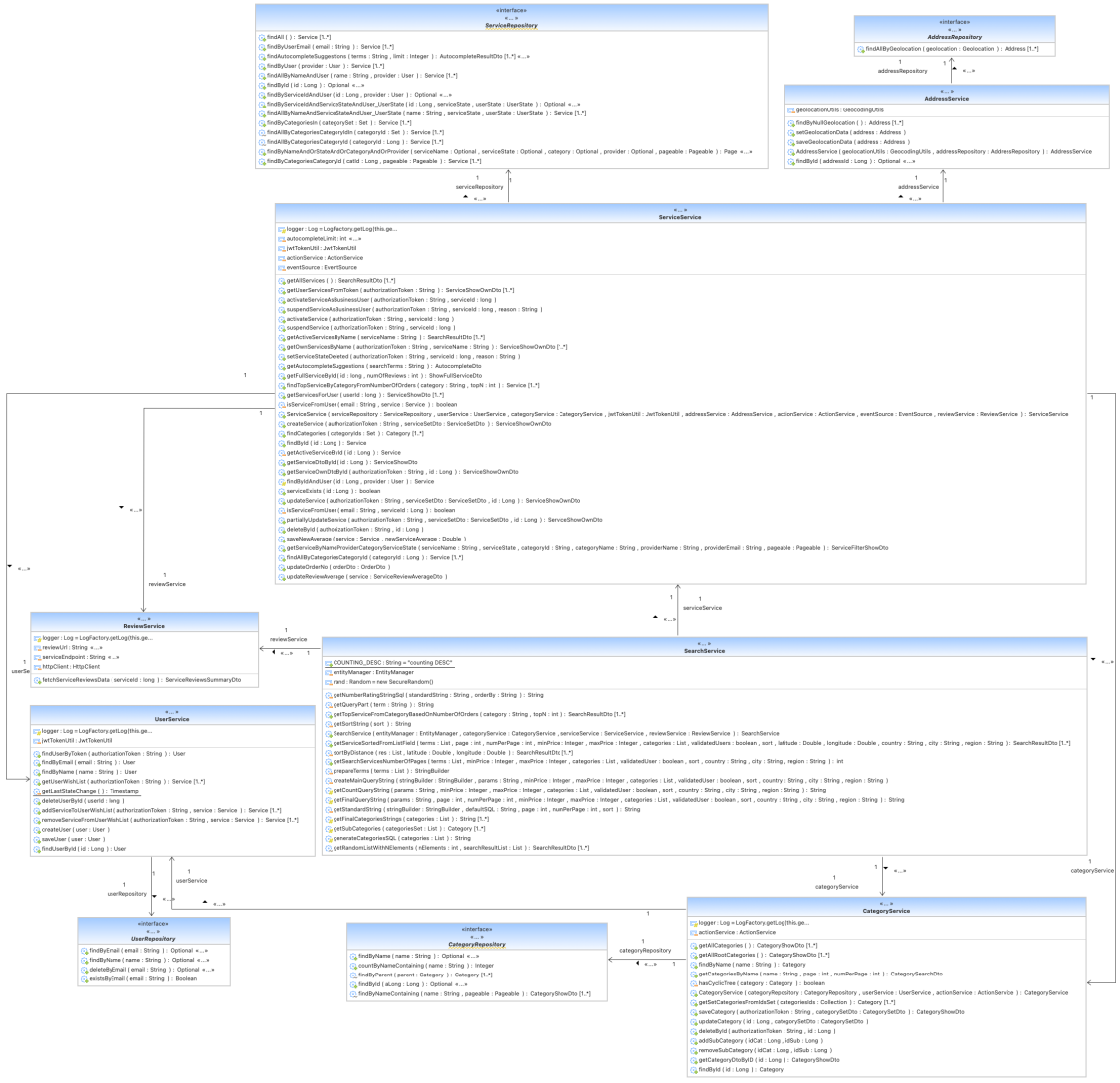


Figura E.2: Modelo de domínio implementado no MicroService, parte 2 de 2.

MicroService REST API

Overview

Documentation for MicroService API

Version information

Version : 1.0.0

URI scheme

Host : localhost:8084
BasePath : /api

Tags

- CategoryController : Category Controller
- GeocodingAPIMock : Geocoding API Mock
- Image upload service mock : Image Upload Service Mock
- SearchController : Search Controller
- ServiceController : Admin Service Controller
- UploadController : Upload Controller
- UserServiceController : User Controller
- WishListController : Wish List Controller

Produces

- `application/json`

Paths

createCategory

```
POST /category
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**categorySetDto** *required*	categorySetDto	CategorySetDto

Responses

HTTP Code	Description	Schema
201	Created	CategoryShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- CategoryController

getAllCategories

```
GET /category/all
```

Responses

HTTP Code	Description	Schema
200	OK	< CategoryShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- CategoryController

getCategoriesByName

```
GET /category/filter
```

Parameters

Type	Name	Description	Schema	Default
Query	**name** *optional*	name	string	
Query	**numPerPage** *optional*	numPerPage	integer (int32)	`10`
Query	**page** *optional*	page	integer (int32)	`1`

Responses

HTTP Code	Description	Schema
200	OK	CategorySearchDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- CategoryController

getAllRootCategories

```
GET /category/root
```

Responses

HTTP Code	Description	Schema
200	OK	< CategoryShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- CategoryController

addSubCategory

```
POST /category/{idCat}/addsubcat/{idSub}
```

Parameters

Type	Name	Description	Schema
Path	**idCat** *required*	idCat	integer (int64)
Path	**idSub** *required*	idSub	integer (int64)

Responses

HTTP Code	Description	Schema
201	Created	No Content
202	Accepted	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- CategoryController

removeSubCategory

```
POST /category/{idCat}/removesubcat/{idSub}
```

Parameters

Type	Name	Description	Schema
Path	**idCat** *required*	idCat	integer (int64)
Path	**idSub** *required*	idSub	integer (int64)

Responses

HTTP Code	Description	Schema
201	Created	No Content
202	Accepted	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- CategoryController

findCategoryById

```
GET /category/{id}
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	CategoryShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- CategoryController

updateCategory

```
PUT /category/{id}
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	integer (int64)
Body	**categorySetDto** *required*	categorySetDto	CategorySetDto

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- CategoryController

deleteCategory

```
DELETE /category/{id}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**id** *required*	id	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- CategoryController

getUserServicesFromToken

```
GET /geocodemock
```

Parameters

Type	Name	Description	Schema
Query	**address** *required*	address	string

Type	Name	Description	Schema
Query	**key** *required*	key	string

Responses

HTTP Code	Description	Schema
200	OK	string
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- GeocodingAPIMock

suspendService

```
POST /saracen-images/{param}
```

Parameters

Type	Name	Description	Schema
Path	**param** *required*	param	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- Image upload service mock

getAllServicesMatchingNameShortAndLongDescription

```
GET /search
```

Parameters

Type	Name	Description	Schema	Default
Query	**cat** *optional*	cat	< string > array(multi)	
Query	**city** *optional*	city	string	
Query	**country** *optional*	country	string	

Type	Name	Description	Schema	Default
Query	**latitude** *optional*	latitude	number (double)	
Query	**longitude** *optional*	longitude	number (double)	
Query	**maxPrice** *optional*	maxPrice	integer (int32)	
Query	**minPrice** *optional*	minPrice	integer (int32)	
Query	**numPerPage** *optional*	numPerPage	integer (int32)	`10`
Query	**page** *optional*	page	integer (int32)	`1`
Query	**region** *optional*	region	string	
Query	**sort** *optional*	sort	enum (RATING_DESC, RATING_ASC, OLDEST, RECENT, PRICE_ASC, PRICE_DESC, REVIEW_NO_DESC, REVIEW_NO_ASC, DISTANCE)	
Query	**term** *optional*	term	string	
Query	**validated** *optional*	validated	boolean	

Responses

HTTP Code	Description	Schema
200	OK	SearchDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SearchController

getAutocompleteSuggestions

```
GET /search/autocomplete
```

Parameters

Type	Name	Description	Schema
Query	**term** *required*	term	string

Responses

HTTP Code	Description	Schema
200	OK	AutocompleteDto
401	Unauthorized	No Content
403	Forbidden	No Content

HTTP Code	Description	Schema
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SearchController

getTopServicesFromCategory

```
GET /search/top/category/{categoryId}
```

Parameters

Type	Name	Description	Schema	Default
Path	**categoryId** *required*	categoryId	string	
Query	**topN** *optional*	topN	integer (int32)	5

Responses

HTTP Code	Description	Schema
200	OK	< SearchResultDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SearchController

createService

```
POST /service/admin
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**serviceSetDto** *required*	serviceSetDto	ServiceSetDto

Responses

HTTP Code	Description	Schema
201	Created	ServiceShowOwnDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

getUserServicesFromToken

```
GET /service/admin
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ServiceShowOwnDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

activateService

```
POST /service/admin/activate/{serviceId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**serviceId** *required*	serviceId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

getServiceByName

```
GET /service/admin/name/{serviceName}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**serviceName** *required*	serviceName	string

Responses

HTTP Code	Description	Schema
200	OK	< ServiceShowOwnDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

suspendService

```
POST /service/admin/suspend/{serviceId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**serviceId** *required*	serviceId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

getServiceById

```
GET /service/admin/{id}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**id** *required*	id	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	ServiceShowOwnDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

updateService

```
PUT /service/admin/{id}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Type	Name	Description	Schema
Path	**id** *required*	id	integer (int64)
Body	**serviceSetDto** *required*	serviceSetDto	ServiceSetDto

Responses

HTTP Code	Description	Schema
200	OK	ServiceShowOwnDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

deleteServiceById

```
DELETE /service/admin/{id}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**id** *required*	id	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

partiallyUpdateService

```
PATCH /service/admin/{id}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Type	Name	Description	Schema
Path	**id** *required*	id	integer (int64)
Body	**serviceSetDto** *required*	serviceSetDto	ServiceSetDto

Responses

HTTP Code	Description	Schema
200	OK	ServiceShowOwnDto
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

getServiceByName

```
GET /service/name/{serviceName}
```

Parameters

Type	Name	Description	Schema
Path	**serviceName** *required*	serviceName	string

Responses

HTTP Code	Description	Schema
200	OK	< SearchResultDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

getServiceById

```
GET /service/{id}
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	ServiceShowDto
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceController

uploadImage

```
POST /upload/image
```

Parameters

Type	Name	Description	Schema
FormData	**file** *required*	file	file

Responses

HTTP Code	Description	Schema
200	OK	ShowUrlDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `multipart/form-data`

Produces

- `application/json`

Tags

- UploadController

deleteImage

```
DELETE /upload/image/{fileKey}
```

Parameters

Type	Name	Description	Schema
Path	**fileKey** *required*	fileKey	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UploadController

getMostRecentUserReviews

```
GET /user/service/{userId}
```

Parameters

Type	Name	Description	Schema
Path	**userId** *required*	userId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	< ServiceShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserServiceController

getUserWishList

```
GET /wishlist
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ShowWishlistItem Dto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- WishListController

addServiceToUserWishList

```
POST /wishlist/add/{serviceId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Type	Name	Description	Schema
Path	**serviceId** *required*	serviceId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	< ShowWishlistItem Dto > array
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- WishListController

removeServiceToUserWishList

```
POST /wishlist/remove/{serviceId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Type	Name	Description	Schema
Path	**serviceId** *required*	serviceId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	< ShowWishlistItem Dto > array
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- WishListController

Apêndice F

Especificações sobre o MicroReview

Neste anexo encontram-se os diagramas de classes do MicroReview, gerados usando UML Lab [106]; e, especificações sobre a API REST extraída a partir da OpenAPI [107] usando Swagger2Markup [108] e Asciidoctor PDF [109].

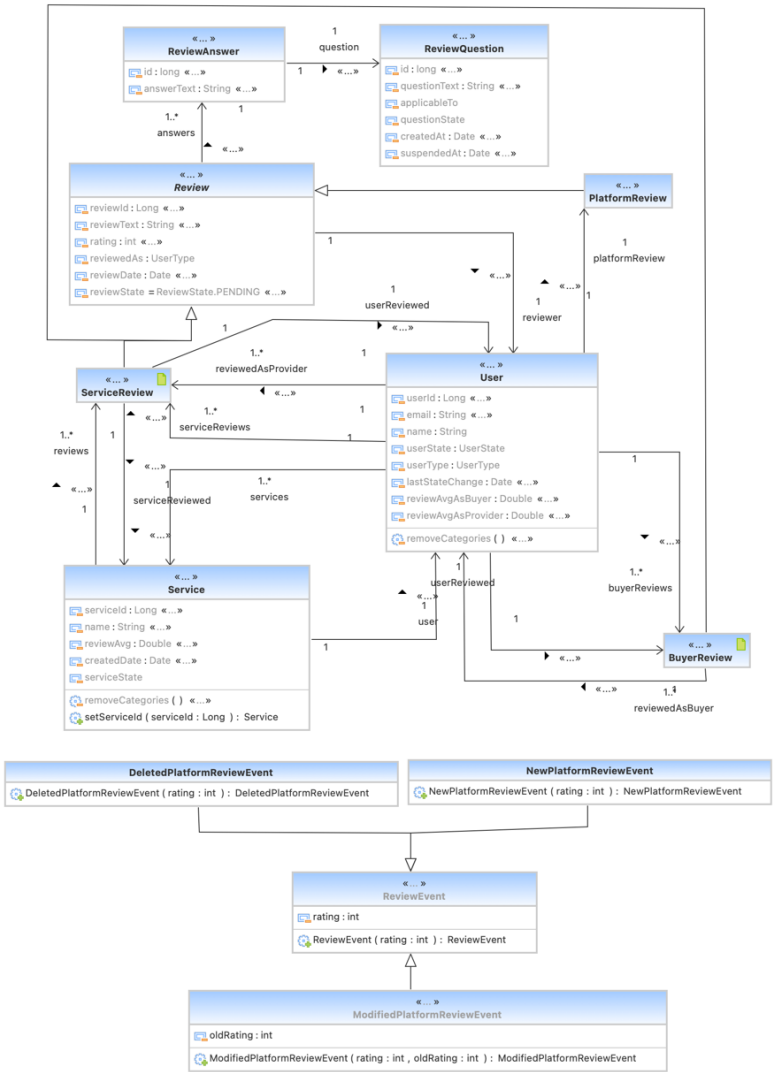


Figura F.1: Modelo de domínio implementado no MicroReview, parte 1 de 2.

Figura F.2: Modelo de domínio implementado no MicroReview, parte 2 de 2.

MicroReview REST API

Overview

Documentation for MicroReview API

Version information

Version : 1.0.0

URI scheme

Host : localhost:8087
BasePath : /api

Tags

- PlatformReviewController : Platform Review Controller
- ReviewController : Review Controller
- ServiceReviewsController : Service Controller
- UserReviewsController : User Controller

Produces

- `application/json`

Paths

getUserBuyerReviews

```
GET /review/buyer
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getMadeBuyerReviews

```
GET /review/buyer/made
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array

HTTP Code	Description	Schema
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getReceivedBuyerReviews

```
GET /review/buyer/received
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content

HTTP Code	Description	Schema
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getMostRecentBuyerReviews

```
GET /review/buyer/{buyerId}/page/{page}
```

Parameters

Type	Name	Description	Schema
Path	**buyerId** *required*	buyerId	integer (int64)
Path	**page** *required*	page	integer (int32)

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- application/json

Produces

- application/json

Tags

- ReviewController

getBuyerReviewsByRating

```
GET /review/buyer/{buyerId}/rating/{rating}
```

Parameters

Type	Name	Description	Schema
Path	**buyerId** *required*	buyerId	integer (int64)
Path	**rating** *required*	rating	integer (int32)

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- application/json

Produces

- `application/json`

Tags

- ReviewController

reviewUserByService

```
POST /review/buyer/{buyerId}/service/{serviceId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**buyerId** *required*	buyerId	integer (int64)
Path	**serviceId** *required*	serviceId	integer (int64)
Body	**reviewSetDto** *required*	reviewSetDto	ReviewSetDto

Responses

HTTP Code	Description	Schema
201	Created	ReviewShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getBuyerReviewById

```
GET /review/buyer/{reviewId}
```

Parameters

Type	Name	Description	Schema
Path	**reviewId** *required*	reviewId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	ReviewShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

modifyBuyerReview

```
PUT /review/buyer/{reviewId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**reviewId** *required*	reviewId	integer (int64)
Body	**reviewEditDto** *required*	reviewEditDto	ReviewEditDto

Responses

HTTP Code	Description	Schema
200	OK	ReviewShowDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

deleteBuyerReview

```
DELETE /review/buyer/{reviewId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**reviewId** *required*	reviewId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

reviewPlatform

```
POST /review/platform
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**platformreviewSetDto** *required*	platformreviewSetDto	PlatformReviewSetDto

Responses

HTTP Code	Description	Schema
201	Created	PlatformReviewShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PlatformReviewController

getOwnPlatformReview

```
GET /review/platform
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	PlatformReviewShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PlatformReviewController

updatePlatformReview

```
PUT /review/platform
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Type	Name	Description	Schema
Body	**platformReviewEditDto** *required*	platformReviewEditDto	PlatformReviewEditDto

Responses

HTTP Code	Description	Schema
200	OK	PlatformReviewShowDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PlatformReviewController

deletePlatformReview

```
DELETE /review/platform
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PlatformReviewController

filterPlatformReviewsByRating

```
GET /review/platform/rating/{number}
```

Parameters

Type	Name	Description	Schema
Path	**number** *required*	number	integer (int32)

Responses

HTTP Code	Description	Schema
200	OK	< PlatformReviewShowDto > array

HTTP Code	Description	Schema
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PlatformReviewController

getPlatformReviewSummary

```
GET /review/platform/summary
```

Responses

HTTP Code	Description	Schema
200	OK	PlatformReviewSummaryDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PlatformReviewController

getPlatformReviewById

```
GET /review/platform/{reviewId}
```

Parameters

Type	Name	Description	Schema
Path	**reviewId** *required*	reviewId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	PlatformReviewShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PlatformReviewController

approveOrRejectPlatformReview

```
PUT /review/platform/{reviewId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**reviewId** *required*	reviewId	integer (int64)
Body	**businessReviewEditDto** *required*	businessReviewEditDto	BusinessReviewEditDto

Responses

HTTP Code	Description	Schema
200	OK	PlatformReviewShowDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PlatformReviewController

getReceivedProviderReviews

```
GET /review/provider/received
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getProviderReviewsByRating

```
GET /review/provider/{providerId}/rating/{rating}
```

Parameters

Type	Name	Description	Schema
Path	**providerId** *required*	providerId	integer (int64)
Path	**rating** *required*	rating	integer (int32)

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

createReviewQuestion

```
POST /review/question
```

Parameters

Type	Name	Description	Schema
Body	**reviewQuestionSetDto** *required*	reviewQuestionSetDto	ReviewQuestionSetDto

Responses

HTTP Code	Description	Schema
200	OK	ReviewQuestion
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getAllReviewQuestions

```
GET /review/question
```

Responses

HTTP Code	Description	Schema
200	OK	< ReviewQuestion > array

HTTP Code	Description	Schema
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getBuyerReviewQuestions

```
GET /review/question/buyer
```

Responses

HTTP Code	Description	Schema
200	OK	< ReviewQuestion > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getPlatformAsBuyerReviewQuestions

```
GET /review/question/platform/buyer
```

Responses

HTTP Code	Description	Schema
200	OK	< ReviewQuestion > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getPlatformAsProviderReviewQuestions

```
GET /review/question/platform/provider
```

Responses

HTTP Code	Description	Schema
200	OK	< ReviewQuestion > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getServiceReviewQuestions

```
GET /review/question/service
```

Responses

HTTP Code	Description	Schema
200	OK	< ReviewQuestion > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

updateReviewQuestion

```
PUT /review/question/{id}
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	integer (int64)
Body	**reviewQuestionEditDto** *required*	reviewQuestionEditDto	ReviewQuestionEditDto

Responses

HTTP Code	Description	Schema
200	OK	ReviewQuestion
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

deleteReviewQuestion

```
DELETE /review/question/{id}
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getServiceReviews

```
GET /review/service
```

Parameters

Type	Name	Description	Schema
Query	**reviewState** *optional*	reviewState	enum (APPROVED, PENDING, REJECTED)
Query	**reviewerName** *optional*	reviewerName	string
Query	**serviceName** *optional*	serviceName	string

Responses

HTTP Code	Description	Schema
200	OK	Page«ReviewShowDto»
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getMadeServiceReviews

```
GET /review/service/made
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getServiceReviewById

```
GET /review/service/{reviewId}
```

Parameters

Type	Name	Description	Schema
Path	**reviewId** *required*	reviewId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	ReviewShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

modifyServiceReview

```
PUT /review/service/{reviewId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**reviewId** *required*	reviewId	integer (int64)

Type	Name	Description	Schema
Body	**reviewEditDto** *required*	reviewEditDto	ReviewEditDto

Responses

HTTP Code	Description	Schema
200	OK	ReviewShowDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

deleteServiceReview

```
DELETE /review/service/{reviewId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Type	Name	Description	Schema
Path	**reviewId** *required*	reviewId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

reviewService

```
POST /review/service/{serviceId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**serviceId** *required*	serviceId	integer (int64)

Type	Name	Description	Schema
Body	**reviewSetDto** *required*	reviewSetDto	ReviewSetDto

Responses

HTTP Code	Description	Schema
201	Created	ReviewShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getMostRecentServiceReviews

```
GET /review/service/{serviceId}/page/{page}
```

Parameters

Type	Name	Description	Schema
Path	**page** *required*	page	integer (int32)
Path	**serviceId** *required*	serviceId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getServiceReviewsByRating

```
GET /review/service/{serviceId}/rating/{rating}
```

Parameters

Type	Name	Description	Schema
Path	**rating** *required*	rating	integer (int32)
Path	**serviceId** *required*	serviceId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	< ReviewShowDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ReviewController

getMostRecentServiceReviews

```
GET /service/review/{serviceId}
```

Parameters

Type	Name	Description	Schema
Path	**serviceId** *required*	serviceId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	ServiceReviewsSummaryDto
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- ServiceReviewsController

getMostRecentUserReviews

```
GET /user/review/{userId}
```

Parameters

Type	Name	Description	Schema
Path	**userId** *required*	userId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	UserReviewsSummaryDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- application/json

Produces

- application/json

Tags

- UserReviewsController

Apêndice G

Especificações sobre o MicroPlatformContents

Neste anexo encontram-se os diagramas de classes do MicroPlatformContents, gerados usando UML Lab [106]; e, especificações sobre a API REST extraída a partir da OpenAPI [107] usando Swagger2Markup [108] e Asciidoctor PDF [109].

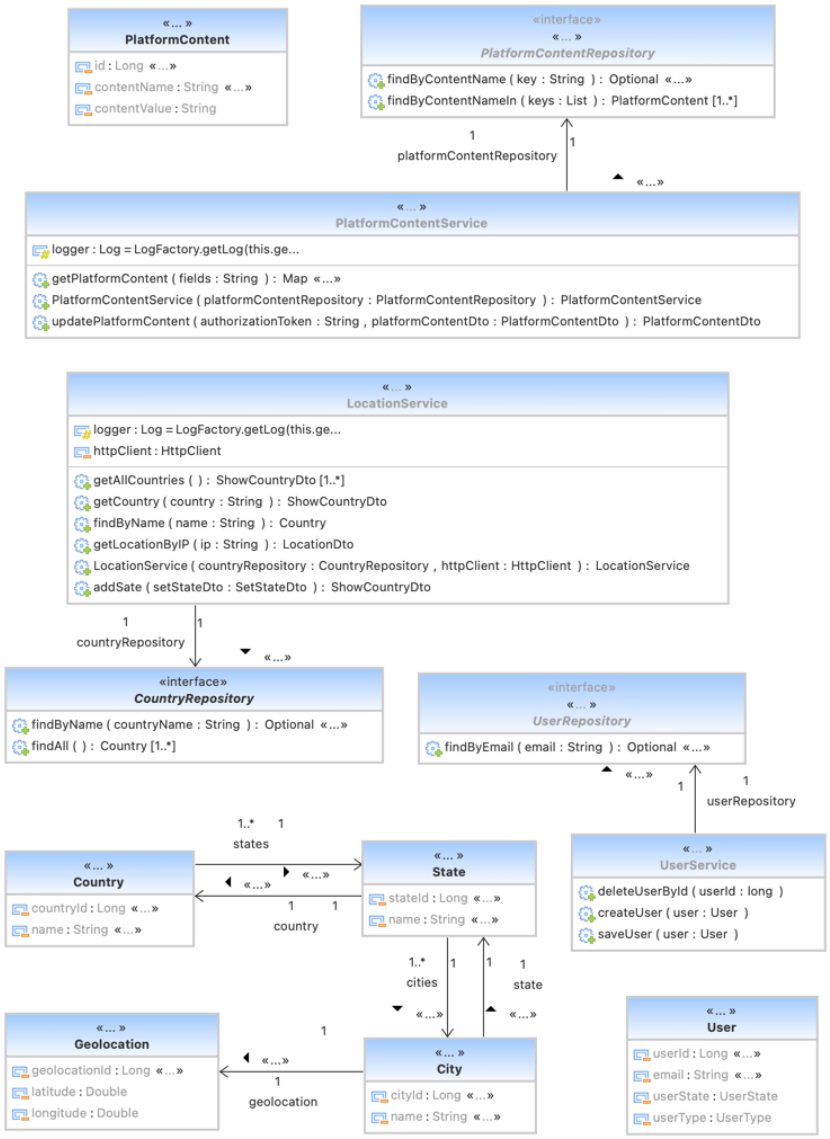


Figura G.1: Modelo de domínio implementado no MicroPlatformContents.

MictoPlatformContent REST API

Overview

Documentation for MictoPlatformContent API

Paths

getPlatformContent

```
GET /content
```

Parameters

Type	Name	Description	Schema
Query	**fields** *required*	fields	string

Responses

HTTP Code	Description	Schema
200	OK	< string, string > map
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PlatformContentController

getLocationByIP

```
GET /location
```

Parameters

Type	Name	Description	Schema
Header	**X-FORWARDED-FOR** *required*	X-FORWARDED-FOR	string

Responses

HTTP Code	Description	Schema
200	OK	LocationDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- LocationController

getAllLocations

```
GET /location/country/all
```

Responses

HTTP Code	Description	Schema
200	OK	< ShowCountryDto > array
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- LocationController

getCountry

```
GET /location/country/{countryName}
```

Parameters

Type	Name	Description	Schema
Path	**countryName** *required*	countryName	string

Responses

HTTP Code	Description	Schema
200	OK	ShowCountryDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- LocationController

addSate

```
POST /location/state
```

Parameters

Type	Name	Description	Schema
Body	**setStateDto** *required*	setStateDto	SetStateDto

Responses

HTTP Code	Description	Schema
200	OK	ShowCountryDto
201	Created	No Content
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- LocationController

Apêndice H

Especificações sobre o MicroSubscription

Neste anexo encontram-se os diagramas de classes do MicroSubscription, gerados usando UML Lab [106]; e, especificações sobre a API REST extraída a partir da OpenAPI [107] usando Swagger2Markup [108] e Asciidoctor PDF [109].

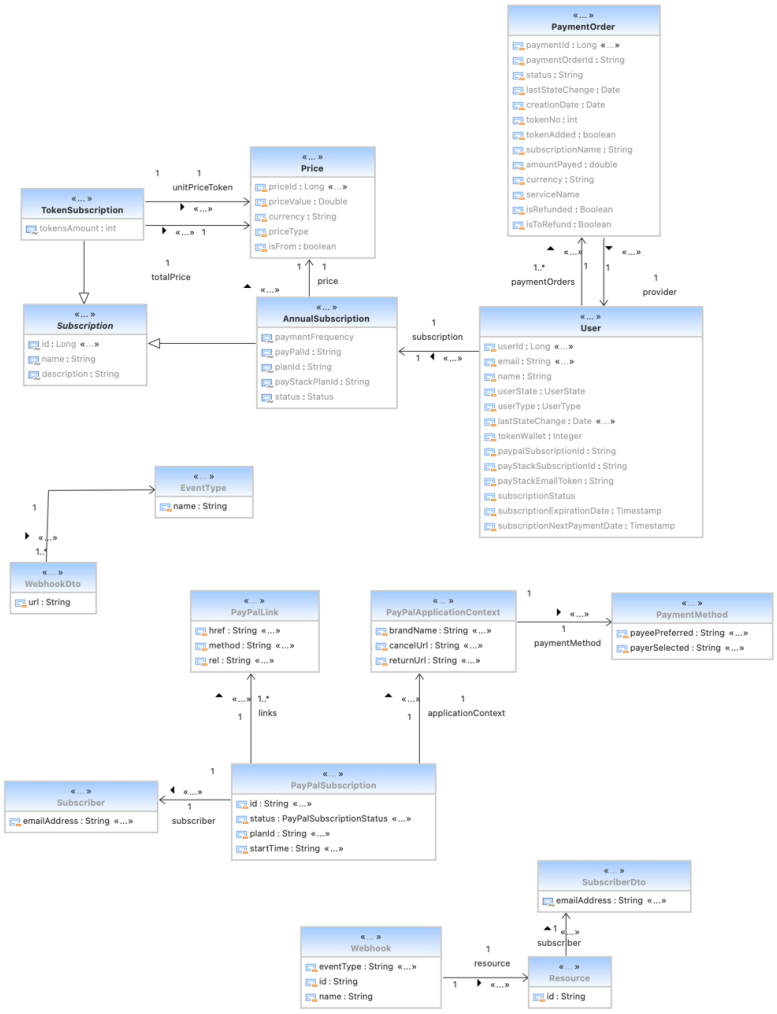


Figura H.1: Modelo de domínio implementado no MicroSubscription, parte 1 de 2.

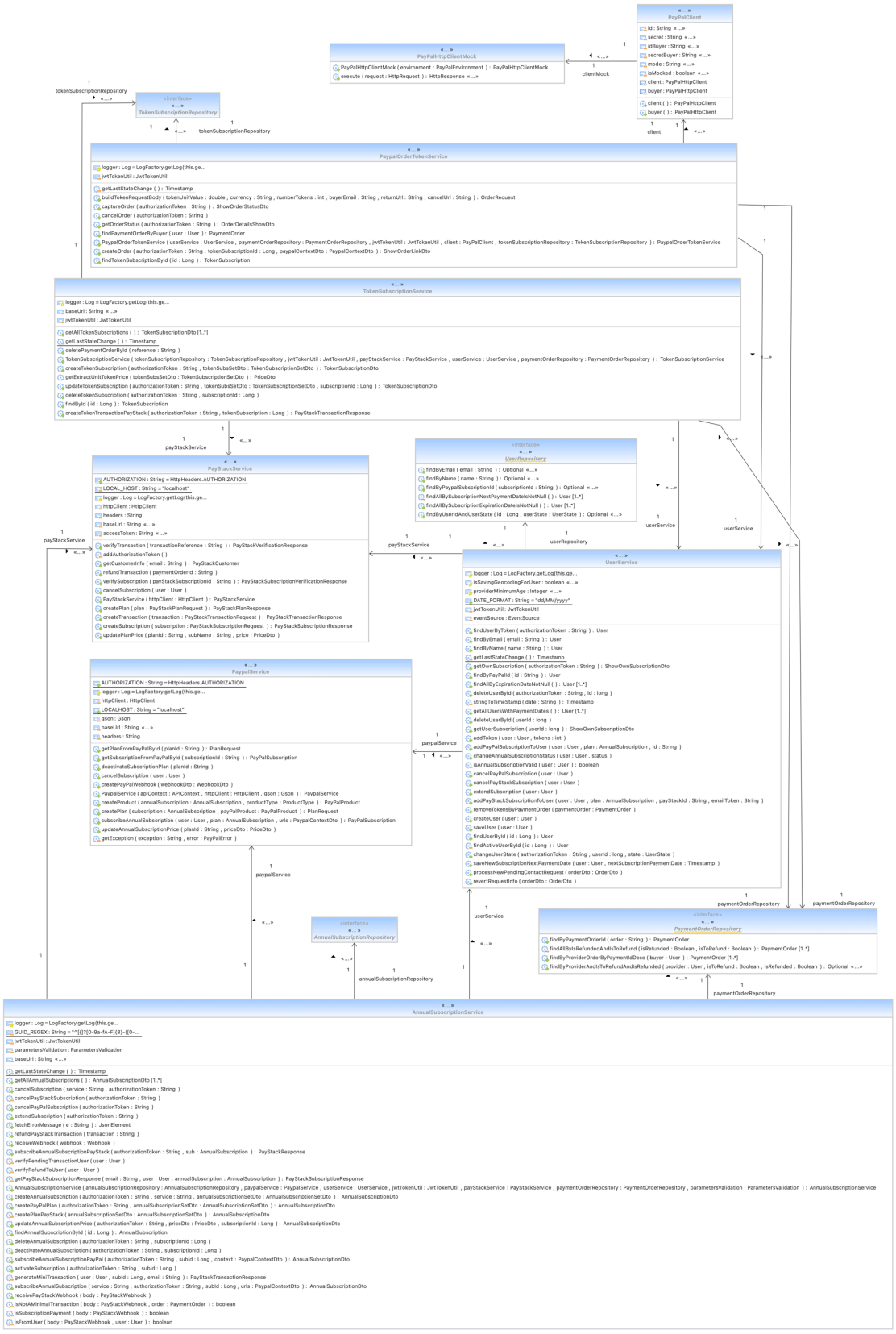


Figura H.2: Modelo de domínio implementado no MicroSubscription, parte 2 de 2.

MicroSubscription REST API

Overview

Documentation for MicroSubscription API

Version information

Version : 1.0.0

URI scheme

Host : localhost:8086
BasePath : /api

Tags

- BackofficeController : Payment Controller
- PayPalAPIMock : Pay Pal API Mock
- PayStackAPIMock : Pay Stack API Mock
- PayStackController : Pay Stack Controller
- PaymentController : Paypal Controller
- SubscriptionController : Subscription Controller
- UserSubscriptionController : User Controller

Produces

- `application/json`

Paths

cancelOrder

```
PUT /payment/token/cancel
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PaymentController

getOrder

```
GET /payment/token/order
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	OrderDetailsShowDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PaymentController

validateOrder

```
POST /payment/token/validate
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
201	Created	No Content

HTTP Code	Description	Schema
202	Accepted	ShowOrderStatus Dto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PaymentController

payToken

```
POST /payment/token/{tokenSubscription}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**tokenSubscription** *required*	tokenSubscription	integer (int64)
Body	**paypalContextDto** *required*	paypalContextDto	PaypalContextDto

Responses

HTTP Code	Description	Schema
201	Created	ShowOrderLinkDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PaymentController

deletePaymentOrderByPaymentOrderId

```
DELETE /payment/{reference}
```

Parameters

Type	Name	Description	Schema
Path	**reference** *required*	reference	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content

HTTP Code	Description	Schema
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- BackofficeController

createPlan

```
POST /paypalmock/v1/billing/plans
```

Parameters

Type	Name	Description	Schema
Body	**planRequest** *required*	planRequest	PlanRequest

Responses

HTTP Code	Description	Schema
201	Created	PlanRequest
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- application/json

Produces

- application/json

Tags

- PayPalAPIMock

getPayPalPlanById

```
GET /paypalmock/v1/billing/plans/{planId}
```

Parameters

Type	Name	Description	Schema
Path	**planId** *required*	planId	string

Responses

HTTP Code	Description	Schema
200	OK	PlanRequest
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- application/json

Produces

- application/json

Tags

- PayPalAPIMock

deactivatePlan

```
POST /paypalmock/v1/billing/plans/{subId}/deactivate
```

Parameters

Type	Name	Description	Schema
Path	**subId** *required*	subId	string

Responses

HTTP Code	Description	Schema
201	Created	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayPalAPIMock

updatePlan

```
POST /paypalmock/v1/billing/plans/{subId}/update-pricing-schemes
```

Parameters

Type	Name	Description	Schema
Path	**subId** *required*	subId	string
Body	**price** *required*	price	object

Responses

HTTP Code	Description	Schema
201	Created	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayPalAPIMock

createSubscription

```
POST /paypalmock/v1/billing/subscriptions
```

Parameters

Type	Name	Description	Schema
Body	**payPalSubscription** *required*	payPalSubscription	PayPalSubscription

Responses

HTTP Code	Description	Schema
201	Created	string
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayPalAPIMock

getPayPalSubscriptionById

```
GET /paypalmock/v1/billing/subscriptions/{subId}
```

Parameters

Type	Name	Description	Schema
Path	**subId** *required*	subId	string

Responses

HTTP Code	Description	Schema
200	OK	PayPalSubscription
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayPalAPIMock

cancelSubscription

```
POST /paypalmock/v1/billing/subscriptions/{subId}/cancel
```

Parameters

Type	Name	Description	Schema
Path	**subId** *required*	subId	string

Responses

HTTP Code	Description	Schema
201	Created	No Content
204	No Content	No Content

HTTP Code	Description	Schema
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayPalAPIMock

createProduct

```
POST /paypalmock/v1/catalogs/products
```

Parameters

Type	Name	Description	Schema
Body	**payPalProduct** *required*	payPalProduct	PayPalProduct

Responses

HTTP Code	Description	Schema
201	Created	PayPalProduct
401	Unauthorized	No Content
403	Forbidden	No Content

HTTP Code	Description	Schema
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayPalAPIMock

createPayPalWebhook

```
POST /paypalmock/v1/notifications/webhooks
```

Parameters

Type	Name	Description	Schema
Body	**webhookDto** *required*	webhookDto	WebhookDto

Responses

HTTP Code	Description	Schema
201	Created	Webhook
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayPalAPIMock

payToken

```
POST /paystack/token/{tokenSubscription}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**tokenSubscription** *required*	tokenSubscription	integer (int64)

Responses

HTTP Code	Description	Schema
201	Created	PayStackTransactionResponse
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackController

verifyTransaction

```
GET /paystack/verify/{id}
```

Parameters

Type	Name	Description	Schema
Path	**id** *required*	id	string

Responses

HTTP Code	Description	Schema
200	OK	PayStackVerificationResponse
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackController

getChargeSuccessWebhook

```
GET /paystackmock/chargesuccesswebhook
```

Responses

HTTP Code	Description	Schema
200	OK	PayStackWebhook
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackAPIMock

getCustomerInfo

```
GET /paystackmock/customer/{email}
```

Parameters

Type	Name	Description	Schema
Path	**email** *required*	email	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
202	Accepted	PayStackCustomer

HTTP Code	Description	Schema
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackAPIMock

createPlan

```
POST /paystackmock/plan
```

Parameters

Type	Name	Description	Schema
Body	**planRequest** *required*	planRequest	PayStackPlanRequest

Responses

HTTP Code	Description	Schema
200	OK	PayStackPlanResponse
201	Created	No Content
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackAPIMock

refundTransaction

```
POST /paystackmock/refund
```

Parameters

Type	Name	Description	Schema
Body	**body** *required*	body	object

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackAPIMock

getSubscribeCreateWebhook

```
GET /paystackmock/subscribecreatewebhook
```

Responses

HTTP Code	Description	Schema
200	OK	PayStackWebhook
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackAPIMock

createSubscription

```
POST /paystackmock/subscription
```

Parameters

Type	Name	Description	Schema
Body	**subscription** *required*	subscription	PayStackSubscriptionRequest

Responses

HTTP Code	Description	Schema
200	OK	PayStackSubscriptionResponse
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackAPIMock

cancelSubscription

```
POST /paystackmock/subscription/disable
```

Parameters

Type	Name	Description	Schema
Body	**subscription** *required*	subscription	JsonObject

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackAPIMock

getSubscriptionVerification

```
GET /paystackmock/subscription/{payStackSubscriptionId}
```

Parameters

Type	Name	Description	Schema
Path	**payStackSubscriptionId** *required*	payStackSubscriptionId	string

Responses

HTTP Code	Description	Schema
200	OK	PayStackSubscriptionVerificationResponse
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackAPIMock

createTransaction

```
POST /paystackmock/transaction/initialize
```

Parameters

Type	Name	Description	Schema
Body	**planRequest** *required*	planRequest	PayStackTransactionRequest

Responses

HTTP Code	Description	Schema
200	OK	PayStackTransactionResponse

HTTP Code	Description	Schema
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- PayStackAPIMock

validateTransaction

```
GET /paystackmock/transaction/validate
```

Responses

HTTP Code	Description	Schema
200	OK	No Content
202	Accepted	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- application/json

Produces

- application/json

Tags

- PayStackAPIMock

verifyTransaction

```
GET /paystackmock/transaction/verify/{reference}
```

Parameters

Type	Name	Description	Schema
Path	**reference** *required*	reference	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
202	Accepted	PayStackVerificationResponse
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- application/json

Produces

- `application/json`

Tags

- PayStackAPIMock

refundPayStackTransaction

```
POST /refund/{reference}
```

Parameters

Type	Name	Description	Schema
Path	**reference** *required*	reference	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- BackofficeController

getOwnSubscription

```
GET /subscription
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	ShowOwnSubscriptionDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SubscriptionController

getSubscription

```
GET /subscription/all
```

Responses

HTTP Code	Description	Schema
200	OK	SubscriptionDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SubscriptionController

createAnnualSubscription

```
POST /subscription/annual
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Query	**service** *required*	service	string
Body	**annualSubscriptionSetDto** *required*	annualSubscriptionSetDto	AnnualSubscriptionSetDto

Responses

HTTP Code	Description	Schema
201	Created	AnnualSubscriptionDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- BackofficeController

cancelSubscription

```
PUT /subscription/annual/cancel
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Query	**service** *required*	service	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SubscriptionController

extendSubscription

```
POST /subscription/annual/extend/sub
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content

HTTP Code	Description	Schema
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SubscriptionController

deleteAnnualSubscription

```
DELETE /subscription/annual/{id}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**id** *required*	id	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content

HTTP Code	Description	Schema
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- BackofficeController

updateAnnualSubscriptionPrice

```
PUT /subscription/annual/{id}/price
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**id** *required*	id	integer (int64)
Body	**priceDto** *required*	priceDto	PriceDto

Responses

HTTP Code	Description	Schema
200	OK	AnnualSubscriptionDto

HTTP Code	Description	Schema
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- BackofficeController

deactivateAnnualSubscription

```
PUT /subscription/annual/{subsAnnualId}/deactivate
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**subsAnnualId** *required*	subsAnnualId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content

HTTP Code	Description	Schema
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- BackofficeController

activateSubscription

```
POST /subscription/annual/{subscription_id}/activate
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**subscription_id** *required*	subscription_id	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content

HTTP Code	Description	Schema
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SubscriptionController

subscribeAnnualSubscription

```
POST /subscription/annual/{subscription_id}/subscribe
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**subscription_id** *required*	subscription_id	integer (int64)
Query	**service** *required*	service	string
Body	**urls** *optional*	urls	PaypalContextDto

Responses

HTTP Code	Description	Schema
200	OK	AnnualSubscriptionDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SubscriptionController

createTokenSubscription

```
POST /subscription/token
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Body	**tokenSubscriptionSetDto** *required*	tokenSubscriptionSetDto	TokenSubscriptionSetDto

Responses

HTTP Code	Description	Schema
201	Created	TokenSubscriptionDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- BackofficeController

updateTokenSubscription

```
PUT /subscription/token/{subsTokenId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**subsTokenId** *required*	subsTokenId	integer (int64)
Body	**tokenSubscriptionSetDto** *required*	tokenSubscriptionSetDto	TokenSubscriptionSetDto

Responses

HTTP Code	Description	Schema
200	OK	TokenSubscriptionDto
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- BackofficeController

deleteTokenSubscription

```
DELETE /subscription/token/{subsTokenId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**subsTokenId** *required*	subsTokenId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	No Content
204	No Content	No Content
401	Unauthorized	No Content
403	Forbidden	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- BackofficeController

receiveWebhook

```
POST /subscription/webhook
```

Parameters

Type	Name	Description	Schema
Body	**body** *required*	body	Webhook

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- SubscriptionController

receivePayStackWebhook

```
POST /subscription/webhook/paystack
```

Parameters

Type	Name	Description	Schema
Body	**body** *required*	body	PayStackWebhook

Responses

HTTP Code	Description	Schema
200	OK	No Content
201	Created	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- application/json

Produces

- application/json

Tags

- SubscriptionController

getUserSubscriptionData

```
GET /user/subscription/{userId}
```

Parameters

Type	Name	Description	Schema
Path	**userId** *required*	userId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	ShowOwnSubscriptionDto
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- application/json

Produces

- application/json

Tags

- UserSubscriptionController

Apêndice I

Especificações sobre o MicroContactRequest

Neste anexo encontram-se os diagramas de classes do MicroContactRequest, gerados usando UML Lab [106]; e, especificações sobre a API REST extraída a partir da OpenAPI [107] usando Swagger2Markup [108] e Asciidoctor PDF [109].

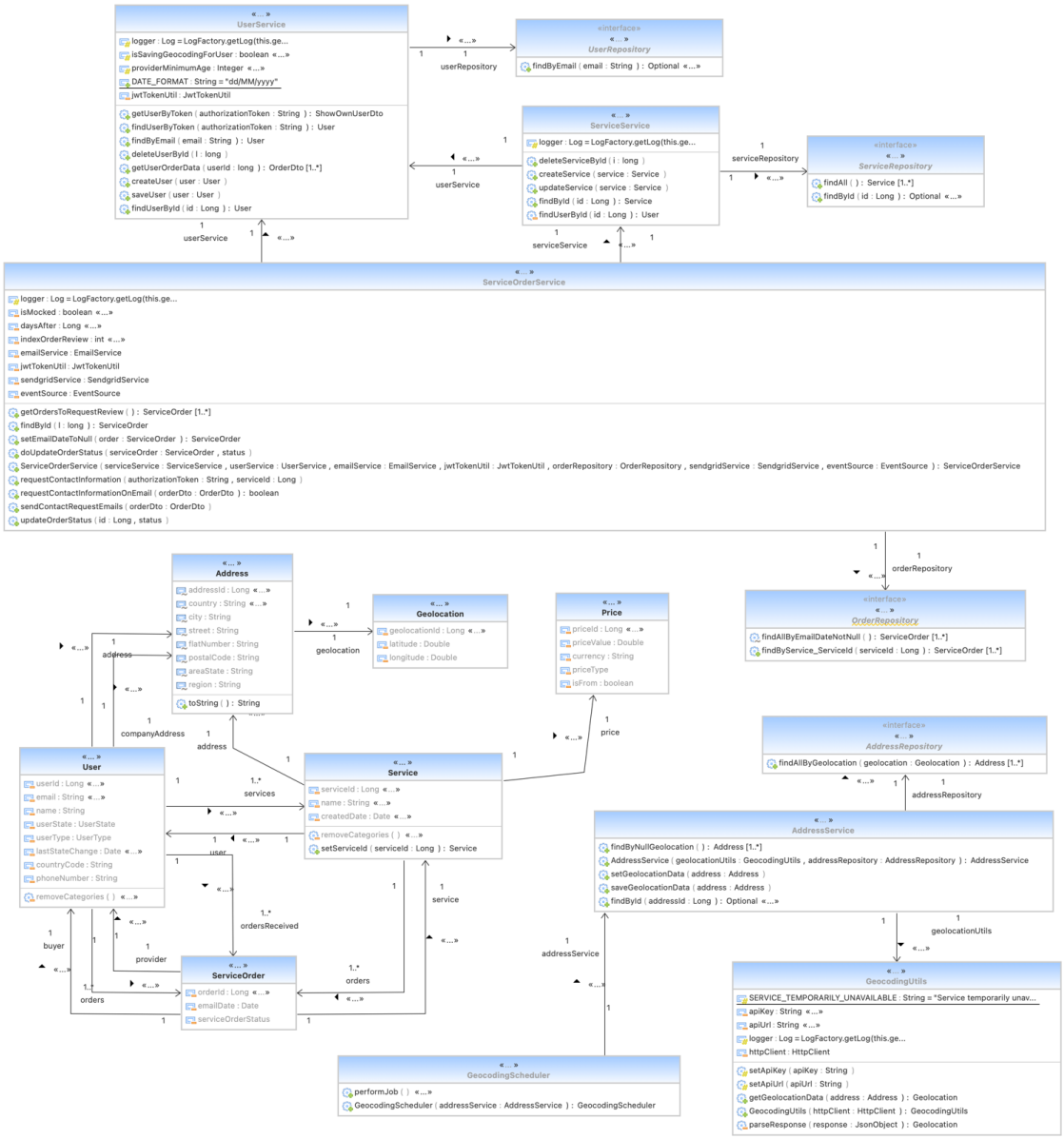


Figura I.1: Modelo de domínio implementado no MicroContactRequest.

MicroContatRequest REST API

Overview

Documentation for MicroContatRequest API

Version information

Version : 1.0.0

URI scheme

Host : localhost:8085
BasePath : /api

Tags

- GeocodingAPIMock : Geocoding API Mock
- OrderController : Service Order Controller
- UserOrderController : User Controller

Produces

- `application/json`

Paths

getUserServicesFromToken

```
GET /geocodemock
```

Parameters

Type	Name	Description	Schema
Query	**address** *required*	address	string
Query	**key** *required*	key	string

Responses

HTTP Code	Description	Schema
200	OK	string
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- GeocodingAPIMock

getServiceContactsOnEmail

```
POST /order/contacts/{serviceId}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	Authorization	string
Path	**serviceId** *required*	serviceId	integer (int64)

Responses

HTTP Code	Description	Schema
201	Created	No Content

HTTP Code	Description	Schema
202	Accepted	No Content
401	Unauthorized	No Content
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- OrderController

getUserOrderData

```
GET /user/order/{userId}
```

Parameters

Type	Name	Description	Schema
Path	**userId** *required*	userId	integer (int64)

Responses

HTTP Code	Description	Schema
200	OK	< OrderDto > array
401	Unauthorized	No Content

HTTP Code	Description	Schema
403	Forbidden	No Content
404	Not Found	No Content

Consumes

- `application/json`

Produces

- `application/json`

Tags

- UserOrderController

Apêndice J

Relatório completo CodeMR sobre a complexidade do código

Neste anexo encontram-se os dados dos relatórios obtidos usando a ferramenta CodeMR para todos os projetos, que suportam os resultados apresentados no Capítulo 8.

MARKETPLACE - List of all classes (#323)						
ID	CLASS	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	UserService	475	**very-high**	**very-high**	high	**very-high**
2	BackofficeController	218	**very-high**	**very-high**	**very-high**	**very-high**
3	AnnualSubscriptio...	354	high	**very-high**	high	medium-high
4	ServiceService	242	medium-high	**very-high**	high	low-medium
5	PlatformReviewSer...	141	low-medium	**very-high**	high	low-medium
6	PaypalService	223	low-medium	high	medium-high	low-medium
7	RegistrationService	198	low-medium	high	high	low-medium
8	PaypalOrderTokenS...	137	low-medium	high	medium-high	low-medium
9	ServiceReviewService	115	low-medium	high	high	low-medium
10	BuyerReviewService	98	low-medium	high	medium-high	low-medium
11	SendgridService	131	low	high	low	low-medium
12	HttpClient	74	low	high	low	low-medium
13	AdminUserController	49	high	medium-high	low-medium	low
14	ServiceOrderService	92	medium-high	medium-high	low-medium	low-medium
15	CategoryService	120	low-medium	medium-high	medium-high	low-medium
16	PayStackService	109	low-medium	medium-high	medium-high	low-medium
17	TokenSubscription...	108	low-medium	medium-high	medium-high	low-medium
18	WebSecurityConfig	65	low-medium	medium-high	low-medium	low-medium
19	LocationService	45	low-medium	medium-high	low	low
20	GeocodingUtils	44	low-medium	medium-high	low	low
21	PayStackAPIMock	341	low	medium-high	medium-high	medium-high
22	PayPalAPIMock	104	low	medium-high	medium-high	low-medium
23	BackofficeUserSer...	59	low	medium-high	low-medium	low-medium
24	JwtAuthentication...	30	low	medium-high	low	low
25	SubscriptionContr...	32	medium-high	low-medium	medium-high	low
26	ReviewController	70	low-medium	low-medium	medium-high	low-medium
27	SearchController	48	low-medium	low-medium	low-medium	low
28	JwtRequestFilter	35	low-medium	low-medium	low	low
29	PayPalHttpClientMock	30	low-medium	low-medium	low	low
30	RegistrationContr...	28	low-medium	low-medium	medium-high	low
31	Malformed*query*Par...	9	low-medium	low-medium	low	low
32	CategoryException...	9	low-medium	low-medium	low	low
33	ExceptionHandling	9	low-medium	low-medium	low	low
34	IOExceptionExcept...	9	low-medium	low-medium	low	low
35	ServiceUnavailabl...	9	low-medium	low-medium	low	low
36	LoginExceptionHan...	9	low-medium	low-medium	low	low
37	UserNotActiveExce...	9	low-medium	low-medium	low	low
38	WriterExceptionEx...	9	low-medium	low-medium	low	low
39	LoginFailedExcept...	9	low-medium	low-medium	low	low
40	ItemExpiredExcept...	9	low-medium	low-medium	low	low
41	User	122	low	low-medium	low	low-medium

42	ShowFullUserDto	74	low	low-medium	low	low-medium
43	Service	55	low	low-medium	low	low-medium
44	ReviewQuestionSer...	51	low	low-medium	low-medium	low-medium
45	EmailService	50	low	low-medium	low	low
46	AnnualSubscriptio...	50	low	low-medium	low	low
47	PlatformReviewEve...	43	low	low-medium	low-medium	low
48	GoogleAuthController	39	low	low-medium	low	low
49	AmazonClient	37	low	low-medium	low	low
50	JwtTokenUtil	31	low	low-medium	low	low
51	PlatformContentSe...	31	low	low-medium	low	low
52	JwtUserDetailsSer...	30	low	low-medium	low	low
53	GoogleAuthService	28	low	low-medium	low	low
54	ShowFullServiceDto	28	low	low-medium	low	low
55	ServiceShowOwnDto	28	low	low-medium	low	low
56	SwaggerConfig	20	low	low-medium	low	low
57	CorsFilterConfigu...	18	low	low-medium	low	low
58	AdminServiceContr...	35	medium-high	low	low	low
59	PayStackAPIException	10	medium-high	low	low	low
60	ParseResponseExce...	5	medium-high	low	low	low
61	Malformed*query*Par...	4	medium-high	low	low	low
62	CategoryException	4	medium-high	low	low	low
63	ReviewException	4	medium-high	low	low	low
64	UserException	4	medium-high	low	low	low
65	LoginFailedException	4	medium-high	low	low	low
66	LoginUserException	4	medium-high	low	low	low
67	RegisterUserExcep...	4	medium-high	low	low	low
68	ServiceUnavailabl...	4	medium-high	low	low	low
69	UserNotActiveExce...	4	medium-high	low	low	low
70	ItemExpiredException	4	medium-high	low	low	low
71	ProductCategory	447	low-medium	low	low	medium-high
72	SearchService	162	low-medium	low	low-medium	low-medium
73	PayPalErrorEnum	27	low-medium	low	low	low
74	PaypalController	25	low-medium	low	low	low
75	PlatformReviewCon...	20	low-medium	low	low	low
76	LocationController	19	low-medium	low	low	low
77	PayStackController	18	low-medium	low	low	low
78	AddressService	16	low-medium	low	low	low
79	AnnualSubscription	14	low-medium	low	low	low
80	PayStackWebhookEv...	14	low-medium	low	low	low
81	GeocodingStatusCodes	13	low-medium	low	low	low
82	HttpException	13	low-medium	low	low	low
83	TokenSubscription	12	low-medium	low	low	low
84	ServiceReview	11	low-medium	low	low	low
85	ServiceOrderContr...	11	low-medium	low	low	low
86	SearchSort	10	low-medium	low	low	low
87	BuyerReview	9	low-medium	low	low	low
88	PayStackAPIExcept...	9	low-medium	low	low	low
89	UserType	7	low-medium	low	low	low
90	PayStackVerificat...	7	low-medium	low	low	low
91	PayStackTransacti...	7	low-medium	low	low	low
92	PayStackSubscript...	7	low-medium	low	low	low
93	PayStackCustomer	7	low-medium	low	low	low
94	PayStackPlanResponse	7	low-medium	low	low	low
95	PayStackSubscript...	7	low-medium	low	low	low
96	PlatformReview	6	low-medium	low	low	low
97	PaymentFrequency	6	low-medium	low	low	low
98	UserState	6	low-medium	low	low	low
99	RegisterUserExcep...	6	low-medium	low	low	low
100	ModifiedPlatformR...	6	low-medium	low	low	low
101	ReviewType	5	low-medium	low	low	low
102	ServiceState	5	low-medium	low	low	low
103	IntervalUnit	5	low-medium	low	low	low
104	HttpResponseMock	5	low-medium	low	low	low
105	ReviewState	4	low-medium	low	low	low
106	PriceType	4	low-medium	low	low	low
107	ProductType	4	low-medium	low	low	low
108	PlatformReviewsEv...	3	low-medium	low	low	low

109	DeletedPlatformRe...	3	low-medium	low	low	low
110	NewPlatformReview...	3	low-medium	low	low	low
111	ExternalPaymentSe...	3	low-medium	low	low	low
112	QuestionState	2	low-medium	low	low	low
113	AnnualSubscriptio...	2	low-medium	low	low	low
114	PayPalSubscriptio...	2	low-medium	low	low	low
115	Status	2	low-medium	low	low	low
116	SetupFeeFailureAc...	2	low-medium	low	low	low
117	TenureType	2	low-medium	low	low	low
118	ProviderShowDto	92	low	low	low	low-medium
119	ReviewUtils	65	low	low	low	low-medium
120	PayPalScheduler	61	low	low	low-medium	low-medium
121	PayStackWebhookData	52	low	low	low	low-medium
122	CredentialRepository	50	low	low	low	low
123	LoginUser	47	low	low	low	low
124	PayStackSubscript...	46	low	low	low	low
125	PayStackSubscript...	46	low	low	low	low
126	PhoneDataValidator	45	low	low	low	low
127	PayPalClient	41	low	low	low	low
128	ProviderDto	38	low	low	low	low
129	ServiceRepository	37	low	low	medium-high	low
130	Address	37	low	low	low	low
131	PayStackSubscript...	34	low	low	low	low
132	PayStackCustomerData	33	low	low	low	low
133	AddressDto	32	low	low	low	low
134	PayStackVerificat...	31	low	low	low	low
135	BusinessHoursDto	30	low	low	low	low
136	SearchResultDto	30	low	low	low	low
137	ShowOwnUserDto	30	low	low	low	low
138	DeferredLiquibase...	30	low	low	low	low
139	ReviewShowDto	30	low	low	low	low
140	MPUserDetails	29	low	low	low	low
141	PayStackPlanData	29	low	low	low	low
142	WishListController	28	low	low	low	low
143	ServiceSetDto	27	low	low	low	low
144	PayStackScheduler	27	low	low	low	low
145	ReviewScheduler	27	low	low	low	low
146	PayStackCustomerS...	27	low	low	low	low
147	ServiceUtils	26	low	low	low	low
148	Category	26	low	low	low	low
149	PlatformReviewSho...	26	low	low	low	low
150	Review	25	low	low	low	low
151	PayStackCustomerA...	25	low	low	low	low
152	PaymentOrder	25	low	low	low	low
153	ServiceShowBackof...	24	low	low	low	low
154	ServiceShowDto	24	low	low	low	low
155	PlatformRatingSer...	24	low	low	low	low
156	PayStackSubscript...	24	low	low	low	low
157	PlanRequest	24	low	low	low	low
158	UserRepository	23	low	low	medium-high	low
159	PayPalSubscription	23	low	low	low	low
160	ShowPublicUserDto	23	low	low	low	low
161	GeocodingAPIMock	22	low	low	low	low
162	GeocodingData	22	low	low	low	low
163	PaypalConfig	22	low	low	low	low
164	PriceDto	21	low	low	low	low
165	PayStackWebhookDa...	21	low	low	low	low
166	AddressDataValidator	20	low	low	low	low
167	AutocompleteDto	20	low	low	low	low
168	ReviewQuestion	19	low	low	low	low
169	TokenSubscriptionDto	19	low	low	low	low
170	UserController	19	low	low	low-medium	low
171	ValidatorExceptio...	19	low	low	low	low
172	ServiceOrder	18	low	low	low	low
173	CategoryShowDto	18	low	low	low	low
174	BusinessHours	18	low	low	low	low
175	Action	18	low	low	low	low

176	State	18	low	low	low	low
177	City	18	low	low	low	low
178	ReviewAnswerSetDto	18	low	low	low	low
179	BillingCycle	18	low	low	low	low
180	PayPalApplication...	17	low	low	low	low
181	ShowUserDto	17	low	low	low	low
182	ProviderEditDto	17	low	low	low	low
183	AmazonClientConfig	17	low	low	low	low
184	ReviewQuestionSetDto	17	low	low	low	low
185	PayPalProduct	17	low	low	low	low
186	ShowLoginUserDto	16	low	low	low	low
187	ServiceReviewsSum...	16	low	low	low	low
188	PayStackVerificat...	16	low	low	low	low
189	PayStackVerificat...	16	low	low	low	low
190	CategoryController	16	low	low	low	low
191	AnnualSubscriptio...	15	low	low	low	low
192	UserSetDto	15	low	low	low	low
193	Price	15	low	low	low	low
194	Country	15	low	low	low	low
195	ReviewAnswer	14	low	low	low	low
196	PlatformContent	14	low	low	low	low
197	OrderDto	14	low	low	low	low
198	ReviewQuestionEdi...	14	low	low	low	low
199	PayPalError	14	low	low	low	low
200	MarketplaceApplic...	14	low	low	low	low
201	ActionService	14	low	low	low	low
202	DeletionReason	13	low	low	low	low
203	SubscriptionShowDto	13	low	low	low	low
204	Geolocation	13	low	low	low	low
205	EmailContactDto	13	low	low	low	low
206	SetUserDto	13	low	low	low	low
207	UploadController	13	low	low	low	low
208	Subscription	13	low	low	low	low
209	ServiceController	13	low	low	low	low
210	ReviewAnswerEditDto	13	low	low	low	low
211	ReviewAnswerShowDto	13	low	low	low	low
212	PlatformReviewSetDto	13	low	low	low	low
213	Detail	13	low	low	low	low
214	ShowCountryDto	13	low	low	low	low
215	PlatformRating	12	low	low	low	low
216	RequestInfoBuyerE...	12	low	low	low	low
217	CategorySetDto	12	low	low	low	low
218	ReviewUserDto	12	low	low	low	low
219	PayPalConstants	12	low	low	low	low
220	PaymentPreferences	12	low	low	low	low
221	DeletionReasonSer...	11	low	low	low	low
222	GeocodingConstants	11	low	low	low	low
223	ObjectInfoDto	11	low	low	low	low
224	Webhook	11	low	low	low	low
225	PayPalLink	11	low	low	low	low
226	VerifyPhoneCodeDto	11	low	low	low	low
227	ShowOwnSubscripti...	11	low	low	low	low
228	PayStackTransacti...	11	low	low	low	low
229	PlatformContentDto	11	low	low	low	low
230	ReviewEditDto	11	low	low	low	low
231	ReviewSetDto	11	low	low	low	low
232	SubscriptionDto	10	low	low	low	low
233	SetUserWithTypeDto	10	low	low	low	low
234	PlatformContentCo...	10	low	low	low	low
235	JwtResponse	10	low	low	low	low
236	PayStackTransacti...	10	low	low	low	low
237	ShowWishlistItemDto	10	low	low	low	low
238	ParametersValidation	10	low	low	low	low
239	ReviewQuestionSho...	10	low	low	low	low
240	PlatformReviewEdi...	10	low	low	low	low
241	PayPalPrice	10	low	low	low	low
242	GoogleAuthConfig	10	low	low	low	low

243	TokenSubscription...	9	low	low	low	low
244	ResetPasswordDto	9	low	low	low	low
245	RegisterEmailDto	9	low	low	low	low
246	SubscriptionExpir...	9	low	low	low	low
247	RecoverPasswordEm...	9	low	low	low	low
248	ServicesByCategor...	9	low	low	low	low
249	WebhookDto	9	low	low	low	low
250	GeocodingScheduler	9	low	low	low	low
251	PaymentMethod	9	low	low	low	low
252	SendPhoneCodeDto	9	low	low	low	low
253	DeletionReasonDto	9	low	low	low	low
254	CountryRegexConfig	9	low	low	low	low
255	PayStackCustomerI...	9	low	low	low	low
256	PayStackResponse	9	low	low	low	low
257	PayStackVerificat...	9	low	low	low	low
258	PayStackPlanRequest	9	low	low	low	low
259	PlatformReviewSum...	9	low	low	low	low
260	SetStateDto	9	low	low	low	low
261	LocationDto	9	low	low	low	low
262	CategoryRepository	8	low	low	low	low
263	PriceSubscriptionDto	8	low	low	low	low
264	PasswordChangeDto	8	low	low	low	low
265	RecoverPasswordDto	8	low	low	low	low
266	AuthenticationDto	8	low	low	low	low
267	AddressDataConstr...	8	low	low	low	low
268	RequestReviewUser...	8	low	low	low	low
269	RequestInfoProvid...	8	low	low	low	low
270	PlatformReviewEma...	8	low	low	low	low
271	WelcomeProviderEm...	8	low	low	low	low
272	DaySubscriptionEx...	8	low	low	low	low
273	ProviderVerifiedE...	8	low	low	low	low
274	WelcomeEmailDto	8	low	low	low	low
275	SearchResultCateg...	8	low	low	low	low
276	CategorySearchDto	8	low	low	low	low
277	ServiceFilterShowDto	8	low	low	low	low
278	SearchDto	8	low	low	low	low
279	Subscriber	8	low	low	low	low
280	UserFilterDto	8	low	low	low	low
281	JwtRequest	8	low	low	low	low
282	PlatformReviewRep...	8	low	low	low-medium	low
283	PayStackWebhook	8	low	low	low	low
284	PayStackSubscript...	8	low	low	low	low
285	PaypalContextDto	8	low	low	low	low
286	Taxes	8	low	low	low	low
287	AuthService	7	low	low	low	low
288	ValidateKeyDto	7	low	low	low	low
289	ScratchCodesDto	7	low	low	low	low
290	ValidationDto	7	low	low	low	low
291	PhoneDataConstraint	7	low	low	low	low
292	DeleteAccountEmai...	7	low	low	low	low
293	EventType	7	low	low	low	low
294	VerifyUserDto	7	low	low	low	low
295	ContactInfoDto	7	low	low	low	low
296	ServiceReviewRepo...	7	low	low	low	low
297	BusinessReviewEdi...	7	low	low	low	low
298	Frequency	7	low	low	low	low
299	ExceptionResponse	6	low	low	low	low
300	EventBusConfig	6	low	low	low	low
301	Resource	6	low	low	low	low
302	SubscriberDto	6	low	low	low	low
303	PaymentOrderRepos...	6	low	low	low	low
304	BuyerReviewReposi...	6	low	low	low	low
305	ShowOrderStatusDto	6	low	low	low	low
306	ShowOrderLinkDto	6	low	low	low	low
307	JwtAuthentication...	6	low	low	low	low
308	ShowUrlDto	6	low	low	low	low
309	OrderDetailsShowDto	5	low	low	low	low

310	PricingScheme	5	low	low	low	low
311	ReviewEvent	5	low	low	low	low
312	AutocompleteResul...	4	low	low	low	low
313	PlatformContentRe...	4	low	low	low	low
314	ReviewQuestionRep...	4	low	low	low	low
315	LoginUserRepository	4	low	low	low	low
316	DeletionReasonRep...	3	low	low	low	low
317	CountryRepository	3	low	low	low	low
318	OrderRepository	3	low	low	low	low
319	AddressRepository	3	low	low	low	low
320	ActionRepository	2	low	low	low	low
321	PlatformRatingRep...	2	low	low	low	low
322	AnnualSubscriptio...	2	low	low	low	low
323	TokenSubscription...	2	low	low	low	low

Tabela J.1: Relatório relativo ao monólito.

MICRO PLATFORM CONTENT - List of all classes (#48)						
ID	CLASS	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	HttpClient	74	low	high	low	low-medium
2	WebSecurityConfig	65	low-medium	medium-high	low-medium	low-medium
3	LocationService	45	low-medium	medium-high	low	low
4	KafkaConfig	52	low	medium-high	low	low-medium
5	JwtRequestFilter	35	low-medium	low-medium	low	low
6	JwtTokenUtil	31	low	low-medium	low	low
7	JwtUserDetailsSer...	23	low	low-medium	low	low
8	SwaggerConfig	20	low	low-medium	low	low
9	CorsFilterConfigu...	18	low	low-medium	low	low
10	ParseResponseExce...	5	medium-high	low	low	low
11	LocationController	19	low-medium	low	low	low
12	HttpException	13	low-medium	low	low	low
13	UserType	7	low-medium	low	low	low
14	UserState	6	low-medium	low	low	low
15	MPUserDetails	32	low	low	low-medium	low
16	DeferredLiquibase...	30	low	low	low	low
17	PlatformContentSe...	21	low	low	low	low
18	State	18	low	low	low	low
19	City	18	low	low	low	low
20	EventSink	18	low	low	low	low
21	Country	15	low	low	low	low
22	User	15	low	low	low	low
23	UserService	14	low	low	low	low
24	PlatformContent	14	low	low	low	low
25	MarketplaceApplic...	14	low	low	low	low
26	Geolocation	13	low	low	low	low
27	ShowCountryDto	13	low	low	low	low
28	PlatformContentDto	11	low	low	low	low
29	PlatformContentCo...	10	low	low	low	low
30	CountryRegexConfig	9	low	low	low	low
31	EventConstants	9	low	low	low	low
32	SetStateDto	9	low	low	low	low
33	LocationDto	9	low	low	low	low
34	EventSource	6	low	low	low	low
35	ExceptionResponse	6	low	low	low	low
36	StateBuilder	4	low	low	low	low
37	CityBuilder	4	low	low	low	low
38	UserBuilder	4	low	low	low	low
39	PlatformContentRe...	4	low	low	low	low
40	GeolocationBuilder	3	low	low	low	low
41	CountryBuilder	3	low	low	low	low
42	LocationDtoBuilder	3	low	low	low	low
43	ShowCountryDtoBui...	3	low	low	low	low
44	SetStateDtoBuilder	3	low	low	low	low
45	PlatformContentBu...	3	low	low	low	low

46	CountryRepository	3	low	low	low	low
47	UserRepository	3	low	low	low	low
48	PlatformContentDt...	2	low	low	low	low

Tabela J.2: Relatório relativo ao MicroPlatformContent.

MICRO SUBSCRIPTION - List of all classes (#213)						
ID	CLASS	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	AnnualSubscriptio...	349	high	**very-high**	high	medium-high
2	PaypalService	223	low-medium	high	medium-high	low-medium
3	PaypalOrderTokenS...	137	low-medium	high	medium-high	low-medium
4	HttpClient	74	low	high	low	low-medium
5	UserService	182	medium-high	medium-high	high	low-medium
6	PayStackService	109	low-medium	medium-high	medium-high	low-medium
7	TokenSubscription...	103	low-medium	medium-high	low-medium	low-medium
8	WebSecurityConfig	65	low-medium	medium-high	low-medium	low-medium
9	PayStackAPIMock	341	low	medium-high	medium-high	medium-high
10	PayPalAPIMock	104	low	medium-high	medium-high	low-medium
11	SendgridService	53	low	medium-high	low	low-medium
12	KafkaConfig	52	low	medium-high	low	low-medium
13	SubscriptionContr...	37	medium-high	low-medium	medium-high	low
14	PaymentController	40	low-medium	low-medium	low-medium	low
15	JwtRequestFilter	35	low-medium	low-medium	low	low
16	PayPalHttpClientMock	30	low-medium	low-medium	low	low
17	ExceptionHandling	9	low-medium	low-medium	low	low
18	AnnualSubscriptio...	50	low	low-medium	low	low
19	JwtTokenUtil	31	low	low-medium	low	low
20	JwtUserDetailsSer...	23	low	low-medium	low	low
21	SwaggerConfig	20	low	low-medium	low	low
22	CorsFilterConfigu...	18	low	low-medium	low	low
23	PayStackAPIException	10	medium-high	low	low	low
24	ParseResponseExce...	5	medium-high	low	low	low
25	UserException	4	medium-high	low	low	low
26	Malformed*query*Par...	4	medium-high	low	low	low
27	ProductCategory	447	low-medium	low	low	medium-high
28	PayPalErrorEnum	27	low-medium	low	low	low
29	EventSink	26	low-medium	low	low	low
30	PaypalController	25	low-medium	low	low	low
31	PayStackController	18	low-medium	low	low	low
32	AnnualSubscription	14	low-medium	low	low	low
33	PayStackWebhookEv...	14	low-medium	low	low	low
34	HttpException	13	low-medium	low	low	low
35	TokenSubscription	12	low-medium	low	low	low
36	UserController	10	low-medium	low	low	low
37	Malformed*query*Par...	9	low-medium	low	low	low
38	PayStackAPIExcept...	9	low-medium	low	low	low
39	UserType	7	low-medium	low	low	low
40	PayStackVerificat...	7	low-medium	low	low	low
41	PayStackTransacti...	7	low-medium	low	low	low
42	PayStackSubscript...	7	low-medium	low	low	low
43	PayStackCustomer	7	low-medium	low	low	low
44	PayStackPlanResponse	7	low-medium	low	low	low
45	PayStackSubscript...	7	low-medium	low	low	low
46	PaymentFrequency	6	low-medium	low	low	low
47	UserState	6	low-medium	low	low	low
48	IntervalUnit	5	low-medium	low	low	low
49	HttpResponseMock	5	low-medium	low	low	low
50	PriceType	4	low-medium	low	low	low
51	ProductType	4	low-medium	low	low	low
52	ExternalPaymentSe...	3	low-medium	low	low	low
53	AnnualSubscriptio...	2	low-medium	low	low	low
54	ServiceOrderStatus	2	low-medium	low	low	low
55	PayPalSubscriptio...	2	low-medium	low	low	low
56	Status	2	low-medium	low	low	low

57	SetupFeeFailureAc...	2	low-medium	low	low	low
58	TenureType	2	low-medium	low	low	low
59	PayPalScheduler	61	low	low	low-medium	low-medium
60	PayStackWebhookData	52	low	low	low	low-medium
61	PayStackSubscript...	46	low	low	low	low
62	PayStackSubscript...	46	low	low	low	low
63	PayPalClient	41	low	low	low	low
64	PayStackSubscript...	34	low	low	low	low
65	PayStackCustomerData	33	low	low	low	low
66	MPUserDetails	32	low	low	low-medium	low
67	User	31	low	low	low	low
68	PayStackVerificat...	31	low	low	low	low
69	DeferredLiquibase...	30	low	low	low	low
70	PayStackPlanData	29	low	low	low	low
71	PayStackCustomerS...	27	low	low	low	low
72	PayStackScheduler	27	low	low	low	low
73	PayStackCustomerA...	25	low	low	low	low
74	PayStackWebhookDa...	25	low	low	low	low
75	PaymentOrder	25	low	low	low	low
76	PlanRequest	24	low	low	low	low
77	EmailService	24	low	low	low	low
78	PayStackSubscript...	24	low	low	low	low
79	PayPalSubscription	23	low	low	low	low
80	PayStackSubscript...	22	low	low	low	low
81	PaypalConfig	22	low	low	low	low
82	PriceDto	21	low	low	low	low
83	PayStackWebhookDa...	21	low	low	low	low
84	PayStackCustomerD...	20	low	low	low	low
85	PayStackSubscript...	20	low	low	low	low
86	ValidatorExceptio...	19	low	low	low	low
87	TokenSubscriptionDto	19	low	low	low	low
88	PayStackVerificat...	19	low	low	low	low
89	BillingCycle	18	low	low	low	low
90	PayStackSubscript...	18	low	low	low	low
91	PayPalProduct	17	low	low	low	low
92	EventConstants	17	low	low	low	low
93	PayPalApplication...	17	low	low	low	low
94	PayStackVerificat...	16	low	low	low	low
95	PayStackVerificat...	16	low	low	low	low
96	Price	15	low	low	low	low
97	UserBuilder	15	low	low	low	low
98	AnnualSubscriptio...	15	low	low	low	low
99	PayPalError	14	low	low	low	low
100	MarketplaceApplic...	14	low	low	low	low
101	PayStackPlanDataB...	14	low	low	low	low
102	PaymentOrderBuilder	14	low	low	low	low
103	Detail	13	low	low	low	low
104	Subscription	13	low	low	low	low
105	SubscriptionShowDto	13	low	low	low	low
106	PayStackCustomerA...	13	low	low	low	low
107	PayPalConstants	12	low	low	low	low
108	PaymentPreferences	12	low	low	low	low
109	EventSource	12	low	low	low	low
110	Webhook	11	low	low	low	low
111	ShowOwnSubscripti...	11	low	low	low	low
112	PayStackTransacti...	11	low	low	low	low
113	PayStackCustomerS...	11	low	low	low	low
114	PayPalLink	11	low	low	low	low
115	ParametersValidation	10	low	low	low	low
116	PayPalPrice	10	low	low	low	low
117	SubscriptionDto	10	low	low	low	low
118	OrderDto	10	low	low	low	low
119	PayStackTransacti...	10	low	low	low	low
120	WebhookDto	9	low	low	low	low
121	CountryRegexConfig	9	low	low	low	low
122	TokenSubscription...	9	low	low	low	low
123	PayStackCustomerI...	9	low	low	low	low

124	PayStackResponse	9	low	low	low	low
125	PayStackVerificat...	9	low	low	low	low
126	PayStackSubscript...	9	low	low	low	low
127	PayStackPlanRequest	9	low	low	low	low
128	PayStackWebhookDa...	9	low	low	low	low
129	PaymentMethod	9	low	low	low	low
130	SubscriptionExpir...	9	low	low	low	low
131	PlanRequestBuilder	8	low	low	low	low
132	PaypalContextDto	8	low	low	low	low
133	Taxes	8	low	low	low	low
134	AnnualSubscriptio...	8	low	low	low	low
135	PriceSubscriptionDto	8	low	low	low	low
136	UserRepository	8	low	low	low	low
137	PayStackVerificat...	8	low	low	low	low
138	PayStackWebhook	8	low	low	low	low
139	PayStackSubscript...	8	low	low	low	low
140	Subscriber	8	low	low	low	low
141	DaySubscriptionEx...	8	low	low	low	low
142	EventType	7	low	low	low	low
143	Frequency	7	low	low	low	low
144	PayPalProductBuilder	7	low	low	low	low
145	PayPalSubscriptio...	7	low	low	low	low
146	Resource	6	low	low	low	low
147	SubscriberDto	6	low	low	low	low
148	PaymentOrderRepos...	6	low	low	low	low
149	ExceptionResponse	6	low	low	low	low
150	ShowOrderStatusDto	6	low	low	low	low
151	ShowOrderLinkDto	6	low	low	low	low
152	AnnualSubscriptio...	6	low	low	low	low
153	TokenSubscription...	6	low	low	low	low
154	PayStackVerificat...	6	low	low	low	low
155	ShowOwnSubscripti...	5	low	low	low	low
156	PayPalErrorBuilder	5	low	low	low	low
157	BillingCycleBuilder	5	low	low	low	low
158	OrderDetailsShowDto	5	low	low	low	low
159	DetailBuilder	5	low	low	low	low
160	PricingScheme	5	low	low	low	low
161	PriceBuilder	5	low	low	low	low
162	AnnualSubscriptio...	5	low	low	low	low
163	OrderDtoBuilder	5	low	low	low	low
164	WebhookBuilder	4	low	low	low	low
165	PaymentPreference...	4	low	low	low	low
166	TokenSubscription...	4	low	low	low	low
167	PriceDtoBuilder	4	low	low	low	low
168	PayPalApplication...	4	low	low	low	low
169	PaypalContextDtoB...	3	low	low	low	low
170	TokenSubscription...	3	low	low	low	low
171	SubscriptionBuilder	3	low	low	low	low
172	SubscriptionShowD...	3	low	low	low	low
173	PayStackVerificat...	3	low	low	low	low
174	PayStackTransacti...	3	low	low	low	low
175	PayStackPlanReque...	3	low	low	low	low
176	PayStackCustomerI...	3	low	low	low	low
177	PayStackTransacti...	3	low	low	low	low
178	PayPalLinkBuilder	3	low	low	low	low
179	SubscriptionExpir...	3	low	low	low	low
180	WebhookDtoBuilder	2	low	low	low	low
181	ResourceBuilder	2	low	low	low	low
182	FrequencyBuilder	2	low	low	low	low
183	OrderDetailsShowD...	2	low	low	low	low
184	PayPalPriceBuilder	2	low	low	low	low
185	TaxesBuilder	2	low	low	low	low
186	AnnualSubscriptio...	2	low	low	low	low
187	TokenSubscription...	2	low	low	low	low
188	SubscriptionDtoBu...	2	low	low	low	low
189	PayStackSubscript...	2	low	low	low	low
190	PayStackWebhookBu...	2	low	low	low	low

191	PayStackResponseB...	2	low	low	low	low
192	PaymentMethodBuilder	2	low	low	low	low
193	DaySubscriptionEx...	2	low	low	low	low
194	EventTypeBuilder	1	low	low	low	low
195	SubscriberDtoBuilder	1	low	low	low	low
196	ShowOrderLinkDtoB...	1	low	low	low	low
197	ShowOrderStatusDt...	1	low	low	low	low
198	PricingSchemeBuilder	1	low	low	low	low
199	PayStackVerificat...	1	low	low	low	low
200	PayStackSubscript...	1	low	low	low	low
201	PayStackCustomerB...	1	low	low	low	low
202	PayStackTransacti...	1	low	low	low	low
203	PayStackSubscript...	1	low	low	low	low
204	PayStackPlanRespo...	1	low	low	low	low
205	SubscriberBuilder	1	low	low	low	low
206	AnnualSubscriptio...	0	low	low	low	low
207	TokenSubscription...	0	low	low	low	low
208	PayStackSubscript...	0	low	low	low	low
209	PayStackVerificat...	0	low	low	low	low
210	PayStackPlanRespo...	0	low	low	low	low
211	PayStackCustomerB...	0	low	low	low	low
212	PayStackTransacti...	0	low	low	low	low
213	PayStackSubscript...	0	low	low	low	low

Tabela J.3: Relatório relativo ao MicroSubscription.

MICRO REVIEW - List of all classes (#113)						
ID	CLASS	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	PlatformReviewSer...	141	low-medium	**very-high**	high	low-medium
2	ServiceReviewService	115	low-medium	high	high	low-medium
3	BuyerReviewService	98	low-medium	high	medium-high	low-medium
4	HttpClient	74	low	high	low	low-medium
5	ReviewController	99	medium-high	medium-high	high	low-medium
6	WebSecurityConfig	65	low-medium	medium-high	low-medium	low-medium
7	KafkaConfig	52	low	medium-high	low	low-medium
8	SendgridService	45	low	medium-high	low	low
9	JwtRequestFilter	35	low-medium	low-medium	low	low
10	PlatformReviewCon...	32	low-medium	low-medium	medium-high	low
11	ExceptionHandling	9	low-medium	low-medium	low	low
12	UserService	73	low	low-medium	low-medium	low-medium
13	ReviewQuestionSer...	51	low	low-medium	low-medium	low-medium
14	PlatformReviewEve...	43	low	low-medium	low-medium	low
15	ServiceService	41	low	low-medium	low-medium	low
16	JwtTokenUtil	31	low	low-medium	low	low
17	JwtUserDetailsSer...	23	low	low-medium	low	low
18	SwaggerConfig	20	low	low-medium	low	low
19	CorsFilterConfigu...	18	low	low-medium	low	low
20	EventSource	12	low	low-medium	low	low
21	ParseResponseExce...	5	medium-high	low	low	low
22	UserException	4	medium-high	low	low	low
23	ReviewException	4	medium-high	low	low	low
24	HttpException	13	low-medium	low	low	low
25	ServiceReview	11	low-medium	low	low	low
26	ServiceController	10	low-medium	low	low	low
27	UserController	10	low-medium	low	low	low
28	BuyerReview	9	low-medium	low	low	low
29	UserType	7	low-medium	low	low	low
30	UserState	6	low-medium	low	low	low
31	PlatformReview	6	low-medium	low	low	low
32	ModifiedPlatformR...	6	low-medium	low	low	low
33	ServiceState	5	low-medium	low	low	low
34	ReviewType	5	low-medium	low	low	low
35	ReviewState	4	low-medium	low	low	low
36	DeletedPlatformRe...	3	low-medium	low	low	low

37	NewPlatformReview...	3	low-medium	low	low	low
38	PlatformReviewsEv...	3	low-medium	low	low	low
39	QuestionState	2	low-medium	low	low	low
40	ReviewUtils	65	low	low	low	low-medium
41	User	54	low	low	low	low-medium
42	Service	32	low	low	low	low
43	MPUserDetails	32	low	low	low-medium	low
44	EventSink	31	low	low	low	low
45	ReviewShowDto	30	low	low	low	low
46	DeferredLiquibase...	30	low	low	low	low
47	PlatformReviewSho...	26	low	low	low	low
48	Review	25	low	low	low	low
49	PlatformRatingSer...	24	low	low	low	low
50	ValidatorExceptio...	19	low	low	low	low
51	ReviewQuestion	19	low	low	low	low
52	ReviewAnswerSetDto	18	low	low	low	low
53	ReviewQuestionSetDto	17	low	low	low	low
54	EventConstants	17	low	low	low	low
55	ServiceReviewsSum...	16	low	low	low	low
56	ReviewQuestionEdi...	14	low	low	low	low
57	MarketplaceApplic...	14	low	low	low	low
58	ReviewAnswer	14	low	low	low	low
59	UserBuilder	14	low	low	low	low
60	ReviewAnswerEditDto	13	low	low	low	low
61	ReviewAnswerShowDto	13	low	low	low	low
62	PlatformReviewSetDto	13	low	low	low	low
63	ReviewUserDto	12	low	low	low	low
64	PlatformRating	12	low	low	low	low
65	UserReviewsSummar...	11	low	low	low	low
66	ReviewEditDto	11	low	low	low	low
67	ReviewSetDto	11	low	low	low	low
68	ReviewQuestionSho...	10	low	low	low	low
69	PlatformReviewEdi...	10	low	low	low	low
70	PlatformReviewSum...	9	low	low	low	low
71	ReviewBuilder	9	low	low	low	low
72	CountryRegexConfig	9	low	low	low	low
73	ObjectInfoDto	8	low	low	low	low
74	ServiceReviewAver...	8	low	low	low	low
75	PlatformReviewSho...	8	low	low	low	low
76	ReviewShowDtoBuilder	8	low	low	low	low
77	PlatformReviewRep...	8	low	low	low-medium	low
78	PlatformReviewEma...	8	low	low	low	low
79	ServiceBuilder	7	low	low	low	low
80	BusinessReviewEdi...	7	low	low	low	low
81	ServiceReviewRepo...	7	low	low	low	low
82	ServiceRepository	7	low	low	low	low
83	EventBusConfig	6	low	low	low	low
84	ExceptionResponse	6	low	low	low	low
85	ReviewQuestionBui...	6	low	low	low	low
86	BuyerReviewReposi...	6	low	low	low	low
87	ReviewEvent	5	low	low	low	low
88	ActionService	5	low	low	low	low
89	PlatformReviewSet...	4	low	low	low	low
90	PlatformReviewEdi...	4	low	low	low	low
91	PlatformReviewSum...	4	low	low	low	low
92	ReviewQuestionRep...	4	low	low	low	low
93	UserRepository	4	low	low	low	low
94	UserReviewsSummar...	3	low	low	low	low
95	ReviewQuestionEdi...	3	low	low	low	low
96	ReviewAnswerShowD...	3	low	low	low	low
97	ReviewSetDtoBuilder	3	low	low	low	low
98	ReviewEditDtoBuilder	3	low	low	low	low
99	ReviewUserDtoBuilder	3	low	low	low	low
100	ReviewAnswerBuilder	3	low	low	low	low
101	ReviewQuestionSet...	2	low	low	low	low
102	ReviewAnswerEditD...	2	low	low	low	low
103	ReviewAnswerSetDt...	2	low	low	low	low

104	ServiceReviewBuilder	2	low	low	low	low
105	PlatformRatingBui...	2	low	low	low	low
106	PlatformRatingRep...	2	low	low	low	low
107	PlatformReviewEma...	2	low	low	low	low
108	BusinessReviewEdi...	1	low	low	low	low
109	BuyerReviewBuilder	1	low	low	low	low
110	PlatformReviewBui...	0	low	low	low	low
111	BuyerReviewBuilde...	0	low	low	low	low
112	ServiceReviewBuil...	0	low	low	low	low
113	PlatformReviewBui...	0	low	low	low	low

Tabela J.4: Relatório relativo ao MicroReview.

MICRO CONTACT REQUEST - List of all classes (#76)						
ID	CLASS	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	HttpClient	74	low	high	low	low-medium
2	ServiceOrderService	108	low-medium	medium-high	medium-high	low-medium
3	WebSecurityConfig	65	low-medium	medium-high	low-medium	low-medium
4	GeocodingUtils	44	low-medium	medium-high	low	low
5	SendgridService	61	low	medium-high	low	low-medium
6	KafkaConfig	52	low	medium-high	low	low-medium
7	EventSink	41	low-medium	low-medium	low	low
8	JwtRequestFilter	35	low-medium	low-medium	low	low
9	ExceptionHandling	9	low-medium	low-medium	low	low
10	EmailService	50	low	low-medium	low	low
11	JwtTokenUtil	31	low	low-medium	low	low
12	EventSource	24	low	low-medium	low	low
13	JwtUserDetailsSer...	23	low	low-medium	low	low
14	SwaggerConfig	20	low	low-medium	low	low
15	CorsFilterConfigu...	18	low	low-medium	low	low
16	ParseResponseExce...	5	medium-high	low	low	low
17	UserException	4	medium-high	low	low	low
18	AddressService	16	low-medium	low	low	low
19	GeocodingStatusCodes	13	low-medium	low	low	low
20	HttpException	13	low-medium	low	low	low
21	UserController	10	low-medium	low	low	low
22	UserType	7	low-medium	low	low	low
23	UserState	6	low-medium	low	low	low
24	ServiceState	5	low-medium	low	low	low
25	PriceType	4	low-medium	low	low	low
26	ServiceOrderStatus	2	low-medium	low	low	low
27	UserService	45	low	low	low	low
28	Address	37	low	low	low	low
29	User	37	low	low	low	low
30	Service	33	low	low	low	low
31	AddressDto	32	low	low	low	low
32	MPUserDetails	32	low	low	low-medium	low
33	DeferredLiquibase...	30	low	low	low	low
34	ServiceService	25	low	low	low	low
35	EventConstants	25	low	low	low	low
36	GeocodingAPIMock	22	low	low	low	low
37	GeocodingData	22	low	low	low	low
38	ShowOwnUserDto	22	low	low	low	low
39	PriceDto	21	low	low	low	low
40	ValidatorExceptio...	19	low	low	low	low
41	OrderDto	19	low	low	low	low
42	ServiceOrder	19	low	low	low	low
43	ShowUserDto	17	low	low	low	low
44	Price	15	low	low	low	low
45	MarketplaceApplic...	14	low	low	low	low
46	Geolocation	13	low	low	low	low
47	UserBuilder	13	low	low	low	low
48	EmailContactDto	13	low	low	low	low
49	RequestInfoBuyerE...	12	low	low	low	low

50	GeocodingConstants	11	low	low	low	low
51	ServiceOrderContr...	11	low	low	low	low
52	AddressBuilder	9	low	low	low	low
53	GeocodingScheduler	9	low	low	low	low
54	CountryRegexConfig	9	low	low	low	low
55	RequestInfoProvid...	8	low	low	low	low
56	AddressDtoBuilder	7	low	low	low	low
57	ShowOwnUserDtoBui...	7	low	low	low	low
58	ServiceBuilder	7	low	low	low	low
59	ContactInfoDto	7	low	low	low	low
60	OrderCountDto	7	low	low	low	low
61	ExceptionResponse	6	low	low	low	low
62	ServiceOrderBuilder	6	low	low	low	low
63	RequestInfoBuyerE...	6	low	low	low	low
64	ShowUserDtoBuilder	5	low	low	low	low
65	PriceBuilder	5	low	low	low	low
66	OrderDtoBuilder	5	low	low	low	low
67	ActionService	5	low	low	low	low
68	PriceDtoBuilder	4	low	low	low	low
69	OrderRepository	4	low	low	low	low
70	ServiceRepository	4	low	low	low	low
71	GeolocationBuilder	3	low	low	low	low
72	UserRepository	3	low	low	low	low
73	AddressRepository	3	low	low	low	low
74	ContactInfoDtoBui...	2	low	low	low	low
75	OrderCountDtoBuilder	2	low	low	low	low
76	RequestInfoProvid...	2	low	low	low	low

Tabela J.5: Relatório relativo ao MicroContactRequest.

MICRO SERVICE - List of all classes (#121)						
ID	CLASS	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	ServiceService	274	medium-high	**very-high**	high	medium-high
2	HttpClient	74	low	high	low	low-medium
3	CategoryService	120	low-medium	medium-high	medium-high	low-medium
4	WebSecurityConfig	65	low-medium	medium-high	low-medium	low-medium
5	GeocodingUtils	44	low-medium	medium-high	low	low
6	KafkaConfig	52	low	medium-high	low	low-medium
7	SearchService	174	low-medium	low-medium	low-medium	low-medium
8	SearchController	48	low-medium	low-medium	low-medium	low
9	JwtRequestFilter	35	low-medium	low-medium	low	low
10	ExceptionHandling	9	low-medium	low-medium	low	low
11	UserService	58	low	low-medium	low-medium	low-medium
12	AmazonClient	37	low	low-medium	low	low
13	JwtTokenUtil	31	low	low-medium	low	low
14	ServiceDto	29	low	low-medium	low	low
15	ShowFullServiceDto	28	low	low-medium	low	low
16	ServiceShowOwnDto	28	low	low-medium	low	low
17	EventSource	24	low	low-medium	low	low
18	JwtUserDetailsSer...	23	low	low-medium	low	low
19	SwaggerConfig	20	low	low-medium	low	low
20	CorsFilterConfigu...	18	low	low-medium	low	low
21	AdminServiceContr...	35	medium-high	low	low	low
22	ParseResponseExce...	5	medium-high	low	low	low
23	UserException	4	medium-high	low	low	low
24	CategoryException	4	medium-high	low	low	low
25	CategoryController	40	low-medium	low	low-medium	low
26	AddressService	16	low-medium	low	low	low
27	GeocodingStatusCodes	13	low-medium	low	low	low
28	HttpException	13	low-medium	low	low	low
29	SearchSort	10	low-medium	low	low	low
30	UserController	10	low-medium	low	low	low
31	UserType	7	low-medium	low	low	low
32	UserState	6	low-medium	low	low	low

33	ServiceState	5	low-medium	low	low	low
34	PriceType	4	low-medium	low	low	low
35	ReviewState	4	low-medium	low	low	low
36	Service	43	low	low	low	low
37	Address	37	low	low	low	low
38	User	34	low	low	low	low
39	AddressDto	32	low	low	low	low
40	MPUserDetails	32	low	low	low-medium	low
41	ServiceRepository	31	low	low	medium-high	low
42	BusinessHoursDto	30	low	low	low	low
43	SearchResultDto	30	low	low	low	low
44	DeferredLiquibase...	30	low	low	low	low
45	WishListController	28	low	low	low	low
46	ServiceSetDto	27	low	low	low	low
47	EventSink	27	low	low	low	low
48	Category	26	low	low	low	low
49	ServiceUtils	26	low	low	low	low
50	ServiceShowBackof...	24	low	low	low	low
51	ServiceShowDto	24	low	low	low	low
52	GeocodingAPIMock	22	low	low	low	low
53	GeocodingData	22	low	low	low	low
54	PriceDto	21	low	low	low	low
55	AutocompleteDto	20	low	low	low	low
56	AddressDataValidator	20	low	low	low	low
57	ValidatorExceptio...	19	low	low	low	low
58	EventConstants	19	low	low	low	low
59	CategoryShowDto	18	low	low	low	low
60	BusinessHours	18	low	low	low	low
61	AmazonClientConfig	17	low	low	low	low
62	ReviewService	15	low	low	low	low
63	Price	15	low	low	low	low
64	ShowUserDto	14	low	low	low	low
65	ServiceBuilder	14	low	low	low	low
66	MarketplaceApplic...	14	low	low	low	low
67	Geolocation	13	low	low	low	low
68	ServiceController	13	low	low	low	low
69	ReviewShowDto	13	low	low	low	low
70	UploadController	13	low	low	low	low
71	CategorySetDto	12	low	low	low	low
72	ReviewUserDto	12	low	low	low	low
73	ObjectInfoDto	11	low	low	low	low
74	SearchResultDtoBu...	11	low	low	low	low
75	ServiceDtoBuilder	11	low	low	low	low
76	GeocodingConstants	11	low	low	low	low
77	ShowWishlistItemDto	10	low	low	low	low
78	UserBuilder	10	low	low	low	low
79	AddressBuilder	9	low	low	low	low
80	ServicesByCategor...	9	low	low	low	low
81	GeocodingScheduler	9	low	low	low	low
82	ReviewAnswerShowDto	9	low	low	low	low
83	CountryRegexConfig	9	low	low	low	low
84	ServiceReviewAver...	8	low	low	low	low
85	SearchResultCateg...	8	low	low	low	low
86	CategorySearchDto	8	low	low	low	low
87	ServiceFilterShowDto	8	low	low	low	low
88	ServiceSetDtoBuilder	8	low	low	low	low
89	SearchDto	8	low	low	low	low
90	BusinessHoursBuilder	8	low	low	low	low
91	ReviewShowDtoBuilder	8	low	low	low	low
92	CategoryRepository	8	low	low	low	low
93	AddressDataConstr...	8	low	low	low	low
94	ImageUploadServic...	7	low	low	low	low
95	OrderDto	7	low	low	low	low
96	BusinessHoursDtoB...	7	low	low	low	low
97	AddressDtoBuilder	7	low	low	low	low
98	ServiceReviewsSum...	7	low	low	low	low
99	ReviewQuestionSho...	7	low	low	low	low

100	CategoryShowDtoBu...	6	low	low	low	low
101	CategoryBuilder	6	low	low	low	low
102	ExceptionResponse	6	low	low	low	low
103	UserRepository	6	low	low	low	low
104	ShowUrlDto	6	low	low	low	low
105	ShowWishlistItemD...	5	low	low	low	low
106	PriceBuilder	5	low	low	low	low
107	PriceDtoBuilder	4	low	low	low	low
108	AutocompleteResul...	4	low	low	low	low
109	CategorySetDtoBui...	4	low	low	low	low
110	ShowUserDtoBuilder	4	low	low	low	low
111	GeolocationBuilder	3	low	low	low	low
112	SearchDtoBuilder	3	low	low	low	low
113	ServiceFilterShow...	3	low	low	low	low
114	CategorySearchDto...	3	low	low	low	low
115	ReviewAnswerShowD...	3	low	low	low	low
116	ReviewUserDtoBuilder	3	low	low	low	low
117	ActionService	3	low	low	low	low
118	AddressRepository	3	low	low	low	low
119	OrderDtoBuilder	2	low	low	low	low
120	ServicesByCategor...	2	low	low	low	low
121	ShowUrlDtoBuilder	1	low	low	low	low

Tabela J.6: Relatório relativo ao MicroService.

MICRO USER - List of all classes (#126)						
ID	CLASS	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	UserService	410	high	**very-high**	high	high
2	HttpClient	74	low	high	low	low-medium
3	AdminUserController	68	medium-high	medium-high	medium-high	low-medium
4	WebSecurityConfig	65	low-medium	medium-high	low-medium	low-medium
5	GeocodingUtils	44	low-medium	medium-high	low	low
6	SendgridService	64	low	medium-high	low	low-medium
7	KafkaConfig	52	low	medium-high	low	low-medium
8	JwtRequestFilter	35	low-medium	low-medium	low	low
9	ExceptionHandling	9	low-medium	low-medium	low	low
10	ProviderShowDto	60	low	low-medium	low	low-medium
11	ShowFullUserDto	59	low	low-medium	low	low-medium
12	JwtTokenUtil	34	low	low-medium	low	low
13	ShowOwnUserDto	30	low	low-medium	low	low
14	JwtUserDetailsSer...	23	low	low-medium	low	low
15	SwaggerConfig	20	low	low-medium	low	low
16	CorsFilterConfigu...	18	low	low-medium	low	low
17	ParseResponseExce...	5	medium-high	low	low	low
18	UserException	4	medium-high	low	low	low
19	UserController	19	low-medium	low	low-medium	low
20	AddressService	16	low-medium	low	low	low
21	GeocodingStatusCodes	13	low-medium	low	low	low
22	HttpException	13	low-medium	low	low	low
23	UserType	7	low-medium	low	low	low
24	UserState	6	low-medium	low	low	low
25	PriceType	4	low-medium	low	low	low
26	ReviewState	4	low-medium	low	low	low
27	ServiceOrderStatus	2	low-medium	low	low	low
28	AnnualSubscriptio...	2	low-medium	low	low	low
29	User	48	low	low	low	low
30	PhoneDataValidator	45	low	low	low	low
31	ProviderDto	38	low	low	low	low
32	Address	37	low	low	low	low
33	UserCreatedDto	33	low	low	low	low
34	AddressDto	32	low	low	low	low
35	MPUserDetails	32	low	low	low-medium	low
36	DeferredLiquibase...	30	low	low	low	low
37	UserCreatedDtoBui...	26	low	low	low	low

38	UserBuilder	25	low	low	low	low
39	ShowPublicUserDto	23	low	low	low	low
40	GeocodingAPIMock	22	low	low	low	low
41	GeocodingData	22	low	low	low	low
42	ShowFullUserDtoBu...	22	low	low	low	low
43	ProviderShowDtoBu...	20	low	low	low	low
44	AddressDataValidator	20	low	low	low	low
45	ValidatorExceptio...	19	low	low	low	low
46	UserRepository	19	low	low	low-medium	low
47	EventSource	18	low	low	low	low
48	ServiceService	17	low	low	low	low
49	ShowUserDto	17	low	low	low	low
50	ProviderEditDto	17	low	low	low	low
51	OrderService	17	low	low	low	low
52	ProviderDtoBuilder	16	low	low	low	low
53	ReviewService	15	low	low	low	low
54	SubscriptionService	15	low	low	low	low
55	MarketplaceApplic...	14	low	low	low	low
56	EventSink	14	low	low	low	low
57	Geolocation	13	low	low	low	low
58	ServiceShowDto	13	low	low	low	low
59	SetUserDto	13	low	low	low	low
60	ReviewShowDto	13	low	low	low	low
61	PlatformReviewSho...	13	low	low	low	low
62	DeletionReason	13	low	low	low	low
63	BusinessHoursDto	12	low	low	low	low
64	ReviewUserDto	12	low	low	low	low
65	VerifyPhoneCodeDto	11	low	low	low	low
66	ShowOwnUserDtoBui...	11	low	low	low	low
67	ShowOwnSubscripti...	11	low	low	low	low
68	UserReviewsSummar...	11	low	low	low	low
69	DeletionReasonSer...	11	low	low	low	low
70	GeocodingConstants	11	low	low	low	low
71	JwtResponse	10	low	low	low	low
72	SetUserWithTypeDto	10	low	low	low	low
73	OrderDto	10	low	low	low	low
74	AddressBuilder	9	low	low	low	low
75	PriceDto	9	low	low	low	low
76	SendPhoneCodeDto	9	low	low	low	low
77	UserIdDto	9	low	low	low	low
78	DeletionReasonDto	9	low	low	low	low
79	GeocodingScheduler	9	low	low	low	low
80	ReviewAnswerShowDto	9	low	low	low	low
81	CountryRegexConfig	9	low	low	low	low
82	EventConstants	9	low	low	low	low
83	SubscriptionShowDto	9	low	low	low	low
84	ShowPublicUserDto...	8	low	low	low	low
85	UserFilterDto	8	low	low	low	low
86	ProviderEditDtoBu...	8	low	low	low	low
87	ReviewShowDtoBuilder	8	low	low	low	low
88	PlatformReviewSho...	8	low	low	low	low
89	AddressDataConstr...	8	low	low	low	low
90	WelcomeProviderEm...	8	low	low	low	low
91	ProviderVerifiedE...	8	low	low	low	low
92	WelcomeEmailDto	8	low	low	low	low
93	BusinessHoursDtoB...	7	low	low	low	low
94	VerifyUserDto	7	low	low	low	low
95	AddressDtoBuilder	7	low	low	low	low
96	ServiceReviewsSum...	7	low	low	low	low
97	ReviewQuestionSho...	7	low	low	low	low
98	PhoneDataConstraint	7	low	low	low	low
99	DeleteAccountEmai...	7	low	low	low	low
100	ExceptionResponse	6	low	low	low	low
101	SearchResultCateg...	5	low	low	low	low
102	ShowOwnSubscripti...	5	low	low	low	low
103	ShowUserDtoBuilder	5	low	low	low	low
104	ObjectInfoDto	5	low	low	low	low

105	OrderDtoBuilder	5	low	low	low	low
106	PriceDtoBuilder	4	low	low	low	low
107	SetUserWithTypeDt...	4	low	low	low	low
108	GeolocationBuilder	3	low	low	low	low
109	UserFilterDtoBuilder	3	low	low	low	low
110	SetUserDtoBuilder	3	low	low	low	low
111	VerifyPhoneCodeDt...	3	low	low	low	low
112	ReviewAnswerShowD...	3	low	low	low	low
113	UserReviewsSummar...	3	low	low	low	low
114	ReviewUserDtoBuilder	3	low	low	low	low
115	SubscriptionShowD...	3	low	low	low	low
116	DeletionReasonRep...	3	low	low	low	low
117	AddressRepository	3	low	low	low	low
118	SendPhoneCodeDtoB...	2	low	low	low	low
119	VerifyUserDtoBuilder	2	low	low	low	low
120	DeletionReasonBui...	2	low	low	low	low
121	ProviderVerifiedE...	2	low	low	low	low
122	WelcomeProviderEm...	2	low	low	low	low
123	WelcomeEmailDtoBu...	2	low	low	low	low
124	DeletionReasonDto...	1	low	low	low	low
125	UserIdDtoBuilder	1	low	low	low	low
126	DeleteAccountEmai...	1	low	low	low	low

Tabela J.7: Relatório relativo ao MicroUser.

MICRO AUTH - List of all classes (#106)						
ID	CLASS	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	RegistrationService	233	low-medium	high	high	low-medium
2	HttpClient	74	low	high	low	low-medium
3	WebSecurityConfig	65	low-medium	medium-high	low-medium	low-medium
4	GeocodingUtils	44	low-medium	medium-high	low	low
5	SendgridService	60	low	medium-high	low	low-medium
6	KafkaConfig	52	low	medium-high	low	low-medium
7	JwtAuthentication...	30	low	medium-high	low	low
8	JwtRequestFilter	35	low-medium	low-medium	low	low
9	RegistrationContr...	31	low-medium	low-medium	medium-high	low
10	IOExceptionExcept...	9	low-medium	low-medium	low	low
11	ServiceUnavailabl...	9	low-medium	low-medium	low	low
12	LoginExceptionHan...	9	low-medium	low-medium	low	low
13	UserNotActiveExce...	9	low-medium	low-medium	low	low
14	WriterExceptionEx...	9	low-medium	low-medium	low	low
15	LoginFailedExcept...	9	low-medium	low-medium	low	low
16	ItemExpiredExcept...	9	low-medium	low-medium	low	low
17	GoogleAuthController	39	low	low-medium	low	low
18	JwtTokenUtil	31	low	low-medium	low	low
19	JwtUserDetailsSer...	30	low	low-medium	low	low
20	GoogleAuthService	28	low	low-medium	low	low
21	SwaggerConfig	20	low	low-medium	low	low
22	CorsFilterConfigu...	18	low	low-medium	low	low
23	ParseResponseExce...	5	medium-high	low	low	low
24	LoginFailedException	4	medium-high	low	low	low
25	UserException	4	medium-high	low	low	low
26	LoginUserException	4	medium-high	low	low	low
27	RegisterUserExcep...	4	medium-high	low	low	low
28	ServiceUnavailabl...	4	medium-high	low	low	low
29	UserNotActiveExce...	4	medium-high	low	low	low
30	ItemExpiredException	4	medium-high	low	low	low
31	AddressService	16	low-medium	low	low	low
32	GeocodingStatusCodes	13	low-medium	low	low	low
33	HttpException	13	low-medium	low	low	low
34	UserType	7	low-medium	low	low	low
35	UserType	7	low-medium	low	low	low
36	UserState	6	low-medium	low	low	low
37	UserState	6	low-medium	low	low	low

38	RegisterUserExcep...	6	low-medium	low	low	low
39	PriceType	4	low-medium	low	low	low
40	ProviderShowDto	57	low	low	low	low-medium
41	CredentialRepository	50	low	low	low	low
42	LoginUser	47	low	low	low	low
43	PhoneDataValidator	45	low	low	low	low
44	ProviderDto	38	low	low	low	low
45	Address	37	low	low	low	low
46	AddressDto	32	low	low	low	low
47	DeferredLiquibase...	30	low	low	low	low
48	LoginUserBuilder	29	low	low	low	low
49	MPUserDetails	29	low	low	low	low
50	EmailService	24	low	low	low	low
51	GeocodingAPIMock	22	low	low	low	low
52	GeocodingData	22	low	low	low	low
53	ProviderShowDtoBu...	20	low	low	low	low
54	AddressDataValidator	20	low	low	low	low
55	ValidatorExceptio...	19	low	low	low	low
56	EventSource	18	low	low	low	low
57	ProviderEditDto	17	low	low	low	low
58	ProviderDtoBuilder	16	low	low	low	low
59	ShowLoginUserDto	16	low	low	low	low
60	UserSetDto	15	low	low	low	low
61	MarketplaceApplic...	14	low	low	low	low
62	UserService	14	low	low	low	low
63	EventSink	14	low	low	low	low
64	Geolocation	13	low	low	low	low
65	GeocodingConstants	11	low	low	low	low
66	JwtResponse	10	low	low	low	low
67	GoogleAuthConfig	10	low	low	low	low
68	AddressBuilder	9	low	low	low	low
69	GeocodingScheduler	9	low	low	low	low
70	CountryRegexConfig	9	low	low	low	low
71	EventConstants	9	low	low	low	low
72	ResetPasswordDto	9	low	low	low	low
73	RegisterEmailDto	9	low	low	low	low
74	RecoverPasswordEm...	9	low	low	low	low
75	JwtRequest	8	low	low	low	low
76	AddressDataConstr...	8	low	low	low	low
77	PasswordChangeDto	8	low	low	low	low
78	RecoverPasswordDto	8	low	low	low	low
79	ProviderEditDtoBu...	8	low	low	low	low
80	AuthenticationDto	8	low	low	low	low
81	WelcomeProviderEm...	8	low	low	low	low
82	WelcomeEmailDto	8	low	low	low	low
83	AuthService	7	low	low	low	low
84	PhoneDataConstraint	7	low	low	low	low
85	ValidateKeyDto	7	low	low	low	low
86	ScratchCodesDto	7	low	low	low	low
87	VerifyUserDto	7	low	low	low	low
88	ValidationDto	7	low	low	low	low
89	AddressDtoBuilder	7	low	low	low	low
90	ExceptionResponse	6	low	low	low	low
91	JwtAuthentication...	6	low	low	low	low
92	ShowLoginUserDtoB...	5	low	low	low	low
93	LoginUserRepository	4	low	low	low	low
94	UserSetDtoBuilder	4	low	low	low	low
95	GeolocationBuilder	3	low	low	low	low
96	PasswordChangeDto...	3	low	low	low	low
97	AddressRepository	3	low	low	low	low
98	RegisterEmailDtoB...	3	low	low	low	low
99	RecoverPasswordEm...	3	low	low	low	low
100	ResetPasswordDtoB...	2	low	low	low	low
101	VerifyUserDtoBuilder	2	low	low	low	low
102	RecoverPasswordDt...	2	low	low	low	low
103	WelcomeProviderEm...	2	low	low	low	low
104	WelcomeEmailDtoBu...	2	low	low	low	low

105	ScratchCodesDtoBu...	1	low	low	low	low
106	ValidationDtoBuilder	1	low	low	low	low

Tabela J.8: Relatório relativo ao MicroAuth.