Raphael Collin Teles Manhães

Proposta de *Design* para uma locadora de carros: uma abordagem baseada em *Domain-Driven Design* e Microsserviços

Campos dos Goytacazes-RJ

Abril de 2024

Raphael Collin Teles Manhães

Proposta de *Design* para uma locadora de carros: uma abordagem baseada em *Domain-Driven Design* e Microsserviços

Trabalho de Conclusão apresentado ao curso Bacharelado em Sistemas de Informação do Instituto Federal de Educação, Ciência e Tecnologia Fluminense, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Instituto Federal de Educação, Ciência e Tecnologia Fluminense

Orientador: Prof. D.Sc. Mark Douglas de Azevedo Jacyntho

Campos dos Goytacazes-RJ

Abril de 2024

Raphael Collin Teles Manhães

Proposta de *Design* para uma locadora de carros: uma abordagem baseada em *Domain-Driven Design* e Microsserviços

Trabalho de Conclusão apresentado ao curso Bacharelado em Sistemas de Informação do Instituto Federal de Educação, Ciência e Tecnologia Fluminense, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Campos dos Goytacazes-RJ, 29 de abril de 2024.

Prof. M.Sc. Nome do Primeiro Professor
Instituto Federal Fluminense (IFF)

Prof. D.Sc. Nome do Segundo Professor
Universidade Estadual do Norte Fluminense - Darcy Ribeiro (UENF)

Prof. D.Sc. Mark Douglas de Azevedo Jacyntho
Instituto Federal Fluminense (IFF)

Instituto Federal Fluminense (IFF)

Campos dos Goytacazes-RJ
Abril de 2024

Agradecimentos

Acima de tudo, agradecemos a Deus pelo dom da vida e pela conclusão do projeto.

Dedicamos este trabalho a nossa família, amigos, ao nosso professor...

Resumo

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi. Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor. Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Palavras-chaves: Curabitur, Malesuada, MassaAndroid, Aliquam.

Abstract

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi. Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor. Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Keywords: Curabitur, Malesuada, MassaAndroid, Aliquam.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Lista de codigos

Siglas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade	23, 62
ACL	Anti-Corruption Layer	31, 62
AMS	Arquitetura de Microsserviços	12, 13, 14, 17, 19, 20, 21, 22, 23, 29, 33, 34, 35, 36, 39, 62
API	Application Programming Interface	33, 36, 62
BC	Bounded Context	13, 28, 31, 33, 34, 35, 62
CPU	Central Processing Unit	17, 62
DDD	Domain-Driven Design	12, 13, 14, 15, 21, 22, 23, 25, 26, 29, 33, 34, 35, 37, 38, 39, 62
DVD	Digital Versatile Disc	12, 62
IDE	Integrated Development Environment	17, 23, 62

IPC	Inter-Process Communication	18, 23, 62
LGPD	Lei Geral de Proteção de Dados Pessoais	22, 62
OHS	Open Host Service	31, 62
POO	Programação Orientada a Objetos	15, 23, 24, 25, 26, 62
SOA	Service-Oriented Architecture	15, 16, 17, 62
VO	Value Object	26, 27, 28, 62

Sumário

1 Introdução

O presente capítulo propõe uma contextualização ampla dos conceitos pertinentes, abrangendo monólitos, microsserviços e *Domain-Driven Design (DDD)*, juntamente com a exposição da justificativa, dos objetivos e da estrutura organizacional da monografia.

1.1 Problema e contexto

Em agosto de 2008, um grande incidente interrompeu as operações da *Netflix*, empresa que nesse período alugava DVDs através de seu site na *web*. Uma corrupção no banco de dados impediu que a companhia pudesse realizar envios por três dias. A partir desse momento, os engenheiros perceberam que necessitavam reduzir pontos únicos de falha, tais como grandes bases de dados relacionais armazenadas em somente um *data center*. Em vez disso, optaram por adotar sistemas distribuídos na nuvem, escalonados horizontalmente, para garantir maior confiabilidade e minimizar o impacto das falhas (**??**).

Essa transição resultou em vários benefícios para a *Netflix*, destacando-se a conquista de uma disponibilidade de 'quatro noves' (99,99%) e significativa redução de custos. Além disso, a Arquitetura de Microsserviços (AMS) permitiu que diferentes aplicações utilizassem diferentes tecnologias, facilitando contratações e extraindo o melhor de cada opção. Assim, em 2016 (8 anos após a migração) a empresa tinha 8 vezes mais assinantes, muito mais ativos, com o número de visualizações crescendo em três ordens de grandeza nesse período (**??**).

A abordagem convencional no desenvolvimento de sistemas corporativos, na qual toda a lógica de negócio, persistência e apresentação são encapsuladas em uma única aplicação (e um único executável), é a mais natural para construção desse tipo de sistema. Essa arquitetura, conhecida comumente como monólito, tem alcançado sucesso devido à sua simplicidade no desenvolvimento, testes, implantação e monitoramento, entre outros fatores. No entanto, com a popularização da computação em nuvem, a crescente demanda de mudanças cada vez mais frequentes e o crescimento das equipes de tecnologia da informação em grandes organizações, essa arquitetura é confrontada com desafios significativos. Inicialmente, destaca-se o acoplamento nos ciclos de mudanças, onde qualquer alteração em uma pequena parte requer a compilação e implantação de todo sistema. Além disso, há dificuldade em escalar horizontalmente apenas funcionalidades específicas que estão sendo mais demandas e existe complexidade na utilização de diferentes linguagens de programação em diversas áreas de um sistema (**??**). Por esses motivos, a adoção da Arquitetura de Microsserviços cresceu de forma exponencial nos últimos anos, com 77%

dos profissionais afirmando que suas organizações utilizam esse estilo arquitetural (**??**).

Por outro lado, *Domain-Driven Design (DDD)* é uma abordagem para o desenvolvimento de software, na qual os elementos de software como pacotes, classes, interfaces, métodos e nomes de variáveis devem corresponder aos conceitos do domínio de negócio (**??**). Adicionalmente, o DDD se configura como uma estratégia para harmonizar especialistas de negócios, desenvolvedores e projetistas, promovendo uma otimização nas interações ao reduzir o mapeamento entre termos de negócio e termos técnicos.

A Arquitetura de Microsserviços (AMS) e DDD podem ser combinados no *design* e desenvolvimento um sistema. De um lado, microsserviços possibilitam escalabilidade independente, implantação desacoplada e utilização de múltiplas tecnologias para determinados casos de uso. Por outro lado, DDD fornece uma abordagem para modelagem do domínio de negócio, diversos padrões para resolver problemas de modelagem recorrentes e facilidade de entendimento e manutenção de código. Além disso, DDD é uma excelente estratégia para definição de limites entre microsserviços. Ao modelar os serviços com base em *Bounded Context*s coesos, é possível o desenvolvimento de novas funcionalidades mais rapidamente e também se torna mais simples a recombinação de microsserviços para fornecer novas funcionalidades aos usuários (**??**).

1.2 Justificativa

Recentemente, observou-se um notável crescimento em popularidade e adoção da AMS e da abordagem *Domain-Driven Design (DDD)* no desenvolvimento de sistemas corporativos. Porém, essas tecnologias são utilizadas de maneira equivocada em muitas implementações devido à falta de compreensão das suas vantagens e desvantagens, entre outros fatores. Por exemplo, a equipe de desenvolvimento do *Amazon Prime Video* recentemente anunciou uma migração de um serviço de monitoramento de vídeo, que foi inicialmente desenvolvido com microsserviços, para uma arquitetura monolítica com objetivo de aumentar a escalabilidade e reduzir custos (**??**).

Paralelamente a isso, é possível identificar uma carência de estudos de caso realistas envolvendo a utilização simultânea da AMS e *Domain-Driven Design (DDD)*, especialmente no cenário nacional. Enquanto algumas publicações concentram-se exclusivamente em um desses conceitos e outras apresentam exemplos simplificados, proporcionando uma compreensão limitada de como essas duas técnicas podem ser efetivamente empregadas em conjunto.

Nesse contexto, este trabalho visa preencher essa lacuna ao oferecer uma contribuição significativa para a compreensão da aplicação integrada dessas estratégias no desenvolvimento de sistemas. Por meio de um estudo de caso realista, busca-se não apenas

enriquecer o conhecimento acadêmico, mas também fornecer uma base sólida para a implementação prática dessas tecnologias em diversos setores da indústria.

1.3 Objetivos

O objetivo geral deste trabalho de conclusão de curso é elaborar uma proposta de *design* para um sistema de locação de veículos realista utilizando a Arquitetura de Microsserviços (AMS) e a abordagem Domain-Driven Design (DDD).

1.3.1 Objetivos Específicos

Para alcançar esse objetivo requer alcançar os seguintes objetivos específicos:

- Coletar os requisitos chaves de um sistema de locação de veículos.
- Elaborar o *design* do sistema, que por um lado satisfaça os requisitos e, por outro lado, utilize a AMS e aplique DDD para definição de limites entre os componentes.
- Realizar uma implementação funcional do *design*.

1.4 Estrutura do Trabalho

Este trabalho está dividido em sete capítulos. O Capítulo 1 expõe o contexto do estudo, as justificativas desta pesquisa e os objetivos a serem atingidos. O Capítulo 2 apresenta conceitos de caracterização de materiais, relata algumas normas de organizações internacionais para este fim e faz uma explanação sobre a área de processamento e análise de imagens. O Capítulo 3... . Finalmente, o Capítulo 7 apresenta a conclusão e os potenciais trabalhos futuros a serem desenvolvidos.

2 Fundamentação Teórica

Este capítulo apresenta os conceitos de Arquitetura de software, Monólito, Sistemas Distribuídos, Service-Oriented Architecture (SOA) Microsserviços, Programação Orientada a Objetos (POO), Domain-Driven Design (DDD) e Arquitetura Hexagonal.

2.1 Arquitetura de software

Há muitas definições sobre o termo "arquitetura de software". Uma conceituação abrangente e precisa é proposta por **??**) "a arquitetura de software de sistema computacional é um conjunto de estruturas necessárias para compreender o sistema, que engloba elementos de software, suas relações e propriedades." (p. 8, tradução nossa). **??**) complementa afirmando que se trata de decisões de projeto que são importantes e difíceis de serem alteradas.

Em geral, a arquitetura de software pode ser entendida como o processo de organizar as diferentes estruturas de um sistema a fim de satisfazer os requisitos funcionais e não-funcionais atuais e futuros.

Dentro do escopo de arquitetura de software, dois outros termos geram confusão: padrão arquitetural e estilo arquitetural. Enquanto este se refere em termos gerais a uma forma de se organizar um sistema, aquele apresenta uma solução específica para um problema recorrente no contexto de engenharia de software.

As principais características de uma boa arquitetura de software são: flexibilidade, tolerância a falhas, reusabilidade, segurança e simplicidade. Por outro lado, sistemas com arquiteturas mal desenvolvidas são rígidos, frágeis, contém demasiada repetição e possuem uma complexidade desnecessária.

2.2 Monólito

Monólito se refere a um estilo arquitetural em que toda a aplicação é construída e implantada como uma única unidade, geralmente um processo de sistema operacional. **??**) vai além dessa definição, considerando um monólito todo sistema no qual as funcionalidades devem ser obrigatoriamente implantadas de maneira unificada.

Monólitos possuem uma série de vantagens quando comparado com microsserviços como, por exemplo: simplicidade para desenvolver, flexibilidade de fazer grandes alterações na aplicação, facilidade de se realizarem testes e simplicidade para se implantar (**??**).

Portanto, esse estilo arquitetural ainda possui sua aplicabilidade em sistemas de pequeno porte, especialmente quando desenvolvidos de maneira modular a fim de seja possível migrar facilmente para uma arquitetura de microsserviços posteriormente.

Por outro lado, **??**) argumentam que monólitos impõem algumas limitações: ciclos de mudança são acoplados - qualquer mudança pequena requer a recompilação e implantação de todo sistema; é difícil manter um nível de modularização ao longo do tempo; escalar o sistema requer o aumento recursos em todo sistema ao invés das pequenas partes que necessitam de mais recursos.

2.3 Sistema Distribuído

Um sistema distribuído é um conjunto de componentes independentes entre si que se apresenta aos seus usuários de maneira transparente como um sistema único (**??**). São sistemas compostos por diversas partes cooperantes, que são executadas geralmente em processos diferentes interconectados por rede a fim de realizar alguma tarefa.

Diferentemente de sistemas monolíticos, nos quais todos os módulos estão interligados em uma única unidade, os sistemas distribuídos dividem o processamento entre as partes. Essa abordagem traz vantagens significativas, como a possibilidade de escalabilidade horizontal e maior flexibilidade para atualizar e substituir componentes individuais sem afetar o sistema como um todo e o compartilhamento de recursos.

Um dos principais desafios desse tipo de sistema é a concorrência e coordenação entre os componentes. Nessa abordagem, vários componentes podem estar operando simultaneamente e em diferentes locais físicos, o que gera problemas de concorrência, onde múltiplos processos tentam acessar os mesmos recursos compartilhados ou dados simultaneamente. Por esse motivo, torna-se essencial a coordenação entre esses processos para evitar conflitos e garantir a consistência dos dados.

2.3.1 *Service-Oriented Architecture (SOA)*

A arquitetura orientada a serviços ou Service-Oriented Architecture (SOA) é estilo arquitetural no qual um sistema é dividido em componentes de software chamados de serviços, responsáveis por satisfazer uma necessidade de negócio específica.

As políticas, práticas e *frameworks* que permitem que a funcionalidade da aplicação seja fornecida e consumida como conjuntos de serviços publicados em uma granularidade relevante para o consumidor do serviço. Os serviços podem ser invocados, publicados e descobertos, e são abstraídos da implementação usando uma única forma de interface baseada em padrões (**??**).

As principais vantagens da SOA são: reusabilidade dos serviços, simplicidade de manutenção, facilidade de escalar os serviços, maior disponibilidade do sistema e possibilidade de utilização de diferentes tecnologias nos serviços.

Porém, essa arquitetura possui uma maior complexidade para *design*, desenvolvimento, teste e implantação. Além desses fatores, é importante ressaltar um nível maior de dificuldade para depurar erros e encontrar profissionais qualificados para atuar nessa área.

2.4 Microsserviços

A Arquitetura de Microsserviços (AMS) é um subtipo da SOA. No entanto, trata-se de uma subcategoria que orienta como as fronteiras entre os serviços devem ser traçadas e que possui como ponto central a independência de implantação entre os serviços (**??**).

Esse estilo arquitetural foca na criação de um sistema robusto, escalável e de fácil manutenção, desenvolvido através da composição de serviços pequenos, flexíveis e que resolvam uma necessidade negócio específica (**??**). Além disso, cada microsserviço deve possuir sua própria base de dados. A troca de informações entre os componentes deve ser realizada através das interfaces fornecidas.

A implantação de um sistema com a utilização de microsserviços aumenta a resiliência geral do sistema, visto que possui melhor isolamento de falhas. Caso ocorra um mal-funcionamento em algum componente, somente uma parte das funcionalidades ficará indisponível. Tratando-se de um monólito, por outro lado, grande parte ou a totalidade dos recursos são afetados, pois todo o sistema é acoplado em uma única unidade.

Cada serviço na AMS pode ser escalado de maneira independente de outros serviços utilizando, por exemplo, escalonamento horizontal ou vertical (**??**). Ademais, para cada serviço pode ser selecionado o *hardware* mais adequado para seu tipo de processamento. Dessa forma, uma parte do sistema que requer maior utilização da CPU pode ser implantada em uma infraestrutura diferente de outra secção que necessita realizar mais operações de entrada e saída.

Outro benefício dessa arquitetura é o fato de cada serviço ser relativamente pequeno, sendo, assim, de fácil compreensão por todos que trabalham na base de código. Adicionalmente, um serviço pequeno possibilita um bom desempenho da IDE e também pode ser executado localmente de maneira rápida. Esses fatores contribuem para um aumento de produtividade da equipe (**??**).

Por outro lado, com um sistema composto de múltiplos componentes, torna possível a utilização de diferentes tecnologias dentro da cada serviço. Dessa forma, a ferramenta mais apropriada para resolver a necessidade específica do serviço pode ser selecionada,

eliminando a obrigatoriedade de escolher uma única tecnologia para todo o sistema (**??**).

2.4.1 Comunicação entre microsserviços

Quando um sistema é composto por um único monólito, a comunicação entre os diferentes módulos é simplificada, pois se trata de um único processo ao nível de sistema operacional e, portanto, compartilha a área da memória principal. No entanto, microsserviços são implantados em diferentes processos(usualmente em diferentes computadores), assim se faz necessária a utilização de algum mecanismo de comunicação entre processos, também conhecido como Inter-Process Communication (IPC).

Existe no mercado uma série de tecnologias, protocolos e ferramentas que viabilizam a comunicação entre microsserviços. Porém, antes de selecionar o mecanismo de IPC, é importante compreender os estilos de interações entre sistemas (**??**). Essa abordagem coloca foco nos requisitos da aplicação, no lugar de detalhes de um mecanismo específico.

Os modos de interação entre um cliente e um serviço podem ser classificados em duas dimensões: a quantidade de serviços que processam a requisição e a sincronicidade. No que diz respeito à primeira dimensão, existem duas opções: **"um para um"**, em que cada requisição é processada por um único serviço; e **"um para muitos"**, em que cada requisição pode ser processada por vários serviços. Por outro lado, a comunicação pode ser **síncrona**, onde o cliente espera uma resposta imediata e geralmente aguarda essa resposta para continuar seu processamento; ou **assíncrona**, onde o cliente envia a requisição e continua seu processamento sem esperar por uma resposta (**??**).

Quadro 1 – Estilos de comunicação entre microsserviços

	Um para um	**Um para muitos**
Síncrona	*Request/Response*	-
Assíncrona	*Asynchronous request/response* e *One-way notifications*	*Publish/subscribe* *Publish/async responses*

Fonte: **??**).

No Quadro 1, observam-se alguns estilos de comunicação categorizados pelas dimensões anteriormente mencionadas. No Quadro 2 são apresentados os diferentes tipos de comunicação **"um para um"**:

Quadro 2 – Padrões de comunicação do tipo **"um para um"**

Nome	**Descrição**
Request/Response	O serviço cliente realiza uma requisição a outro serviço e espera(geralmente bloqueando sua execução) por uma resposta.
Asynchronous request/response	O serviço cliente envia uma solicitação a um serviço, que responde de forma assíncrona. O cliente não fica bloqueado enquanto espera, pois o serviço pode não enviar a resposta por um longo período.
One-way notifications	O serviço cliente envia uma requisição, mas não espera nenhum tipo de resposta do serviço que processará a requisição.

Fonte: **??**).

Frequentemente, em ambientes de sistemas distribuídos, há a necessidade de vários serviços consumirem uma mensagem produzida por um determinado serviço. Nesse sentido, no Quadro 3 são listadas algumas opções de padrões para realizar comunicações do tipo **"um para muitos"**:

Quadro 3 – Padrões de comunicação do tipo **"um para muitos"**

Nome	**Descrição**
Publish/Subscribe	Um cliente publica uma mensagem, que é consumida por zero ou mais serviços..
Publish/async responses	Um cliente publica uma mensagem contendo uma requisição e espera por um determinado período por respostas.

Fonte: **??**).

2.4.2 Estratégias para delimitação de microsserviços

Uma decisão de grande impacto no projeto de sistema que utilize a AMS é o escopo de cada serviço. Em outras palavras, que funcionalidades serão parte de um determinado serviço e quais serão delegadas a outro componente.

??) argumenta que a possibilidade de alterar um microsserviço de maneira isolada é fundamental para essa arquitetura. Além disso, AMS é apenas outra forma de decomposição modular, afirma o autor. Assim, os conceitos de *information hiding*, coesão e acoplamento podem ser aplicados nesse contexto para criar as delimitações eficientes entre os serviços.

2.4.2.1 *Information Hiding*

Information hiding é uma abordagem para desenvolvimentos de módulos, na qual se busca esconder o máximo de detalhes possíveis detrás da interface de um módulo (**??**). Dessa forma, os benefícios esperados com a construção de módulos de alta qualidade são:

- **Tempo de desenvolvimento aprimorado:** com a possibilidade de módulos serem construídos de forma independente, o desenvolvimento pode ser realizado de maneira paralela.
- **Compreensibilidade:** cada módulo pode ser entendido de maneira isolado. Assim, se torna mais fácil compreender o que o sistema como um todo faz.
- **Flexibilidade:** módulos podem ser modificados de maneira independente, possibilitando, dessa forma, alterações em uma parte do sistema sem a necessidade de alterações em outros módulos.

(**??**).

2.4.2.2 Coesão

Coesão diz respeito à medida que os elementos dentro de um módulo estão organicamente relacionados entre si (**??**). Outra definição possível é código que, alterado conjuntamente, deve estar localizado no mesmo lugar (**??**).

No contexto da AMS, a adição ou atualização de funcionalidades deve, idealmente, envolver um único serviço. Dessa forma, é possível disponibilizar novos recursos aos usuários de maneira mais ágil (**??**). Por esse motivo, buscar alta coesão é importante no momento da definição do escopo de cada serviço. O objetivo principal é evitar que sejam necessárias alterações em múltiplos serviços no desenvolvimento de funcionalidades para o sistema.

2.4.2.3 Acoplamento

Quando serviços são fracamente acoplados, uma alteração em um serviço não requer alteração no outro (**??**). A grande vantagem da AMS é a possibilidade de realizar mudanças em partes do sistema de maneira isolada. Nesse sentido, a projeção de serviços com alta coesão e fraco acoplamento é essencial para que os objetivos da utilização dessa tecnologia sejam atingidos.

Nem todo acoplamento é necessariamente maléfico. De fato, algum nível de acoplamento entre os serviços é inevitável. Além disso, há diferentes tipos de acoplamento listados no Quadro 4.

Quadro 4 – Tipos de acoplamento

Tipo	**Descrição**
Domain Coupling	Descreve uma situação em que um microsserviço necessita interagir com outro serviço para fazer uso de sua(s) funcionalidade(s). Na AMS, esse tipo de iteração é quase inevitável. No entanto, é necessário que cada serviço exponha o mínimo de detalhes possível (*information hiding*) para não gerar um nível de acoplamento muito alto.
Pass-Through Coupling	Acontece quando um microsserviço envia dados a outro serviço somente porque essas informações são necessárias por um terceiro serviço. Trata-se de um dos tipos de acoplamento mais problemáticos, pois implica que o cliente da primeira requisição conheça tanto o serviço que inicialmente recebe a requisição, quanto o que irá, de fato, utilizar os dados.
Common Coupling	Ocorre quando dois ou mais microsserviços compartilham o mesmo conjunto de dados. O exemplo mais comum desse tipo de acoplamento é a utilização de uma base de dados compartilhada. O principal problema dessa abordagem é o fato de modificações ou falhas na base de dados compartilhadas afetar mais de um serviço.
Content Coupling	Descreve uma situação em que o serviço acessa e modifica estados internos de outro serviço. É semelhante ao *Common coupling*, porém nesse caso as linhas de responsabilidades de cada serviço se tornam confusas. Assim, desenvolvedores encontram grandes dificuldades para realizar alterações.

Fonte: **??**).

2.4.2.4 *Domain-Driven Design (DDD)*

Visando a construção de serviços com alta coesão e baixo acoplamento, portanto estáveis, os conceitos de DDD podem ser utilizados. Uma abordagem interessante e comumente aplicada é mapear um Bounded Context (BC) para cada serviço (**??**). Como os Bounded Contexts representam uma parte do domínio onde um modelo se aplica, eles permitem que microsserviços sejam modelados ao redor de uma secção coesa e completamente funcional que ofereça funcionalidades de negócio.

Outra opção para modelagem de microsserviços é mapear cada *Aggregate* para um serviço. Essa estratégia oferece um nível de granularidade menor que um Bounded Context

(BC), visto que este geralmente engloba múltiplos *Aggregate*s. No entanto, **??**) sugere recorrer primeiro a delimitações maiores na construção de microsserviços, englobando um (ou até mais) Bounded Context (BC). Dessa forma, caso a equipe sinta necessidade de subdividir o serviço, poderá fazê-lo sem complicações.

2.4.2.5 Alternativas

Apesar de Domain-Driven Design (DDD) ser uma excelente maneira de delimitar microsserviços, não é a única técnica disponível (**??**). São listadas a seguir algumas alternativas frequentemente utilizadas:

- **Volatilidade:** decomposição baseada em volatilidade refere-se a uma abordagem, na qual partes do sistema que são frequentemente alteradas são extraídas em seus próprios serviços, de forma que possam ser trabalhadas de maneira mais efetiva. Essa estratégia por ser útil em organizações com ritmo de mudanças acelerado. No entanto, se o objetivo da utilização da AMS é a possibilidade de escalar partes do sistema de maneira independente, decomposição baseada em volatilidade não atingirá a meta.
- **Dados:** a natureza das informações que necessitam ser armazenadas pode direcionar a estratégia de decomposição. Por exemplo, é muito usual restringir quais microsserviços podem acessar informações de identificação pessoal para cumprir com legislações como a Lei Geral de Proteção de Dados Pessoais (LGPD).
- **Tecnologia:** A necessidade de se utilizar diferentes tecnologias também pode ser um fator a se considerar na definição do escopo de um microsserviço. Se há a necessidade, por exemplo, de alto desempenho em uma parte do sistema que demande a utilização de uma linguagem de programação diferente, este pode ser um ponto importante para delimitação do serviço.

(**??**).

É importante mencionar que essas opções não são mutualmente exclusivas. Em um sistema real, provavelmente uma mistura de estratégias necessitará ser aplicada para satisfazer os requisitos funcionais e não-funcionais.

2.4.3 Desafios na implementação de microsserviços

Certamente, não há nenhuma tecnologia que não possua limitações. Isso não é diferente com a Arquitetura de Microsserviços (**??**).

Primeiramente, como microsserviços baseiam-se no conceito de Sistema Distribuído, uma série de novos problemas estão presentes nessa arquitetura. Como os serviços

necessitam utilizar algum mecanismo de IPC, há um grande aumento na complexidade do ciclo de desenvolvimento como um todo. IDEs e outras ferramentas de desenvolvimento possuem maior foco no desenvolvimento de aplicações monolíticas. Adicionalmente, escrever testes automatizados envolvendo múltiplos serviços é uma tarefa complexa. Além disso, a AMS introduz dificuldades significantes de operação, como necessidade de alto nível de automação e ferramentas de monitoramento avançadas. Assim, uma equipe técnica altamente qualificada se torna necessária, podendo inclusive ser útil a contratação de consultorias especializadas nessa tecnologia (**??**).

Por outro lado, a Arquitetura de Microsserviços (AMS) engloba desafios para manter a consistência dos dados. Enquanto em um sistema monolítico, dados são usualmente armazenados em uma única base de dados relacional, que oferece suporte a transações ACID, em um sistema distribuído, no qual múltiplos banco de dados - possivelmente de tipos e fornecedores diferentes - são empregados, esse tipo de segurança não pode ser atingido facilmente (**??**). Transações distribuídas ou *Sagas* podem ser aplicadas para reduzir problemas de consistência, no entanto, essas técnicas adicionam grande complexidade ao sistema (**??**).

2.5 Programação Orientada a Objetos (POO)

A Programação Orientada a Objetos (POO) é um paradigma de desenvolvimento de software, que busca realizar a modelagem do mundo real utilizando, entre outras outros, os conceitos de classe e objeto (**??**). Além disso, trata-se de uma forma de se organizar aplicações através de uma coleção de objetos que colaboram entre si, cada um sendo uma instância de uma classe (**??**). Esse paradigma tem os seguintes objetivos:

- Promover a reutilização de código;
- Facilitar a manutenção do sistema;
- Tornar mais organizado e legível o código-fonte;

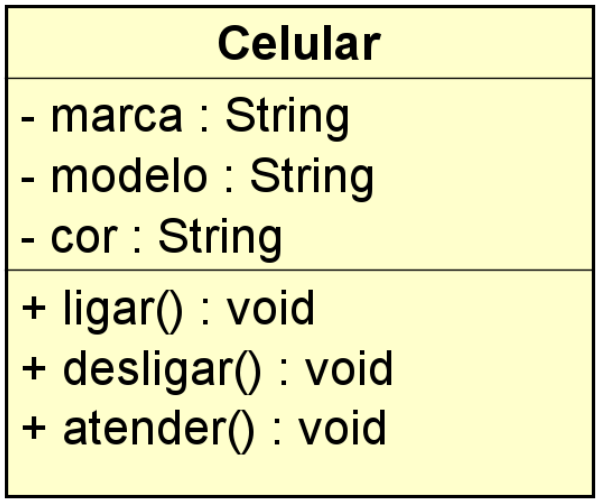
(**??**).

POO é frequentemente o paradigma escolhido para aplicar os conceitos de Domain-Driven Design (DDD). Em primeiro lugar, sistemas escritos no estilo procedural, por exemplo, possuem funções complexas unidas por caminhos de execução definidos antecipadamente, sem conexões conceituais com o domínio do negócio (**??**). Além disso, POO é poderosa, pois se baseia em modelagem e fornece implementações para os elementos de modelagem como, por exemplo, classes e interfaces (**??**). Há 4 pilares do paradigma que são descritos a seguir: abstração, encapsulamento, herança e polimorfismo (**??**).

2.5.1 Classe e Objeto

Uma classe por ser entendida como a implementação de um tipo abstrato de dado formado por propriedades, conhecidas como atributos, e operações, chamadas de métodos na terminologia POO (**??**). Pode-se definir, por exemplo, uma classe **Celular** composta pelos seguintes atributos: **marca**, **modelo**, **cor**. Como métodos, tem-se **ligar**, **desligar** e **atender**. Uma ilustração da classe **Celular** está disponível na Figura 1.

Figura 1 – Exemplo de classe



Fonte: o autor

Por outro lado, um objeto é considerado uma instância de uma classe, que possui valores próprios para os atributos definidos (**??**). Tomando como base o exemplo anterior, dois objetos possíveis para a classe **Celular** são um *IPhone* 14 branco, desenvolvido pela *Apple Inc*, e um *Galaxy* S21 azul da *Samsung Eletronics*.

2.5.2 Abstração

Um objeto no mundo físico geralmente possui infinitas características e comportamentos. Por esse motivo, é necessário que a representação computacional desse objeto (classe) simplifique essas definições conforme o contexto em que o objeto está inserido. Nesse sentido, abstração pode ser compreendida como uma seleção dos atributos e métodos relevantes para o sistema em questão (**??**).

2.5.3 Encapsulamento

Encapsulamento é uma técnica que consiste em esconder detalhes de implementação de um objeto (**??**). Ela é usualmente alcançada através da ocultação de informações. Além disso, encapsulamento fornece barreiras explícitas entre diferentes abstrações e, portanto, gera uma clara separação de responsabilidades (**??**).

2.5.4 Herança

Em sistemas não triviais, muitas abstrações estão presentes. Frequentemente, parte delas formam uma relação hierárquica, na qual um tipo abstrato estende outro tipo (usualmente chamado de supertipo). Assim, o conceito de herança no contexto de POO pode ser entendido como um ranqueamento ou ordenação de abstrações (**??**).

2.5.5 Polimorfismo

O polimorfismo é a possibilidade de subtipos possuírem uma implementação diferente para um método definido pelo supertipo. Ou seja, através da utilização deste recurso, abstrações filhas podem sobrescrever comportamentos da abstração pai (**??**).

??) afirma que polimorfismo talvez seja o recurso mais poderoso de linguagens orientadas a objetos, sendo uma das principais características que distingue POO de outros paradigmas de programação. Além disso, trata-se de um recurso que complementa a herança.

2.6 *Domain-Driven Design (DDD)*

Domain-Driven Design (DDD) é uma abordagem para o desenvolvimento de software focada na construção de um modelo de domínio que representa de maneira sofisticada os processos e regras de negócio do sistema em questão (**??**). DDD é muito efetivo para desenvolver sistemas com regras de negócio complexas, que possuem, por exemplo, diversos atores, entidades e validações.

Um ponto central da metodologia DDD no desenvolvimento de softwares complexos, é a utilização de uma linguagem ubíqua que insere terminologia e jargões do domínio em componentes de software como atributos, métodos, classes, interfaces e pacotes (**??**). Assim, tanto especialistas de negócio quanto desenvolvedores e projetistas compartilham um vocabulário comum que facilita na comunicação e reduz a necessidade de realizar o mapeamento entre termos técnicos e termos do domínio. Além disso, a estrutura de sistema passa a representar a estrutura do negócio, tornando-se, assim, mais fácil de ser compreendida e modificada.

O nome dessa estrategia vem de um livro publicado em **??** por Eric Evans que descreve a abordagem, uma série de conceitos e um catálogo de padrões comumente utilizados para resolução de problemas de modelagem de domínios. Os mais relevantes dentre estes são descritos a seguir.

2.6.1 Modelo e Domínio

Os conceitos de modelo e domínio, no contexto da engenharia de software, são fundamentais para o entendimento do DDD. Modelo é uma simplificação, uma interpretação da realidade que abstrai os aspectos relevantes para resolver o problema em questão e ignora detalhes superficiais (**??**). Da mesma forma, todo sistema computacional está relacionado a alguma atividade ou interesse do usuário final. Assim, a área para qual o usuário utiliza o sistema é domínio do software (**??**). Alguns exemplos de domínios comuns são: financeiro, médico, logística e *marketplace*.

2.6.2 Entidade

No contexto da Programação Orientada a Objetos (POO), muitos objetos não são fundamentalmente definidos por seus atributos, mas por um fio de continuidade e identidade (**??**). Eles podem ter distintas representações como, por exemplo, em memória ou em um banco de dados. No entanto, eles necessitam ser distinguíveis mesmo se todas suas propriedades forem iguais. Em um domínio financeiro, por exemplo, uma transação composta por valor, data e usuário necessita ser diferenciável de outra transação com as mesmas características. Por outro lado, para uma classe fictícia chamada *Money* composta por valor e moeda, dois objetos com os mesmos atributos podem ser geralmente considerados iguais, sem gerar corrupções de dados ou impactos para os usuários finais do sistema. Em resumo, quando um objeto definido principalmente por sua identidade, ele é chamado de Entidade (**??**).

2.6.3 *Value Object (VO)*

Muitos objetos não possuem identidade conceitual, sendo somente utilizados para descrever características de outro objeto. São objetos que descrevem coisas, classificados como Value Object (VO) (**??**). Para um sistema de uma loja de roupas, por exemplo, um objeto que representa cor de cada peça formado pelas propriedades *red*, *green* e *blue*, não possui identidade. Não necessita ser diferenciável de outro objeto com as mesmas propriedades. Este objeto somente descreve uma característica de uma peça de roupa. É importante ressaltar, porém, que a diferenciação entre Entidade e *Value Object* depende do contexto do software. Para um sistema de software de computação gráfica, a cor é um conceito chave no sistema e pode possuir identidade.

Pode-se pensar que é mais vantajoso sempre projetar os objetos do sistema como Entidade, pois adicionaria identidade, facilitaria o rastreamento de mudanças e agregaria mais recursos ao objeto. No entanto, a adição de identidade para objetos que não necessitam desta classificação pode prejudicar o desempenho do sistema, adicionar mais esforço de análise e aumentar a complexidade do modelo de domínio (**??**). Por esse motivo,

recomenda-se a utilização de VO sempre que for possível, pois se tratam de objetos mais fáceis de gerenciar.

2.6.4 *Domain Event*

2.6.5 Serviço

Em muitos casos, há operações importantes no modelo que não pertencem conceitualmente a nenhuma Entidade ou VO. Dessa forma, surge o conceito de Serviço: uma operação fornecida como uma interface autônoma no modelo, sem encapsular estado como outros tipos de objeto (**??**).

Um erro comum é a abusar da utilização de Serviços, desistindo de encontrar o objeto apropriado para uma operação de domínio. Da mesma forma, a inserção de um método que não está alinhado com a definição do objeto resulta na perda de coesão desse objeto, tornando-o mais difícil de compreender e refatorar (**??**). Considerando esses pontos, um serviço deve ser utilizado quando um processo ou transformação relevante no modelo de domínio não faz parte da responsabilidade natural de uma Entidade ou VO.

Além disso, **??**) destaca três características de um bom Serviço:

- A operação não possui estado (*stateless*).
- A interface é defina em termos de outros elementos do modelo de domínio.
- A operação se relaciona a um conceito do modelo que não é parte natural de uma Entidade ou VO.

2.6.6 *Aggregate*

Um *Aggregate* é um conjunto de objetos associados, tratados como uma única unidade para o proposito de alterações de dados (**??**). Em um sistema de grande porte, é difícil garantir a consistência de alterações em objetos com relacionamentos complexos. As invariantes que precisam ser mantidas geralmente envolvem um grupo de objetos intimamente associados, ao invés de objetos isolados. Assim, o padrão *Aggregate* propõe delimitações definidas e restrições de acesso visando o melhor tratamento das associações entre objetos.

Cada *Aggregate* possui uma raiz e uma delimitação, que define o que está dentro do *Aggregate* (**??**). A raiz é a única entidade global - que possui identidade em todo o sistema - contida dentro do *Aggregate*, também conhecida como *Aggregate root* (**??**). Objetos fora da delimitação do *Aggregate* só podem possuir referências a raiz e não a outros objetos dentro do *Aggregate*. Excluindo a raiz, outras entidades somente possuem identidade local,

necessitando somente ser distinguíveis dentro dos limites do *Aggregate*, visto que nenhum objeto de fora consegue obtê-las desassociadas da entidade raiz (**??**).

2.6.7 Bounded Context (BC)

Em grandes projetos, muitos modelos coexistem em diferentes contextos. Quando a base de código, no entanto, combina diferentes modelos sem que haja uma separação clara, o *software* se torna defeituoso, frágil e difícil de ser compreendido (**??**). Assim, é necessário especificar em qual o contexto um determinado modelo está sendo aplicado. Assim, um *Bounded Context (BC)* delimita a aplicabilidade de um modelo, de tal forma que os membros da equipe tenham um entendimento claro do que precisa ser consistente e a relação do modelo com outros contextos. Um BC engloba tanto componentes de modelo como *Aggregates*, Entidades, *Value Objects* e Serviços, quanto detalhes técnicos como persistência. É uma parte completa e funcional de software sob responsabilidade de uma única equipe.

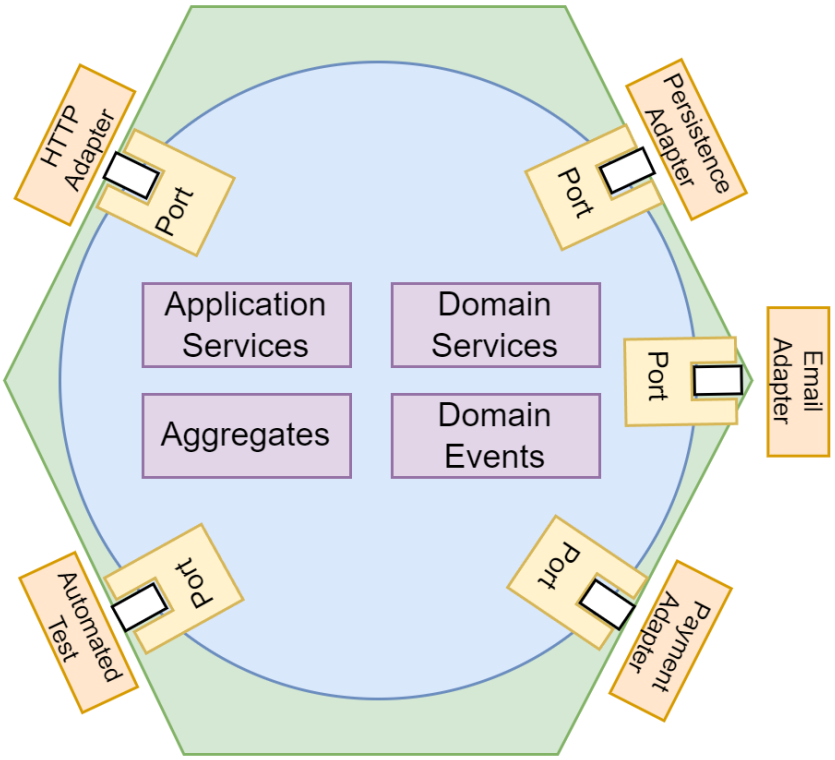
Por exemplo, um *software* para um *e-commerce* possui um módulo que lida com pedidos, em que cada pedido contém um ou mais produtos. Da mesma forma, outra equipe é responsável por um módulo de realiza o gerenciamento de produtos em estoque. O conceito de um produto para um sistema de pedidos é distinto de uma aplicação de estoque. Enquanto este pode ter um enfoque no armazém, na categoria e no estoque do produto, aquele está mais preocupado com o preço, nome e disponibilidade. Assim, ao tratar esses dois conceitos da maneira igual, confusões e defeitos são gerados.

2.7 Arquitetura Hexagonal

A arquitetura hexagonal, também conhecida como *Ports and Adapters*, possui como objetivo isolar regras de negócio de agentes externos (**??**). Nesse sentido, o que está dentro do hexágono representa o modelo de domínio. Do mesmo modo, componentes de fora representam detalhes de implementação, como persistência, interface de usuário, envio de *e-mails*, entre outros. Um ponto crucial nessa arquitetura é a direção das dependências: o que está dentro do hexágono não pode depender do que está fora. Assim, é a garantida a independência e o isolamento de regras de negócio das dependências técnicas.

Na Figura 2, pode-se observar que a injeção de agentes externos, como persistência em um banco de dados, é realizada por meio de portas definidas dentro do núcleo da aplicação. As portas, nessa arquitetura, são geralmente representadas por interfaces em linguagens orientadas a objetos. Então, um adaptador fornece uma implementação de uma tecnologia específica para a porta. Assim, implementações para tecnologias específicas são definidas em função do modelo de domínio.

Figura 2 – Arquitetura Hexagonal



Fonte: o autor

Outros benefícios importantes, além do isolamento das regras de negócio, são: facilidade de testar o núcleo da aplicação com a utilização de *mocks* e *stubs*; possibilidade de utilização da mesma aplicação via interfaces de usuário, testes automatizados e *batches*.

Enquanto a Arquitetura de Microsserviços (AMS) visa organizar um sistema como um conjunto de serviços independentes, cada um executando uma função específica, a arquitetura hexagonal concentra-se na organização interna de um serviço, propondo uma separação clara entre a lógica de negócios e as implementações técnicas, usando portas e adaptadores. Por outro lado, DDD apresenta-se como uma excelente abordagem para desenvolvimento do núcleo da aplicação na arquitetura hexagonal. Assim, esses três conceitos: microsserviços, arquitetura hexagonal e DDD podem ser combinados para o desenvolvimento de sistemas altamente coesos, desacoplados, escaláveis, inteligíveis e sustentáveis.

3 Trabalhos Relacionados

Este capítulo apresenta os trabalhos relacionados ao objeto de pesquisa obtidos utilizando o protocolo descrito no Capítulo 4.

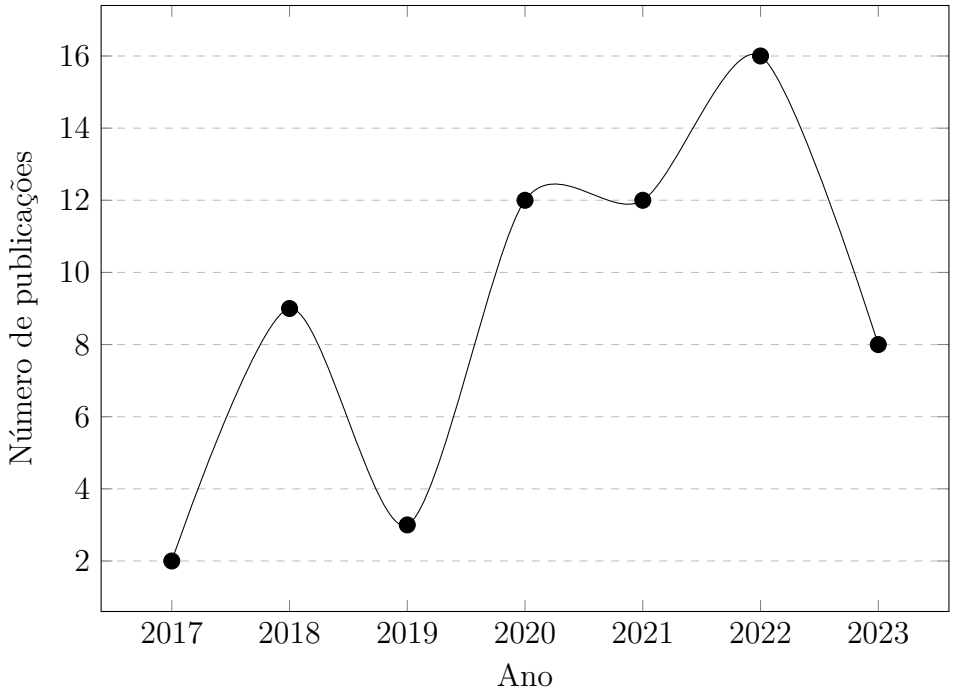
3.1 Caracterização dos estudos

Na Figura 3, pode-se observar a quantidade de publicações por ano, encontradas a partir da expressão de busca descrita na seção 4.2.5.4. No total, foram obtidos 65 resultados. Percebe-se, claramente, um aumento exponencial no número de publicações sobre o tema, principalmente a partir de 2019, demonstrando crescimento no interesse de se realizar pesquisas sobre o assunto.

A Alemanha lidera em número de publicações, como pode ser visualizado na Figura 4. Por outro lado, o Brasil só possui duas entre as sessenta e cinco publicações retornadas.

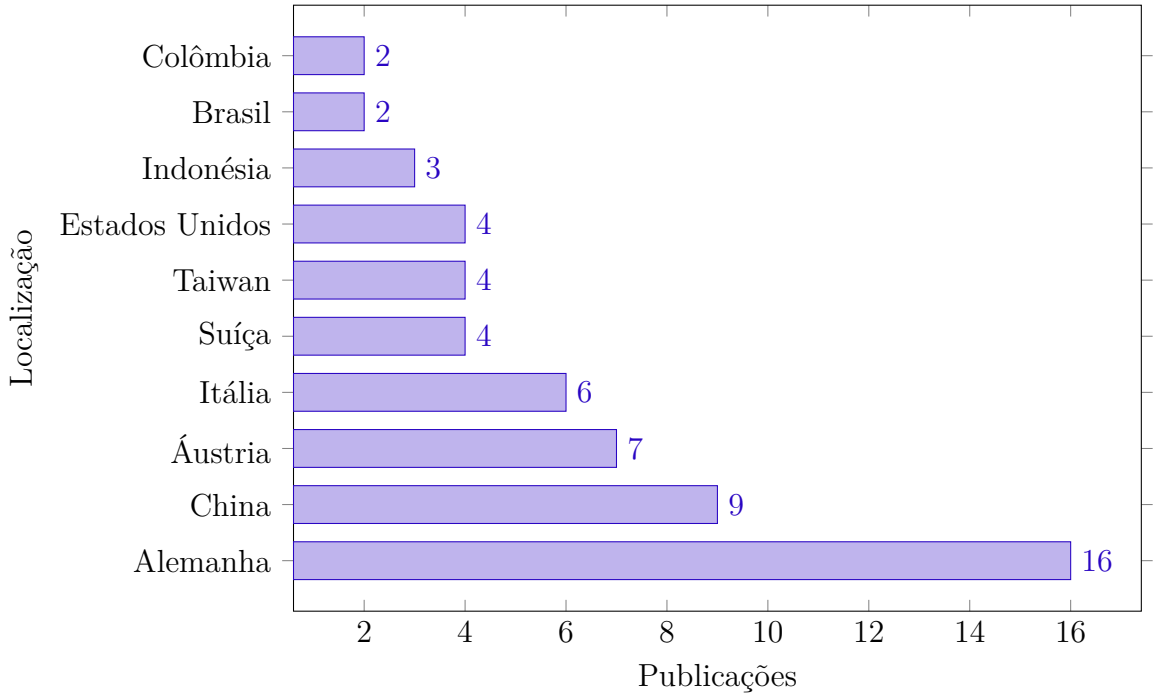
Utilizando-se os critérios de inclusão e exclusão descritos nas seções 4.2.6.1 e 4.2.6.2, respectivamente, dos 65 artigos retornados, 27 foram selecionados inicialmente. Após realizada as leituras, apenas 16 publicações mostraram-se relevantes para responder às

Figura 3 – Número de publicações ao longo dos anos



Fonte: *Scopus*

Figura 4 – Publicações por localização



Fonte: *Scopus*

questões de pesquisa e/ou apoiar na elaboração do estudo de caso. Esta listagem está disponível no Quadro 5.

3.2 Estratégias para elaboração de sistemas com microsserviços e DDD

Todas as publicações revisadas sugerem o mapeamento de cada Bounded Context (BC) para um microsserviço. **??**) afirmam que o objetivo principal da delimitação é alcançar serviços com baixo acoplamento e alta coesão. Porém, o grande desafio está na definição de maneira apropriada do escopo de cada BC.

??) e **??**) mencionam os padrões *Open Host Service (OHS)* e *Anti-Corruption Layer (ACL)* como estratégias para comunicação, conceitualmente, entre Bounded Contexts. No OHS, o serviço que envia as mensagens implementa uma camada extra com objetivo de realizar a tradução para um formato que possa ser processado pelo serviço receptor. Dessa forma, os detalhes de implementação do serviço cliente não são expostos, diminuindo o acoplamento entre as partes. Semelhantemente, o ACL é uma camada extra inserida no serviço que recebe as mensagens. Trata-se de uma abordagem útil quando o serviço receptor não deseja aderir ao contrato do componente que produz as mensagens (**??**).

Outro ponto importante levantando no material revisado é o mapeamento do

Título	**Ano**
Microservice Migration Using Strangler Fig Pattern and Domain-Driven Design	??
Microservice architecture and model-driven development: Yet singles, Soon Married ?	??
Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?	??
Modeling Microservices with DDD	??
Following Domain Driven Design principles for Microservices decomposition: Is it enough?	??
An Ontology-based Approach for Domain-driven Design of Microservice Architectures	??
Challenges of domain-driven microservice design: A model-driven perspective	??
Model-based engineering for microservice architectures using Enterprise Integration	??
Patterns on Deriving APIs and their Endpoints from Domain Models	??
Refactoring with domain-driven design in an industrial context: An action research report	??
A microservice based reference architecture model in the context of enterprise architecture	??
Design of Domain-driven Microservices-based Software Talent Evaluation and Recommendation System	??
Partitioning microservices: A domain engineering approach	??
Practitioner Views on the Interrelation of Microservice APIs and Domain-Driven Design: A Grey Literature Study Based on Grounded Theory	??
A Systematic Framework of Application Modernization to Microservice based Architecture	??
Domain-specific language and tools for strategic domain-driven design, context mapping and bounded context modeling	??

Quadro 5 – Publicações selecionadas

modelo de domínio para uma Application Programming Interface (API) para possibilitar o consumo das funcionalidades tanto por outros serviços quanto de aplicações cliente. Algumas opções citadas por **??**) são:

- Expor todo o modelo de domínio em uma relação de 1 para 1 com a API.
- Expor uma parte do modelo como uma API.
- Expor cada BC como uma API.
- Expor parte dos BCs como APIs.

As alternativas mais indicadas são a exposição total ou de grupo de BCs como APIs. Além disso, **??**) e **??**) levantam abordagens para definição do contrato de API de acordo com modelo de domínio. As duas principais opões apresentadas pelos autores são: explicitamente definir o contrato de API e extrair o contrato de API a partir do modelo. **??**) argumenta que ambas opções auxiliam na obtenção da separação do contrato de API e das responsabilidades do modelo de domínio.

??) apresenta uma abordagem para DDD no contexto da AMS. O autor propõe um metamodelo que representa a sintaxe abstrata de um linguagem formal de modelagem. Em outras palavras, ele define os conceitos suportados pela linguagem e seus relacionamentos. Os principais componentes do metamodelo são: *External Context*, *Bounded Context*, *Domain Model*, *Attribute* e *Association*. O objetivo principal desse modelo é minimizar os impactos dos desafios semânticos da utilização dessas tecnologias (**??**).

3.3 Desafios na utilização de DDD como estratégia de delimitação de microsserviços

??) menciona três principais desafios da utilização de DDD no contexto da AMS. Adicionalmente, **??**) cita desafios semânticos juntamente com sua proposta de metamodelo.

3.3.1 Extraindo microsserviços a partir de modelos de domínio

Para fomentar o foco em conceitos relevantes e *design* efetivo, modelos de domínio tipicamente omitem informações obrigatórias para extração de microsserviços, como:

- Interfaces e operações;
- Parâmetros e tipos de retorno das operações
- *Endpoints*, protocolos e formatos de mensagens.

(**??**).

Informações específicas sobre esses pontos são cruciais para implementação dos serviços. Considerando diferentes BCs conectados entre si no modelo, e que cada BC é mapeado para um microsserviço, um componente vai necessitar acessar instâncias de um outro serviço. Assim, uma operação desse tipo deve ser fornecida e usualmente não é especificada no modelo, deixando espaço para ambiguidade (**??**).

3.3.2 Componentes de infraestrutura faltantes no modelo de domínio

Modelos de domínio intencionalmente não compreendem componentes de infraestrutura da AMS (**??**). Esses componentes incluem *API Gateways*, *containers*, banco de dados, entre outros. Por outro lado, requisitos do modelo podem afetar questões técnicas e esses pontos devem ser documentados separadamente do modelo de domínio (**??**). Por exemplo, caso um microsserviço que realize o gerenciamento de usuários necessite ser acessível externamente para faturamento, configurações em diferentes componentes técnicos como *API Gateway* necessitarão ser aplicadas.

3.3.3 Modelagem de domínio autônoma

Responsabilidade sobre um microsserviço é geralmente atribuída a um único time, devido à alta coesão e baixo acoplamento (**??**). Essa equipe é responsável pela implementação do serviço, operação, *design* e manutenção desse Bounded Context (BC). Dessa forma, surgem desafios relacionados a visibilidade dos modelos e gerenciamento de alterações.

Primeiramente, é crucial definir a visibilidade de cada BC, ou seja, especificar quais equipes terão acesso a quais modelos de domínios (**??**). Além disso, a permissão para realizar alterações é uma consideração essencial. Embora seja possível conceder a outras equipes privilégios para modificar outros BC, essa abordagem apresenta a desvantagem de possibilitar alterações, por vezes críticas, efetuadas por profissionais que não estão familiarizados com o contexto específico. No entanto, restringir exclusivamente à equipe responsável a capacidade de realizar mudanças pode resultar em gargalos significativos, especialmente em projetos envolvendo diversos contextos.

3.3.4 Desafios semânticos

??) apresenta uma série de desafios semânticos na elaboração de microsserviços com DDD. Inicialmente, um problema de semântico típico ocorre quando um atributo de um conceito de domínio é derivado de outro atributo. Por exemplo, quando um atributo de uma entidade é criado a partir da concatenação de dois outros atributos de outra entidade, ocorre um problema de semântica porque os dados ficam fragmentados.

Simultaneamente, é importante reconhecer que diferentes BCs podem interpretar os conceitos de domínio compartilhados de maneiras distintas (**??**). Nesse contexto, surgem desafios como atributos com nomes diversos, mas significados idênticos, bem como propriedades com o mesmo nome, porém, com significados diferentes. Esse risco é amplificado no contexto da AMS, onde é comum que diferentes BCs sejam desenvolvidos por equipes distintas. Esse ambiente descentralizado pode potencializar a disparidade de interpretações e a falta de consistência nos conceitos de modelos compartilhados.

Além disso, um mecanismo de definição de identificadores únicos deve ser definido entre as equipes para permitir a identificação semântica de diferentes conceitos do modelo com objetivo de tornar os serviços capazes de distinguir diferentes elementos em um contexto distribuído (**??**).

3.4 Anti-padrões a serem evitados

??), **??**), **??**), **??**) e **??**) identificaram uma série de anti-padrões no contexto da AMS e DDD. Um compilado pode ser observado no Quadro 6.

3.5 Discussões

O mapeamento da literatura mostrou-se efetivo para responder às questões de pesquisa, na medida em que foram encontradas publicações que auxiliam na resolução das perguntas-chave. O levantamento das estratégias, desafios e anti-padrões foi realizado com êxito a partir de artigos de alta qualidade.

No entanto, publicações em outros idiomas que não foram revisadas, assim como publicações não indexadas pelas ferramentas de busca utilizadas, representam uma ameaça a validade da pesquisa. Isso se deve ao fato de que informações essenciais podem não ter sido revisadas devido a essas limitações

Por fim, as informações obtidas nesse mapeamento contribuem tanto para atingir os objetivos gerais da pesquisa quanto para a elaboração do estudo de caso.

Quadro 6 – Anti-padrões

Nome	**Descrição**
Chattiness of a service	Refere-se a excessiva comunicação entre microsserviços que gera ineficiência devido à latência de rede.
Nanoservice	Um serviço excessivamente granular no qual a sobrecarga de comunicação, manutenção e operação supera sua utilidade.
Anemic domain model	Trata-se de um modelo de domínio em que os objetos contém pouca ou nenhuma regra de negócio. As invariantes são misturadas com outras lógicas, o que dificulta manutenção e refatoração.
Data class	Similar ao *Anemic domain model*, esse tipo de classe só contem, além de atributos, *getters* e *setters*. Os comportamentos são criados fora da classe, o que reduz coesão e dificulta a manutenção.
Distributed monoliths	Refere-se a um sistema que externamente se assemelha a Arquitetura de Microsserviços, porém possui um alto nível de acoplamento entre os componentes, diminuindo assim as vantagens dos microsserviços.
Start with API Design	Trata-se de uma abordagem na qual o contrato de API é definido antes do modelo de domínio. **??**) levantou que esta estratégia costuma gerar a criação de *Anemic domain models*.
Feature envy	Acontece quando um método está mais interessado em uma classe diferente da que está inserido.
Inappropiate intimacy	Descreve um par de classes não relacionadas conceitualmente, mas que possuem grande acoplamento entre si.
Message chain	Uma cadeia de mensagem ocorre quando um cliente envia uma mensagem a outro objeto que, por sua vez, a envia outro o objeto e assim por diante.

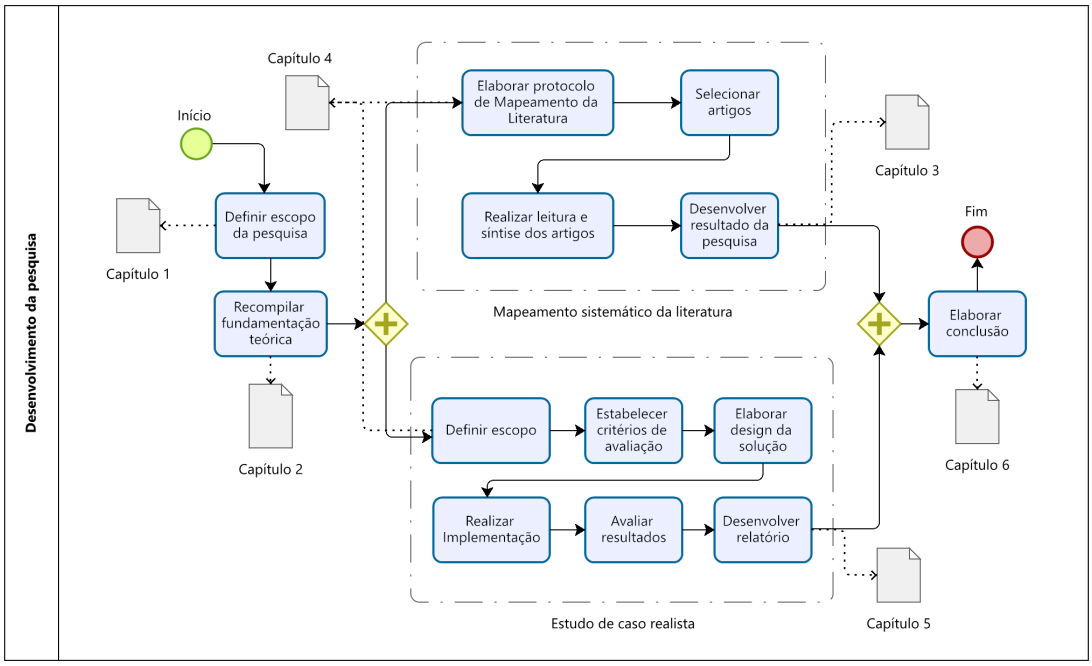
Fonte: **??**), **??**), **??**), **??**) e **??**)

4 Metodologia

Este capítulo apresenta a metodologia e os recursos utilizados para atingir o objetivo do trabalho.

4.1 Visão geral

Figura 5 – Etapas de desenvolvimento da pesquisa



Fonte: o autor

Pode-se observar na Figura 5 as etapas de execução dessa pesquisa. Inicialmente, o escopo é definido e primeiro capítulo é elaborado. Em seguida, a fundamentação teórica com os conceitos-chave é construída. Posteriormente, se realiza um mapeamento da literatura buscando trabalhos similares. Adicionalmente, um estudo de caso realista com a utilização de microsserviços e diversos conceitos do DDD é desenvolvido e um relatório é produzido. Por fim, é escrita a conclusão do trabalho.

4.2 Mapeamento sistemático da Literatura

Esta seção apresenta o protocolo de mapeamento sistemático da literatura usado para atingir os objetivos da pesquisa.

4.2.1 Problema de pesquisa e contexto

Nos últimos anos, ocorreu uma mudança paradigmática no desenvolvimento de software (**??**), impulsionada pela necessidade de escalabilidade, flexibilidade e agilidade. Nesse cenário, a arquitetura de microsserviços emergiu como abordagem para construção de grandes sistemas formado por partes independentes e interconectadas entre si. No entanto, uma dos principais desafios dessa arquitetura é a delimitação do escopo de cada serviço (**??**). Assim, Domain-Driven Design (DDD) se apresenta como uma importante alternativa para enfrentar essa questão. Assim, esse mapeamento visa compreender como esses dois conceitos estão sendo combinados para potencializar o desenvolvimento de software.

4.2.2 Objetivo

Este mapeamento da literatura visa encontrar artigos contendo estudos de caso, padrões, recomendações e análises da utilização combinada da arquitetura de microsserviços e DDD. Especificamente sobre os estudos de caso, esta pesquisa está interessada tanto em casos de construção de sistemas de zero, quanto de migração de monólitos existentes.

4.2.3 Justificativa

Com esse mapeamento da literatura, pretende-se fornecer uma análise valiosa para arquitetos de software, desenvolvedores e pesquisadores sobre os benefícios e desafios da integração de microsserviços com DDD. Além disso, o conhecimento obtido com esse mapeamento auxiliará na elaboração do estudo de caso realista.

4.2.4 Questões de pesquisa

Q1: Quais são as principais estratégias para elaboração de sistemas com microsserviços e DDD?

Q2: Quais são os principais desafios na utilização de DDD como estratégia de delimitação de microsserviços?

Q2: Quais são os anti-padrões a serem evitados na utilização de DDD com microsserviços?

4.2.5 Estratégia de busca

Esta seção apresenta a estratégia de buscas de artigos científicos e livros relacionados à pesquisa

4.2.5.1 Ferramentas

Periódicos Capes: É uma ferramenta disponibilizada pelo governo federal para uso de estudantes e pesquisadores. Acessando através da instituição de ensino ou pesquisa, é possível ter acesso completo a uma grande quantidade de artigos científicos publicados em variadas revistas, conferências e universidades. A principal vantagem dessa ferramenta é a possibilidade de ler o conteúdo integral de grande parte das publicações disponíveis. Por outro lado, as expressões de busca atualmente suportadas são bem limitadas.

Scopus: Trata-se de um ferramenta similar ao Periódicos Capes. No entanto, o *Scopus* permite a elaboração de expressões de buscas mais complexas e sofisticadas, servindo para descobrir publicações não detectadas pelas outras plataformas. Além disso, possui um acervo bem mais amplo que o Periódicos Capes. Entretanto, algumas publicações não podem ser vistas na íntegra de forma gratuita.

Google Docs: Ferramenta desenvolvida pela *Google LLC* que permite a criação e edição de documentos de texto. Suas grandes vantagens em relação a ferramentas de outros fornecedores são as avançadas ferramentas de colaboração e a possibilidade de acesso por meio de navegadores *web*, sem necessidade de instalação de *software* específico.

Google Sheets: Com as mesmas características e vantagens do *Google Docs*, essa ferramenta fornece recursos para elaboração de planilhas de cálculo. É muito útil para realizar análise de dados simples e também visualizar e apresentar dados tabulares.

4.2.5.2 Idioma e período

Como grande parte das publicações na área de computação são em inglês, esta pesquisa utiliza esse idioma para fazer buscas nas ferramentas indicadas.

Além disso, Arquitetura de Microsserviços (AMS) e Domain-Driven Design (DDD) são relativamente recentes, as buscas se limitaram a publicações feitas nos últimos 20 anos.

4.2.5.3 Termos ou palavras chaves

Os termos-chave para realização das buscas são: Microsserviço, DDD e Domain-Driven Design. Como a busca será feita em inglês, se usará *microservice* nas buscas.

4.2.5.4 Expressão de busca

Quadro 7 – Expressão de busca utilizada

Expressão de Busca
((TITLE-ABS-KEY (microservice) AND TITLE-ABS-KEY (domain-driven AND design)) OR (TITLE-ABS-KEY (microservice) AND TITLE-ABS-KEY (ddd)))

Fonte: o autor

No Quadro 7, percebe-se que a expressão de busca pretende retornar todas as publicações que contenham as palavras chaves no título, resumo ou na seção de *keywords*.

4.2.6 Estratégia de seleção

A seguir são apresentados critérios para inclusão e exclusão de publicações na pesquisa. Além disso, características para analisar os materiais quanto a qualidade são discutidas.

4.2.6.1 Critérios de inclusão

- Publicações revisadas por pares.
- Texto completo disponível de forma gratuita pelo portal Periódicos Capes.
- Materiais relacionados ao tópico de interesse, ou seja, título ou resumo.
- permitem identificar o trabalho como relevante para responder às questões postuladas.

4.2.6.2 Critérios de exclusão

- Publicações duplicadas.
- Materiais que não dispõem de informação relevante para responder às questões de pesquisa.
- Publicação com Índice de qualidade insuficiente, ou seja, C5 <= 3, conforme critérios para avaliar qualidade.

4.2.6.3 Critérios de qualidade

- Descrição clara do problema de pesquisa e seu contexto: 2 pontos.
- Descrição clara dos objetivos da pesquisa: 2 pontos.
- Descrição clara dos métodos usados no desenvolvimento: 1 ponto.

4.2.7 Estratégia para extração de dados e análise

Para atingir o objetivo do mapeamento da literatura, são filtrados manualmente nos artigos selecionados segundo os critérios de inclusão e exclusão. A partir da listagem reduzida, todas as publicações são lidas de forma integral. Adicionalmente, todos os gráficos e tabelas nos artigos selecionados são avaliados visando extrair algum dado que permita realizar a comparação entre aspectos quantitativos das estratégias como, tempo de resposta, latência e taxa de transferência.

4.2.7.1 Instrumentos de coleta de dados

Informações qualitativas como recomendações, destaques, conceitos e estudos de casos são registrados em um documento no *Google Docs*.

Dados quantitativos como taxa de transferência, latência, tempo de processamento são armazenados em uma planilha no *Google Sheets*.

4.2.7.2 Procedimentos

A pesquisa é executada seguindo os passos a seguir:

1. As expressões de busca citadas são inseridas nas ferramentas mencionadas.
2. É feito o armazenamento das publicações retornadas em uma planilha de cálculo.
3. As publicações retornadas são filtradas conforme os critérios de inclusão e exclusão.
4. Em cada artigo selecionado, é realizada a extração dos dados relevantes para responder às questões de pesquisa.
5. Finalmente, são produzidas respostas para questões de pequisa com as informações extraídas das publicações.

4.3 Estudo de Caso

A FAZER

4.4 Ferramentas Utilizadas na elaboração da Monografia

A FAZER

- Overleaf
- ChatGPT: Refraseamento

- LanguageTool
- Bizagi
- Asta

5 Desenvolvimento

Neste capítulo, é apresentado as etapas de desenvolvimento do projeto.

5.1 Levantamento das Estórias

As estórias foram identificadas, analisadas e descritas abaixo:

- **Estória 01: Cadastrar Catador**

  Descrição: Permite a criação do usuário responsável pelo recolhimento do material descartado.

- **Estória 02: Cadastrar Separador**

  Descrição: Permite a criação do usuário responsável pelo descarte do material.

- **Estória 03: Efetuar login**

  Descrição: Autoriza o acesso às funcionalidades do sistema.

- **Estória 04: Recuperar senha**

  Descrição: Possibilita que o usuário redefina a senha.

- **Estória 05: Tirar dúvidas**

  Descrição: Esclarece sobre a definição dos perfis.

- **Estória 06: Solicitar recolhimento**

  Descrição: Requisita a coleta do material descartado.

- **Estória 07: Realizar logoff**

  Descrição: Permite que o usuário saia do sistema.

- **Estória 08: Lembrar login**

  Descrição: Permite que os dados de autenticação do usuário, fiquem salvos na tela de *login*.

- **Estória 9: Traçar rota da coleta**

  Descrição: Possibilita que o usuário visualize no mapa a duração e a distância do trajeto a ser percorrido.

- **Estória 10: Visualizar mapa**

  Descrição: Facilitar a visualização da localização.

- **Estória 11: Escolher tipo de material**

  Descrição: Informar qual tipo de material reciclável será descartado.

- **Estória 12: Informar localização**

  Descrição: Informar a localização para o descarte do material.

5.2 Diagramas do Modelo Proposto

5.2.1 Diagrama de Classes

Na construção do diagrama de classes foi analisada as principais classes do sistema, suas características, seus relacionamentos e suas funcionalidades. É possível observar através da Figura 6, a estrutura do desenvolvimento do sistema.

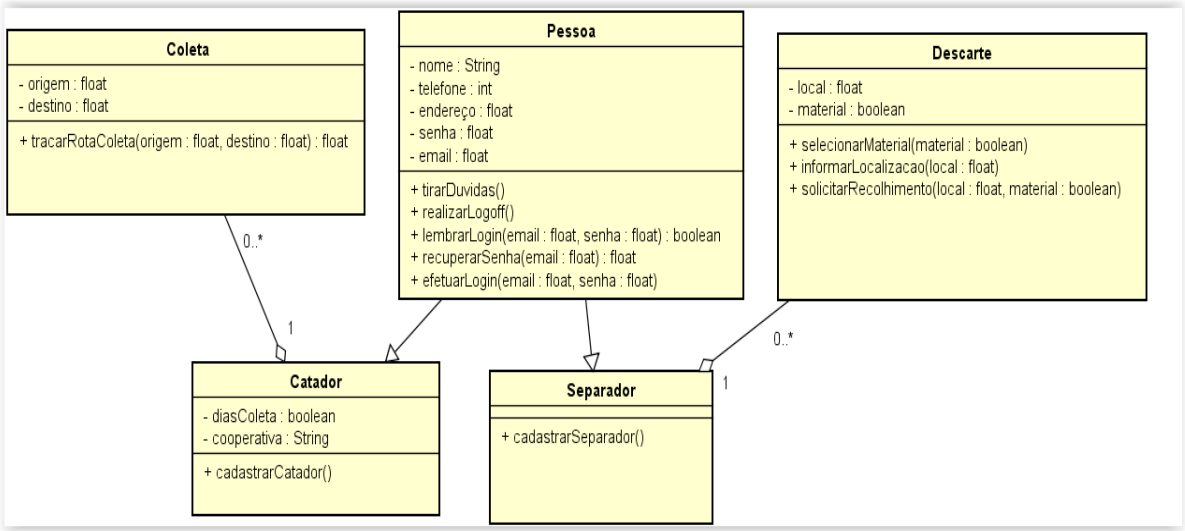


Figura 6 – Diagrama de Classes

Na Figura 6 citada acima podemos observar o Diagrama de Classes utilizado para a construção do aplicativo.

As classes Catador e Separador são extensões da classe Pessoa, que tem como objetivo informar seus dados, tirar dúvidas sobre a definição dos perfis, criar uma conta perfil, redefinir a senha esquecida, efetuar *login*, salvar os dados do *login* e realizar *logoff*.

A classe Catador que faz parte de uma das contas perfil, é responsável pelo cadastro do usuário catador. Este usuário é encarregado por traçar a rota da coleta e recolher o material descartado.

A classe Separador que faz parte de uma das contas perfil, é responsável pelo cadastro do usuário separador. Ao informar sua localização e o tipo de material, o usuário poderá solicitar o recolhimento do material descartado.

5.2.2 Diagrama de Casos de Uso

Esse parágrafo descreve as funcionalidades do sistema e as interações entre os atores, através da Figura 7.

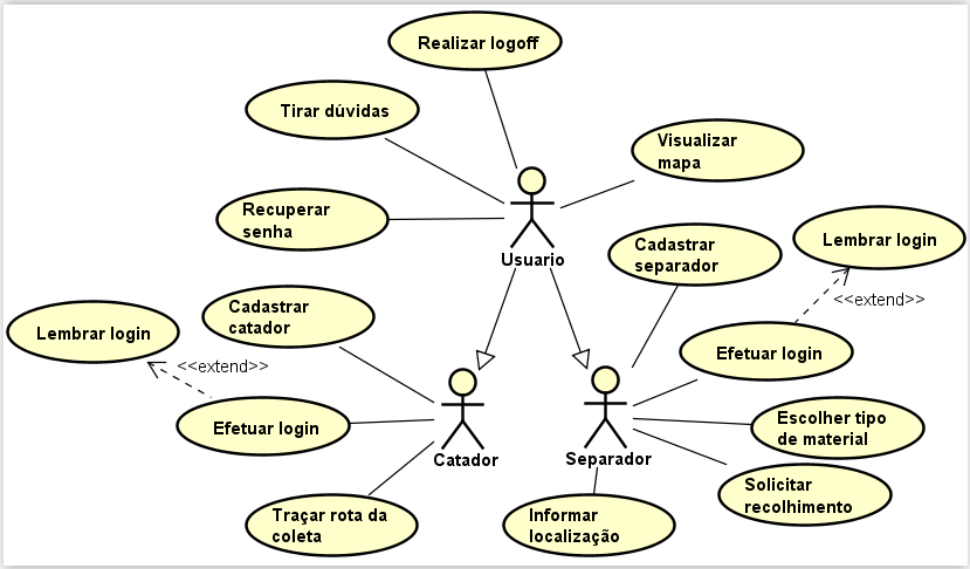


Figura 7 – Diagrama de Casos de Uso

Na Figura 7 citada acima, podemos ver o Diagrama de Casos de Uso. Na tela inicial do aplicativo, o usuário poderá realizar as seguintes funções: logar no APP, cadastrar usuário, tirar dúvidas, efetuar *logoff* e recuperar a senha. Se o usuário ainda não estiver cadastrado, irá se cadastrar no perfil desejado: Catador ou Separador. Caso já tenha cadastro, basta efetuar o *login*, o qual pode ser salvo pelo sistema. Em caso de senha esquecida, o sistema permite a recuperação através do *e-mail*.

A partir da inserção da localização e do tipo de material, o Separador poderá solicitar o recolhimento do rejeito. O Catador recebendo a solicitação do Separador, irá traçar a rota para a coleta do material.

5.2.2.1 Especificação dos Casos de Uso

Esse capítulo é responsável pelo detalhamento das funcionalidades do sistema. O (Quadro 8), descreve o passo a passo para o cadastro do usuário Catador.

Quadro 8 – Cadastrar Catador

Caso de Uso	Cadastrar Catador
Descrição	Permite a criação de perfil usuário denominado catador.
Ator	Catador
Pré-condições	Preencher todos os dados solicitados
Pós-condições	Cadastro do perfil catador no sistema
Fluxo Principal	1. Na tela inicial, o usuário solicita o cadastro; 2. Em seguida preenche todos os dados; 3. O sistema valida os dados preenchidos; 4. O cadastro é realizado.
Fluxo de Exceção	**Dados incorretos** 1. No passo 3 do Fluxo Principal, o usuário não preencheu os dados corretamente, o sistema sinaliza qual campo não foi preenchido. **Dados não cadastrados** 1. No passo 3 do Fluxo Principal, o usuário preenche algum campo já cadastrado, o sistema exibe qual campo já foi cadastrado.

O caso de uso Cadastrar Catador (Quadro 8), citado acima é responsável por avaliar as ações necessárias para realizar o cadastro do perfil catador.

O Quadro 9 descreve o passo a passo para o cadastro do usuário Separador.

Quadro 9 – Cadastro Separador

Caso de Uso	Cadastrar Separador
Descrição	Permite a criação de perfil usuário denominado separador.
Ator	Separador
Pré-condições	Preencher todos os dados solicitados
Pós-condições	Cadastro do perfil separador no sistema
Fluxo Principal	1. Na tela inicial, o usuário solicita o cadastro; 2. Em seguida, preenche todos os dados; 3. O sistema verifica os dados; 4. O cadastro é realizado.

O caso de uso Cadastrar Separador (Quadro 9), citado acima é responsável por avaliar as ações necessárias para realizar o cadastro do perfil separador.

O Quadro 10 descreve o passo a passo para efetuar o *login*.

Quadro 10 – Efetuar Login

Caso de Uso	Efetuar *login*
Descrição	Permite acesso às funcionalidades do sistema.
Ator	Catador e Separador
Pré-condições	O usuário deve estar cadastrado no sistema.
Pós-condições	O usuário estará logado no sistema.
Fluxo Principal	1. Na tela inicial o usuário, informa o *e-mail* e senha; 2. Em seguida realiza o *login*; 3. O sistema verifica os dados; 4. O usuário acessa a aplicação.
Fluxo de Exceção	**Usuário ou senha inválidos** 1. No passo 1, o usuário informa *e-mail* e/ou senha não cadastrados; 2. O sistema informa que os dados estão incorretos. **Campo nulo** 1. No passo 1, o usuário não preenche algum campo; 2. O sistema informa qual campo não foi preenchido; 3. Retorna para o passo 1.

O caso de uso Efetuar Login (Quadro 10), citado acima é responsável por avaliar as ações necessárias para realizar o *login* do usuário.

O Quadro 11 descreve o passo a passo para a recuperação da senha.

Quadro 11 – Recuperar Senha

Caso de Uso	Recuperar senha
Descrição	Redefinir senha
Ator	Separador e Catador
Pré-condições	O usuário deve estar cadastrado no sistema.
Pós-condições	A senha será recuperada após o preenchimento do *e-mail*.
Cenário Principal	1. O usuário solicita a recuperação da senha; 2. Em seguida, informa o *e-mail*; 3. Um *e-mail* para a redefinição da senha é enviado para o usuário.

O caso de uso Recuperar Senha (Quadro 11), citado acima é responsável por avaliar as ações necessárias para realizar a redefinição da senha esquecida pelo usuário.

O Quadro 12 descreve o passo a passo para tirar dúvidas.

Quadro 12 – Tirar Dúvidas

Caso de Uso	Tirar dúvidas
Descrição	Explicar a definição dos perfis.
Ator	Separador e Catador
Pré-condições	O usuário não precisa ser cadastrado ou estar logado no sistema.
Pós-condições	O sistema deve ser iniciado.
Cenário Principal	1. O usuário clica no ícone de dúvidas; 2. O sistema redireciona o usuário para a tela informativa.

O caso de uso Tirar Dúvidas (Quadro 12), citado acima é responsável por avaliar as ações necessárias para esclarecer a definição dos perfis.

O Quadro 13 descreve o passo a passo para realizar *logoff*.

Quadro 13 – Realizar Logoff

Caso de Uso	Realizar logoff
Descrição	Sair do aplicativo
Ator	Separador e Catador
Pré-condições	O usuário precisa estar cadastrado no sistema.
Pós-condições	O usuário é deslogado do sistema.
Cenário Principal	1. O usuário solicita a saída da aplicação; 2. O sistema desloga o usuário.

O caso de uso Realizar *Logoff* (Quadro 13), citado acima é responsável por avaliar as ações necessárias para o término do uso do sistema.

O Quadro 14 descreve o passo a passo para buscar a localização.

Quadro 14 – Informar Localização

Caso de Uso	Informar localização
Descrição	Informar a localização do material descartado.
Ator	Separador
Pré-condições	O usuário deve estar cadastrado no sistema.
Pós-condições	O sistema deve ser executado.
Cenário Principal	1. O usuário informa o local para a coleta do material.

O caso de uso Informar Localização (Quadro 14), citado acima é responsável por avaliar as ações necessárias para a localização do material informada pelo usuário.

O Quadro 15 descreve o passo a passo para solicitação do recolhimento do material descartado.

Quadro 15 – Solicitar Recolhimento

Caso de Uso	Solicitar Recolhimento
Descrição	O usuário requisita o recolhimento do material.
Ator	Separador
Pré-condições	O usuário precisa estar cadastrado.
Pós-condições	O sistema deve ser iniciado.
Cenário Principal	1. O usuário informa o tipo de rejeito a ser descartado; 2. Em seguida, informa a localização do material; 3. Por fim, solicita a coleta do rejeito.

O caso de uso Solicitar Recolhimento (Quadro 15), citado acima é responsável por avaliar as ações necessárias para realizar a coleta do material descartado.

O Quadro 16 descreve o passo a passo para o salvamento dos dados de autenticação do usuário.

Quadro 16 – Lembrar Login

Caso de Uso	Lembrar login
Descrição	Permite que os dados de autenticação, fiquem salvos na tela inicial da aplicação.
Ator	Catador e Separador
Cenário Principal	1. O usuário preenche os dados de autenticação; 2. Selecionar o recurso “Lembrar Login” ; 3. O sistema armazena os dados no dispositivo.

O caso de uso Lembrar Senha (Quadro 16), citado acima é responsável por avaliar as ações necessárias, para salvar os dados de autenticação do usuário.

O Quadro 17 descreve o passo a passo para a visualização do mapa.

Quadro 17 – Visualizar Mapa

Caso de Uso	Visualizar Mapa
Descrição	Permite que o usuário visualize sua localização ou a rota a ser percorrida, conforme o tipo de perfil do usuário.
Ator	Catador e Separador
Pré-condições	O usuário deve estar cadastrado no sistema.
Cenário Principal	**Usuário Catador** 1. O mapa será visualizado, quando o catador traçar a rota. **Usuário Separador** 1. Sua localização será visualizada no mapa.

O caso de uso Visualizar Mapa (Quadro 17), citado acima é responsável por avaliar as ações necessárias, para a visualização do mapa.

O Quadro 18 descreve o passo a passo para escolha do tipo de material.

Quadro 18 – Escolher Tipo de Material

Caso de Uso	Escolher Tipo de Material
Descrição	Permite que o usuário selecione o tipo de material a ser descartado.
Ator	Separador
Pré-condições	O usuário deve estar cadastrado no sistema.
Cenário Principal	1. O sistema irá apresentar uma lista de materiais; 2. O usuário informa qual tipo de material será descartado.

O caso de uso Escolher Tipo de Material (Quadro 18), citado acima é responsável por avaliar as ações necessárias, para a escolha do tipo de material descartado.

O Quadro 19 descreve o passo a passo para traçar a rota.

Quadro 19 – Traçar Rota

Caso de Uso	Traçar Rota
Descrição	Permite que o usuário visualize através do mapa, a rota a ser percorrida.
Ator	Catador
Pré-condições	O usuário deve estar cadastrado no sistema
Cenário Principal	1. O usuário informa a origem e o destino; 2. O sistema exibe a rota no mapa com a duração e a distância.

O caso de uso Traçar Rota (Quadro 19), citado acima é responsável por avaliar as ações necessárias, para traçar a rota no mapa e informar sua duração e distância.

6 Resultados e Discussões

Neste capítulo são apresentados os resultados obtidos após o desenvolvimento do protótipo e a análise dessas informações. O desenvolvimento deste projeto cumpriu as etapas planejadas e teve como resultado um protótipo com dois perfis: O perfil catador, o qual é responsável pelo recolhimento do material e o perfil separador, que solicita o serviço da coleta de resíduos sólidos.

Para avaliar a usabilidade da aplicação, foram realizados testes com 4 pessoas, onde estas responderam um questionário com perguntas avaliativas. O roteiro da avaliação pode ser visualizado no ANEXO A. No questionário foram realizadas sete perguntas com notas de um a cinco. Durante a avaliação foi possível observar a utilização do protótipo pelos participantes e os pontos que poderiam ser melhorados, a fim de facilitar o uso da aplicação.

No final do testes foi coletado o questionário de avaliação dos participantes com os itens avaliados, sendo estas informações representadas na Figura 8.

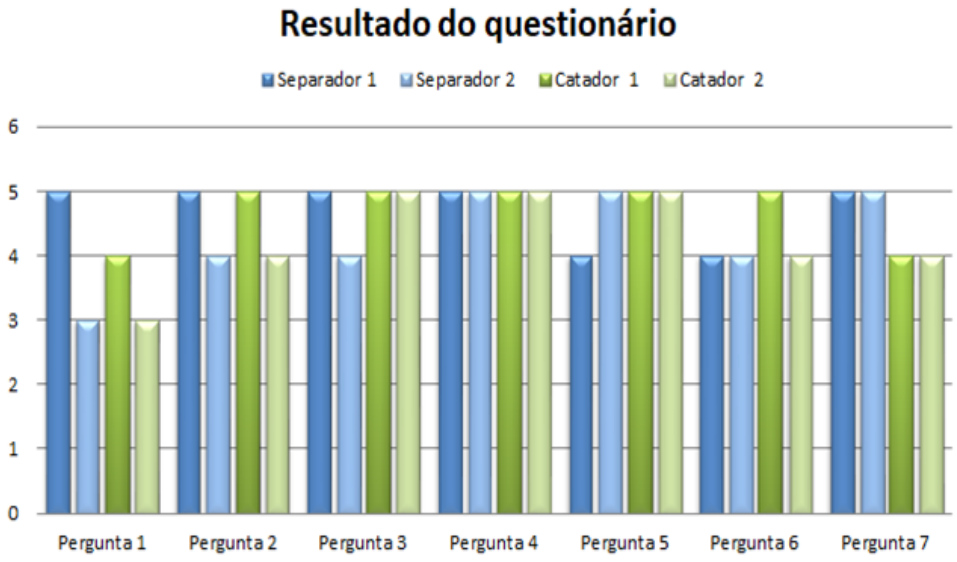


Figura 8 – Gráfico dos resultados obtidos do questionário

Na Figura 8 citada acima foi possível observar que foram criados quatro perfis de usuário: dois perfis separadores e dois perfis catadores. A média dos resultados das perguntas foi satisfatória.

O Codigo 6.1 é responsável por salvar o usuário catador no *firebase realtime database*.

A linha 3 do Codigo 6.1, consiste na criação do nó (*child*) denominado “Catadores”, onde cada catador possui um identificador (*id*), onde seus dados são salvos (*setValue*).

```
1 public void salvar(){
2     DatabaseReference databaseReference = ConfiguracaoFirebase.getFirebase();
3     databaseReference.child("Catadores").child(getId()).setValue(this);
4 }
```

Codigo 6.1 – Salvar usuário catador no banco de dados

Na Figura 9 a seguir, é apresentada a tela inicial da aplicação.

Figura 9 – Tela de Login

Na tela inicial (Figura 9) da aplicação citada acima, o usuário já cadastrado irá informar seu *e-mail* e senha e realizar o *login* no sistema. A aplicação permite a visualização da senha digitada e o armazenamento dos dados de autenticação. Caso o usuário tenha esquecido seu *e-mail* e/ou senha, poderá recuperá-lo em “Esqueceu sua senha?”. Caso não possua cadastro, o usuário irá clicar em “Não possui conta? Cadastre-se”, e será redirecionado para a tela de cadastro. No caso de dúvidas referentes à definição dos perfis, basta o usuário clicar no ícone de interrogação.

Na Figura 10 a seguir, é apresentado o processo de cadastro do usuário catador.

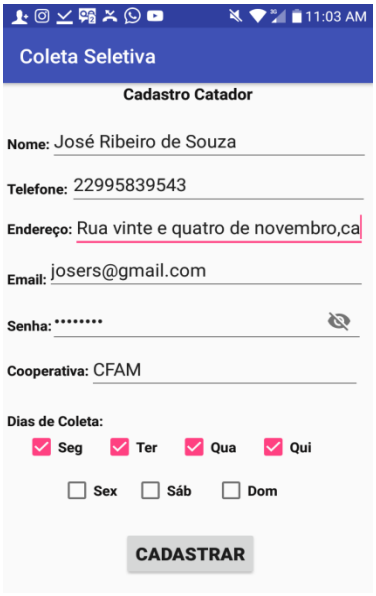


Figura 10 – Cadastro do usuário catador

A Figura 10 citada acima, apresenta a tela de cadastro do usuário catador, onde o mesmo irá informar o nome, o telefone, o endereço, o *e-mail*, a senha, a cooperativa associada e os dias que realiza a coleta. Após o preenchimento de todos esses dados, o usuário irá clicar no botão "cadastrar"e será exibido na tela a confirmação do cadastro. Se algum dado não for preenchido ou *e-mail* já tenha sido cadastrado, a aplicação irá informá-lo.

Na Figura 11 abaixo, é apresentado o processo de cadastro do usuário separador.

Figura 11 – Cadastro do usuário separador

A Figura 11 citada acima, apresenta a tela de cadastro do usuário separador, onde o mesmo irá informar o nome, o telefone, o *e-mail*, a senha e o endereço. Após o preenchimento de todos esses dados, o usuário irá clicar no botão "cadastrar"e será exibido na tela a confirmação do cadastro. Se algum dado não for preenchido ou *e-mail* já tenha sido cadastrado, a aplicação irá informá-lo.

Na Figura 12 a seguir, é apresentada a tela da lista de materiais.

Figura 12 – Lista de materiais

A Figura 12 citada acima, apresenta para o usuário separador, os tipos de materiais que podem ser reciclados. Os tipos de materiais são: plástico, papel, alumínio, metal, eletrônico e vidro. Caso não tenha o tipo de material desejado na lista, o usuário poderá escolher a opção "outros". Após informar o tipo do material será possível enviar a solicitação do recolhimento do resíduo sólido.

Na Figura 13 a seguir, é apresentada a tela referente à localização do material a ser descartado pelo usuário separador.

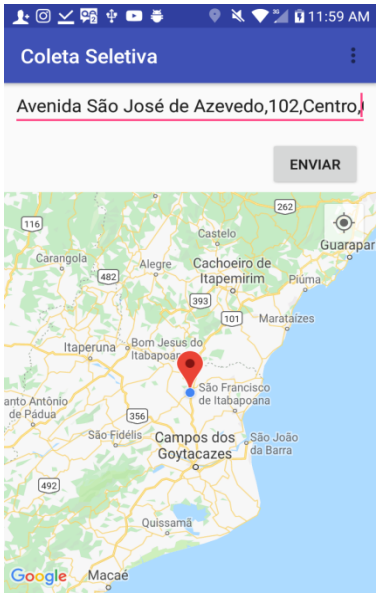


Figura 13 – Informar localização do material

A Figura 13 citada acima, apresenta a tela onde o usuário separador irá informar a localização do material a ser descartado. O ponto de localização presente no mapa é referente à localização do usuário. O mapa contém opções de botões para afastar e aproximar o mapa e um controle de zoom bastante prático, apenas com o deslizar do toque na tela.

Na Figura 14 abaixo, é apresentada a lista de solicitações do recolhimento do material.

Figura 14 – Solicitações do recolhimento do material

A Figura 14 citada cima, apresentada a lista com a requisições de coleta de material que serão visualizadas pelo usuário catador. Os dados apresentados nesta lista são: o

nome do separador, o tipo de material e o endereço da localização deste material. Ao clicar no botão "traçar rota"o catador, será redirecionado para a tela correspondente.

Na Figura 15 a seguir, é apresentada a tela traçar rota.

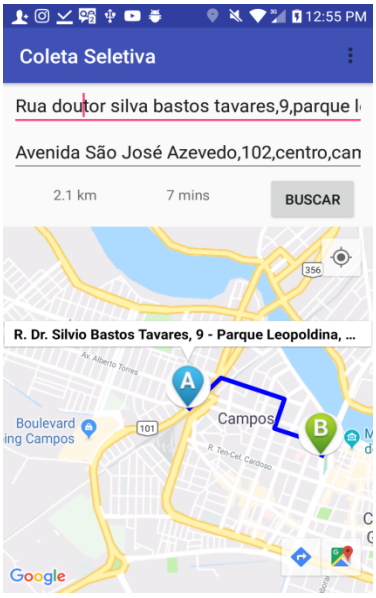


Figura 15 – Tela traçar rota

A Figura 15, apresentada acima, observa-se o funcionamento da rota para o recolhimento do material. O usuário de perfil catador insere a sua localização atual e o local onde o material foi depositado e ao realizar a busca, lhe é fornecido a duração e a distância da rota.

Na Figura 16 a seguir, é apresentada a tela do cadastro de usuários.

Figura 16 – Cadastro de usuários

O protótipo de aplicação faz uso de dois perfis: catador e separador. No ato do cadastro (Figura 16), o usuário irá escolher qual perfil se encaixa e será redirecionado para a tela correspondente.

Na Figura 17 a seguir, é apresentada a tela de recuperação da senha.

Figura 17 – Recuperação da senha

Para recuperar a senha (Figura 17) o usuário irá informar seu *e-mail*, caso não tenha cadastro, o sistema irá informá-lo. Caso contrário, receberá um *e-mail* contendo o *link* e as instruções para a redefinição da senha.

Na Figura 18 a seguir, é apresentada a tela de tira-dúvidas.

Figura 18 – Tira-Dúvidas

A Figura 18 citada acima, é responsável por informar a definição dos perfis: catador e separador.

O recurso de traçar a rota contido na aplicação, permite a previsão da coleta do material, pelo fato de informar a duração e a distância até o ponto do recolhimento.

A informação do tipo de material a ser descartado pelo separador, é benéfica para o catador, pois o mesmo pode otimizar o seu tempo indo aos locais onde existem os materiais de seu interesse.

Neste Capítulo foram apresentados alguns aspectos referentes ao protótipo de aplicação implementado e aos testes realizados. Através do questionário de avaliação, foi possível confirmar que o protótipo de aplicação foi útil e gerou interesse aos participantes do questionário.

7 Conclusão

Com o passar dos anos, o acúmulo de detritos de lixo causaram transtornos ao meio ambiente. A reciclagem de materiais surge como solução deste problema, e se torna essencial para subsistência da nossa sociedade.

A proposta deste trabalho, foi o desenvolvimento de uma ferramenta que possibilite a integração entre pessoas que realizam a separação dos materiais recicláveis, e cooperativas ou pessoas que trabalham com a coleta de forma independente utilizando o processamento e a facilidade do celular para a otimização dos serviços de coleta seletiva. A experimentação realista simulando o uso do sistema, foi bem aceita pelos usuários. Foi possível relacionar as visões dos separadores com os catadores.

Além disso, com a informação do tipo de material depositado pelo separador, o catador otimiza seu tempo, podendo ter um maior número de materiais recolhidos, e com a facilidade de escolher onde descartar o material separado, mais pessoas podem realizar a separação de materiais em suas rotinas.

Com o protótipo de aplicação aplicativo torna possível minimizar os impactos ambientais, que o excesso de acúmulo de detritos de lixo trazem para a sociedade e ao meio ambiente.

Entende-se que como um primeiro passo para um produto maior, as várias funcionalidades idealizadas para este projeto foram implementadas, experimentadas de forma realística e bem aceitas, confirmando o sucesso deste projeto.

7.1 Trabalhos Futuros

Durante o desenvolvimento deste trabalho foram observadas algumas melhorias a serem implementadas, sendo assim vistas como trabalhos futuros. Dentre elas podemos citar:

- Acrescentar todos os pontos de descarte no mapa;
- Otimizar a rota através de algoritmos;
- Aplicar filtros ao mapa, para que o catador possa visualizar somente os pontos pelo tipo de material selecionado;

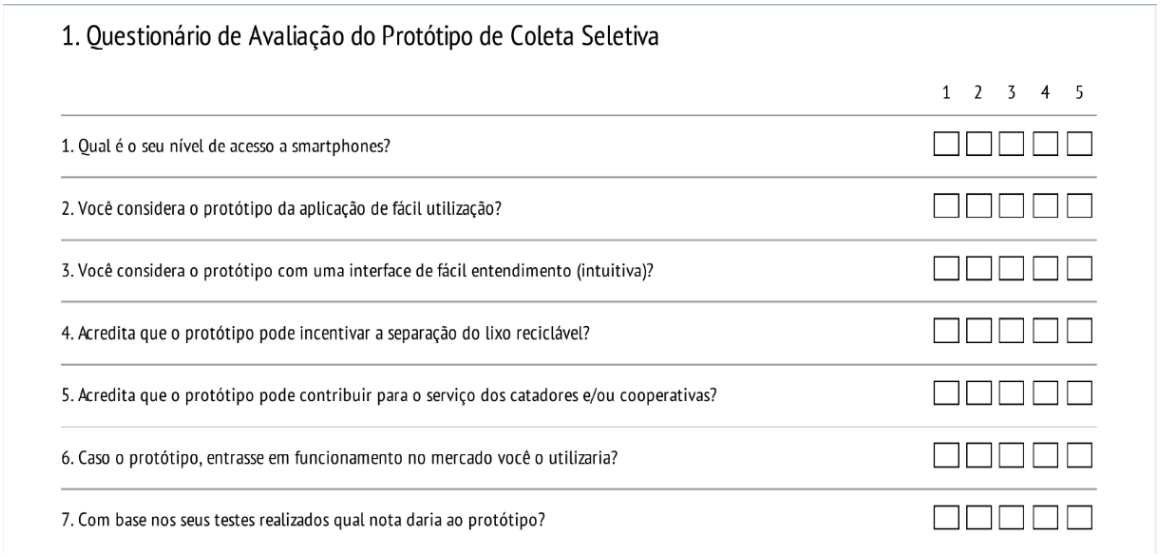
- Enviar notificações quando forem inseridos novos pontos de descarte e quando for sinalizado o recolhimento de um material;

- Adicionar outros métodos de *login*, como por exemplo: o *Facebook,* o *Gmail* e o número de telefone;

- Inserir uma rota dinâmica no programa, em função da localização instantânea do usuário.

Apêndices

APÊNDICE A – Questionário de Avaliação do Protótipo



1. Questionário de Avaliação do Protótipo de Coleta Seletiva

	1	2	3	4	5
1. Qual é o seu nível de acesso a smartphones?	☐	☐	☐	☐	☐
2. Você considera o protótipo da aplicação de fácil utilização?	☐	☐	☐	☐	☐
3. Você considera o protótipo com uma interface de fácil entendimento (intuitiva)?	☐	☐	☐	☐	☐
4. Acredita que o protótipo pode incentivar a separação do lixo reciclável?	☐	☐	☐	☐	☐
5. Acredita que o protótipo pode contribuir para o serviço dos catadores e/ou cooperativas?	☐	☐	☐	☐	☐
6. Caso o protótipo, entrasse em funcionamento no mercado você o utilizaria?	☐	☐	☐	☐	☐
7. Com base nos seus testes realizados qual nota daria ao protótipo?	☐	☐	☐	☐	☐

APÊNDICE B – Códigos fonte comentados

B.1 Salvar usuário catador no banco de dados

O Codigo B.1 é responsável por salvar o usuário catador no *firebase realtime database*. A linha 3 do Codigo B.1, consiste na criação do nó (*child*) denominado "Catadores", onde cada catador possui um identificador (*id*), onde seus dados são salvos (*setValue*).

```
1 public void salvar(){
2     DatabaseReference databaseReference = ConfiguracaoFirebase.getFirebase();
3     databaseReference.child("Catadores").child(getId()).setValue(this);
4 }
```

Codigo B.1 – Salvar usuário catador no banco de dados

B.2 Salvar usuário separador no banco de dados

```
1 public void salvar(){
2     DatabaseReference databaseReference = ConfiguracaoFirebase.getFirebase();
3     databaseReference.child("Separadores").child(getId()).setValue(this);
4 }
```

Codigo B.2 – Salvar usuário separador no banco de dados

O Codigo B.2 é responsável por salvar o usuário separador no *firebase realtime database*. A linha 3 do Codigo B.1, consiste na criação do nó (*child*) denominado "Separadores", onde cada separador possui um identificador (*id*), onde seus dados são salvos (*setValue*).

B.3 Cadastrar usuário catador

```
1  private void cadastrarCatador(){
2      firebaseAuth = ConfiguracaoFirebase.getFirebaseAuth();
3      firebaseAuth.createUserWithEmailAndPassword(
4              catador.getEmail(),
5              catador.getSenha()
6      ).addOnCompleteListener(CadastroCatador.this, new OnCompleteListener<
           AuthResult>() {
7          @Override
8          public void onComplete(@NonNull Task<AuthResult> task) {
9              if(task.isSuccessful()){
10                 Toast.makeText(CadastroCatador.this,"Sucesso ao cadastrar
                       catador",Toast.LENGTH_LONG).show();
11                 FirebaseUser firebaseUser = task.getResult().getUser();
12                 catador.setId(firebaseUser.getUid());
13                 catador.salvar();
14                 startActivity(new Intent(CadastroCatador.this, Login.class));
```

Codigo B.3 – Cadastrar usuário catador na aplicação

O Codigo B.3 consiste no cadastro do usuário catador. A função *createUserWithEmailAndPassword*, que se encontra na linha 3, é responsável pela criação da autenticação dos usuário através do *e-mail* e a senha. Em caso de sucesso, o sistema irá exibir a seguinte mensagem "Sucesso ao cadastrar catador" e o usuário será redirecionado para a tela de *login*.

B.4 Cadastrar usuário separador

```
1  private void cadastrarSeparador(){
2      firebaseAuth = ConfiguracaoFirebase.getFirebaseAuth();
3      firebaseAuth.createUserWithEmailAndPassword(
4              separador.getEmail(),
5              separador.getSenha()
6      ).addOnCompleteListener(CadastroSeparador.this, new OnCompleteListener<
           AuthResult>() {
7          @Override
8          public void onComplete(@NonNull Task<AuthResult> task) {
9              if(task.isSuccessful()){
10                 Toast.makeText(CadastroSeparador.this,"Sucesso ao cadastrar
                       separador",Toast.LENGTH_LONG).show();
11                 FirebaseUser firebaseUser = task.getResult().getUser();
12                 separador.setId(firebaseUser.getUid());
13                 separador.salvar();
14                 startActivity(new Intent(CadastroSeparador.this, Login.class));
```

Codigo B.4 – Cadastrar usuário separador

O Codigo B.4 consiste no cadastro do usuário separador. A função *createUserWithEmailAndPassword*, que se encontra na linha 3, é responsável pela criação da autenticação do usuário através do *e-mail* e a senha. Em caso de sucesso, o sistema irá exibir a seguinte

mensagem “Sucesso ao cadastrar separador” e o usuário será redirecionado para a tela de *login*.

B.5 Redefinir a senha

```
1         private void redefinirsenha(){
2         firebaseAuth = FirebaseAuth.getInstance();
3         firebaseAuth.sendPasswordResetEmail(recuperar.getEmail())
4                 .addOnCompleteListener(new OnCompleteListener<Void>() {
5                     @Override
6                     public void onComplete(@NonNull Task<Void> task) {
7                         if(task.isSuccessful()) {
8                             Toast.makeText(RecuperarSenha.this, "E-mail enviado.
                                  ", Toast.LENGTH_LONG).show();
9                             startActivity(new Intent(RecuperarSenha.this, Login.
                                  class));
10                        }else{
11                            Toast.makeText(RecuperarSenha.this,"E-mail nao
                                  cadastrado.",Toast.LENGTH_LONG).show();
12                        }
13                    }
14                });
15    }
16    \caption{Redefinir a senha}
17    \label{cod:senha}
```

O **??** consiste na redefinição da senha do usuário. A linha 3 verifica se o *e-mail* informado pelo usuário está cadastrado no *firebase auth*. Em caso de sucesso, o sistema irá enviar um *e-mail* para a redefinição da senha e o usuário será redirecionado para a tela de *login*. Caso contrário o sistema irá exibir a seguinte mensagem “*E-mail* não cadastrado”.

B.6 Deslogar usuário

```
1 private void deslogarUsuario() {
2         firebaseAuth.signOut();
3         Toast.makeText(MapsActivity.this, "Usuario deslogado!", Toast.
              LENGTH_LONG).show();
4         Intent intent = new Intent(MapsActivity.this, Login.class);
5         startActivity(intent);
6         finish();
```

Codigo B.5 – Deslogar usuário

O Codigo B.5 consiste no *logout* do usuário. A função *signOut* que encontra na linha 2 é responsável por desconectar o usuário da aplicação. A linha 3 tem a função de exibir para o usuário a seguinte mensagem "Usuario deslogado". A linha 4 tem o papel de redirecionar o usuário para a tela de *login*.

B.7 Validar separador

```
1  private void validarSeparador() {
2      firebaseAuth2 = ConfiguracaoFirebase.getFirebaseAuth();
3      firebaseAuth2.signInWithEmailAndPassword(
4          separador.getEmail(),
5          separador.getSenha()
6      ).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
7          @Override
8          public void onComplete(@NonNull Task<AuthResult> task) {
9              if (task.isSuccessful()) {
10                 FirebaseUser currentUser = FirebaseAuth.getInstance().
                       getCurrentUser();
11                 assert currentUser != null;
12                 String id = currentUser.getUid();
13                 DatabaseReference j = FirebaseDatabase.getInstance().
                       getReference().child("Separadores").child(id);
14                 j.addValueEventListener(new ValueEventListener() {
15                     @Override
16                     public void onDataChange(@NonNull DataSnapshot dataSnapshot)
17                     {
18                         String userType = (String) dataSnapshot.child("userType"
                               ).getValue();
19                         if (userType != null && userType.equals("Separadores"))
                               {
20                             Toast.makeText(Login.this, "Login efetuado com
                                   sucesso!", Toast.LENGTH_SHORT).show();
21                             Intent intentResident = new Intent(Login.this,
                                   Materiais.class);
22                             startActivity(intentResident);
23                             finish();
24                         }
25                 }
```

Codigo B.6 – Validar separador

O Codigo B.6 consiste na validação do usuário separador. A verificação do tipo de usuário separador é realizada através da referência do nó "Separador", do identificador e do tipo de usuário.

B.8 Validar catador

```
1  private void validarCatador() {
2      firebaseAuth2 = ConfiguracaoFirebase.getFirebaseAuth();
3      firebaseAuth2.signInWithEmailAndPassword(
4          catador.getEmail(),
5          catador.getSenha()
6      ).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
7          @Override
8          public void onComplete(@NonNull Task<AuthResult> task) {
9              if (task.isSuccessful()) {
10                 FirebaseUser currentUser = FirebaseAuth.getInstance().
                       getCurrentUser();
11                 assert currentUser != null;
12                 String id = currentUser.getUid();
13                 DatabaseReference j = FirebaseDatabase.getInstance().
                       getReference().child("Catadores").child(id);
14                 j.addValueEventListener(new ValueEventListener() {
15                     @Override
16                     public void onDataChange(@NonNull DataSnapshot dataSnapshot)
17                     {
18                         String userType = (String) dataSnapshot.child("userType"
                               ).getValue();
19                         if (userType != null && userType.equals("Catador")) {
20                             Toast.makeText(Login.this, "Login efetuado com
                                   sucesso!", Toast.LENGTH_SHORT).show();
21                             Intent intentResident = new Intent(Login.this,
                                   Servicos.class);
22                             startActivity(intentResident);
23                             finish();
24                         }
25                 }
```

Codigo B.7 – Validar catador

O Codigo B.7 consiste na validação do usuário catador. A verificação do tipo de usuário catador é realizada através da referência do nó "Catadores", do identificador e do tipo de usuário.

B.9 Traçar rota

```
1         private void parseJSon(String data) throws JSONException {
2         if (data == null)
3             return;
4         List<Route> routes = new ArrayList<Route>();
5         JSONObject jsonData = new JSONObject(data);
6         JSONArray jsonRoutes = jsonData.getJSONArray("routes");
7         for (int i = 0; i < jsonRoutes.length(); i++) {
8             JSONObject jsonRoute = jsonRoutes.getJSONObject(i);
9             Route route = new Route();
10            JSONObject overview_polylineJson = jsonRoute.getJSONObject("
                  overview_polyline");
11            JSONArray jsonLegs = jsonRoute.getJSONArray("legs");
12            JSONObject jsonLeg = jsonLegs.getJSONObject(0);
13            JSONObject jsonDistance = jsonLeg.getJSONObject("distance");
14            JSONObject jsonDuration = jsonLeg.getJSONObject("duration");
15            JSONObject jsonEndLocation = jsonLeg.getJSONObject("end_location");
16            JSONObject jsonStartLocation = jsonLeg.getJSONObject("start_location
                  ");
17            route.distance = new Distance(jsonDistance.getString("text"),
                  jsonDistance.getInt("value"));
18            route.duration = new Duration(jsonDuration.getString("text"),
                  jsonDuration.getInt("value"));
19            route.endAddress = jsonLeg.getString("end_address");
20            route.startAddress = jsonLeg.getString("start_address");
21            route.startLocation = new LatLng(jsonStartLocation.getDouble("lat"),
                   jsonStartLocation.getDouble("lng"));
22            route.endLocation = new LatLng(jsonEndLocation.getDouble("lat"),
                  jsonEndLocation.getDouble("lng"));
23            route.points = decodePolyLine(overview_polylineJson.getString("
                  points"));
24            routes.add(route);
25        }
26        listener.onDirectionFinderSuccess(routes);
27    }
```

Codigo B.8 – Traçar rota

O Codigo B.8 é responsável por traçar a rota informando sua distância e duração.