FACULDADE DE TECNOLOGIA DR. THOMAZ NOVELINO

JOHNNY DONIZETI VAZ FERREIRA
JONAS ANTONIO LOPES DE PAULA

SISTEMA PARA COLETA DE RESÍDUOS BASEADO EM MICROSERVIÇOS

Volume 1

Franca
2021

JOHNNY DONIZETI VAZ FERREIRA
JONAS ANTONIO LOPES DE PAULA

SISTEMA PARA COLETA DE RESÍDUOS BASEADO EM MICROSERVIÇOS

Volume 1

Trabalho de Graduação apresentado à Faculdade de Tecnologia “Dr Thomaz Novelino” – Fatec Franca, como parte dos requisitos obrigatórios para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Me. Ely Fernando do Prado

Franca
2021

JOHNNY DONIZETI VAZ FERREIRA
JONAS ANTONIO LOPES DE PAULA

SISTEMA PARA COLETA DE RESÍDUOS BASEADO EM MICROSERVIÇOS

Volume 1

Trabalho de Graduação apresentado à Faculdade de Tecnologia “Dr Thomaz Novelino” – Fatec Franca, como parte dos requisitos obrigatórios para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Franca, 17 de Maio de 2021

BANCA EXAMINADORA

_____

Prof. Dr. .....

Universidade .....

_____

Prof. Dr. .....

Universidade .....

_____

Prof. Dr. .....

Universidade .....

Dedico este trabalho à minha esposa Mônica que me ajudou dando incentivo e compreendendo o esforço necessário para conclusão deste trabalho e também às minhas filhas Nívea e Sophia.(Johnny)

AGRADECIMENTOS

Agradecemos a todos os professores envolvidos em nosso processo de formação profissional, e a todos os funcionários envolvidos nessa trajetória. Em especial, agradecemos ao Orientador Prof. Me. Ely Fernando do Prado por todo o incentivo, apoio, paciência, dedicação e auxílio durante todo o processo de desenvolvimento do projeto.

Agradecemos aos professores que compõem a banca examinadora, por aceitarem ser os avaliadores do presente trabalho.

Agradecemos aos nossos familiares e amigos por todo o apoio, paciência, palavras de motivação e incentivo que foram essenciais para que continuássemos focados na execução deste trabalho.

A todos que direta ou indiretamente fizeram parte da nossa formação, o nosso muito obrigado.

"Uma ideia não é nada se não tiver quem a execute."
(autor desconhecido)

RESUMO

Embora no último ano tenha havido uma melhora na política de meio ambiente no que se refere a reciclagem, e o fechamento de vários lixões a céu aberto, ainda é muito pouco em relação ao que precisa ser feito. Com o objetivo de contribuir com o meio ambiente é que este projeto foi concebido, criando um sistema robusto para gerenciar a coleta de resíduos recicláveis. O objetivo deste projeto é desenvolver uma plataforma que possa ser usada por outras empresas, para reunir as informações de solicitação de coletas e os dados dos coletores, mais conhecidos como catadores, bem como informações sobre postos de coleta. Este documento aborda as tecnologias necessárias para a criação de um sistema baseado em microserviços com uma interface para ser integrado em outro sistema de parceiros, usando um sistema de mensageria para tornar robusta e escalável esta solução.

Palavras-chave: Lixo. Resíduo. Microserviço. API. Mensageria.

ABSTRACT

Although in the last year there has been an improvement in the environment policy with regard to recycling and with the closure of several open dumps, there is still very little in relation to what needs to be done. In order to contribute to the environment, this project was conceived, creating a robust system to manage the collection of recyclable waste. The objective of this project is to develop a platform that can be used by other companies, to gather the collection request information and the data of the collectors, better known as collectors, as well as information about collection points. This document will address the technologies needed to create a microservice based system with an interface to be integrated into another partner system, using a messaging system to make this solution robust and scalable.

Keywords: Garbage. Residue. Microservice. API. Messaging.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
JPA	Java Persistence API
JQPL	Java Persistence Query language
JSON	Javascript Object Notation
ORM	Mapeador de Objeto Relacional
QR Code	Quick Response, "resposta rápida" em português
REST	Representational State Transfer
SQL	Structured Query Language
URL	Uniform Resource Locator

SUMÁRIO

1 **INTRODUÇÃO**

Sobre descarte de lixo, é sabido que é um problema em todas as cidades, mas vamos ver alguns dados:

Segundo estimativa da Associação Brasileira de Empresas de Limpeza Pública e Resíduos Especiais (Abrelpe), cada brasileiro produz, em média, 387 kg de lixo por ano (AMARO).

A estimativa é de que apenas 3% dos resíduos secos sejam reciclados (AMARO, 2021).

> Em 2019, 40,1% do lixo produzido no Brasil foi descartado de maneira incorreta. Isso representa 29 milhões de toneladas. Os dados foram divulgados pela Associação Brasileira de Empresas de Limpeza Pública e Resíduos Especiais (Abrelpe) (ALVES, 2021).

> Ao menos 3.000 dos 5.570 municípios do país mantêm lixões a céu aberto, e quase metade deles ainda utiliza os locais para depositar resíduos sólidos, segundo a Abrelpe (Associação Brasileira de Empresas de Limpeza Pública e Resíduos Especiais) (MOTTER, 2021).

Muita coisa precisa ser alterada, desde a cultura dos brasileiros em separar e descartar o lixo de forma correta, a melhorias no setor público e privado.

No campo do conhecimento, muito precisa ser feito, pois de acordo com esses dados é percebida certa ignorância por parte da sociedade.

Mas o objetivo deste projeto é desenvolver uma tecnologia que contribua com o meio ambiente, e que além de trazer essa melhoria ecológica e sustentável, também possa de alguma forma criar novas formas de negociação. Em uma versão futura também pode contribuir com a divulgação do conhecimento sobre o assunto.

Em termos de tecnologia, uma das coisas que fica evidente é que existe uma necessidade de automatizar a comunicação entre as partes envolvidas, desde o gerador do material até o seu reciclador, encurtando caminhos, eliminando custos com intermediários, aproveitando recursos disponíveis.

Essa tecnologia deve ser acessível e robusta, e, neste ponto, este projeto aborda alguns pontos que podem contribuir para uma solução a essa demanda.

Como a problemática citada é em nível nacional, a solução apresentada deve atender o requisito de escalabilidade, nesse ponto, deve possuir uma estrutura que seja "First Cloud", ou seja, estruturada para iniciar na nuvem, dentro dos padrões já conhecidos do mercado. Dentre essa estrutura um dos pontos importantes é a conteinerização, expressão usada para indicar que um aplicativo está localizado dentro de um "contêiner" (a tecnologia mais conhecida para uso de conteiners é o docker) (PERLOW).

Todos os microserviços desenvolvidos neste projeto pode ser conteinerizados, para uso em Cloud.

Uma solução com essa robustez, também necessita se utilizar de alguns padrões de projeto. Por isso, será tratada, no capítulo sobre Conceitos, a tecnologia de uso do padrão OpenAPI para comunicação entre aplicativos, bem como a implementação do Swagger nas APIs do projeto.

Um dos padrões utilizados no desenvolvimento foi o SOLID, cujo primeiro conceito é:

- Princípio da responsabilidade única (THE PRINCIPLES...).

Por isso o sistema foi bem fragmentado em classes pequenas.

A estratégia usada para o desenvolvimento dos microserviços foi usar a linguagem JAVA usando o Framework Spring Boot para facilitar a injeção de dependências e uso de anotações. No caso da comunicação com o banco de dados foi utilizado o Spring JPA, um ORM (Object Relational Mapper), ou seja, um mapeador de objeto relacional para manipular a criação de tabelas, inserção de dados e consultas via código JPQL - (Java Persistence Query language), usando apenas JAVA para manipular o banco de dados.

O aplicativo que fará a interface com o frontend, nesse caso, será o BFF (Backend For Frontend), único sistema feito em javascript com Typescript, isso porque foi usado o Framework AdonisJs.

O AdonisJs possui diversas ferramentas como: Validadores, Autenticação, Routing, Websocket, Seeds, e diversas outras que irão agilizar no desenvolvimento dos recursos futuros, como a criação de perfil de acesso e o sistema de segurança da API.

Para uma compreensão da integração entre os microdserviços, o capítulo Diagramas apresenta as imagens que darão uma visão geral das funcionalidades do sistema.

E no capítulo Detalhamento dos Microsserviços, é apresentado um mapa mental para apresentar os recursos separados em 1.0 e 2.0, sendo inclusos, nesse último, alguns recursos não desenvolvidos, colocados como versão 2.0 deste projeto, ou seja, itens fora do escopo deste trabalho de conclusão.

2 **CONTEXTUALIZAÇÃO**

Diante do problema de descarte de resíduos e reciclagem, foi pensada uma solução completa para o melhor descarte destes materiais. Entretanto, é importante destacar que o objeto deste Trabalho de Conclusão é apenas no sistema "por baixo" do Aplicativo mobile ou site, classificado como backend.

Este backend é a estrutura que faz a gestão das solicitações de coleta e as notificações. No entanto, para que este projeto funcione é necessária a integração com um outro sistema, para ser a interface com o usuário. Por isso o foco deste projeto são os microsserviços necessários para funcionamento do backend.

Neste Model Canvas está sendo apresentado o projeto completo, contemplando tanto o backend como frontend.

Canvas desenvolvido no site: https://www.sebraecanvas.com

Figura 1 — Canvas

Fonte: Os autores (2021)

A ideia é que o usuário do sistema instale, no seu ponto comercial ou indústria, uma lixeira com um QR Code, e, com o sistema configurado, faça a solicitação de coleta, apenas apontando o aplicativo para o QR Code.

Em seguida, o coletor responsável por aquele posto de coleta, receberá uma notificação, e poderá dar, ou não, o aceite da coleta. Caso o coletor aceite, será enviada, também, uma notificação para o responsável do posto de coleta. Caso o coletor negue a coleta, essa notificação será enviada, tanto para o dono do posto de coleta, quanto para o administrador do sistema.

Outros recursos que serão objeto de versão futura:

- Solicitação de coleta sem o QR Code;
- Agendamento de solicitações de coleta;
- Consulta postos de coleta por tipo de material;
- Sistema de avaliação do coletor, resíduos e posto de coleta;
- Impressão de melhor rota para coletas;
- Tela para administrador do sistema;
- Geração de dados para relatórios e dashboard.

A parceria com uma empresa, para integrar neste backend, é um requisito fundamental para funcionamento deste projeto.

O software foi estruturado para poder ser integrado com vários tipos de sistema, ou seja, o parceiro integrador pode ser uma rede de lojas, um aplicativo de vendas, ou até mesmo uma rede social.

O sistema de login não foi desenvolvido porque a ideia é utilizar essa informação do frontend.

3 **OBJETIVO**

Desenvolver o backend para solicitação de coleta de resíduos.

3.1 **OBJETIVOS ESPECÍFICOS**

Desenvolver microsserviços que interajam, entre si, para o fluxo de solicitação de coleta, expondo o serviço através de uma API padronizada;

Apresentado uma forma moderna para aumentar a resiliência da aplicação com uso de mensageria.

3.2 **VIABILIDADE DO PROJETO**

O descarte de resíduos é um problema sério e volumoso em todas as cidades. Por este motivo, este documento apresenta uma forma robusta de contribuir para a solução deste problema, focando em uma tecnologia que possa ser escalável e resiliente.

Esta solução implementa o conceito de camadas independentes, entre as vantagens desta estrutura estão:

I. Escalabilidade: no caso de aumento significativo de usuários, é possível fazer o escalonamento da infraestrutura no ponto crítico, ou seja, na camada que está sofrendo o maior uso. E o uso do recurso físico pode ser feito sob demanda.

II. Resiliência: Controles contra falha como "Circuit Breacker" (Disjuntor), podem ser implementados nas camadas para que se um determinado serviço parar, esta parte é desligada e não acarreta problemas maiores. Contendo as requisições para futuro processamento quando a falha no circuito estiver corrigida.

III. Manutenibilidade: Com a definição objetiva de cada microsserviço, a complexidade é reduzida significativamente, e com isso, a manutenção do sistema se torna muito mais fácil. Com as camadas desacopladas e independentes, um novo recurso pode ser implementado, "como uma peça de lego".

IV. Produtividade: Com essa característica já citada, as equipes podem se tornar especialistas em parte do sistema, e novos recursos, ou ações de manutenção, são implementados de forma muito mais rápida.

V. Flexibilidade: Nesse padrão de arquitetura, a equipe pode escolher a linguagem de programação, banco de dados ou framework de cada

Microsserviço, definindo, assim, a melhor tecnologia que se encaixe em certo cenário específico do Microsserviço.

> Circuit Breaker Pattern - (padrão disjuntor) é a solução para este problema. A ideia básica por trás do disjuntor é muito simples. Você envolve uma chamada de função protegida em um objeto de disjuntor, que monitora as falhas. Uma vez que as falhas atingem um determinado limite, o disjuntor desarma e todas as chamadas adicionais para o disjuntor retornam com um erro ou com algum serviço alternativo ou mensagem padrão, sem que a chamada protegida seja feita. Isso garantirá que o sistema está respondendo e os threads não estão esperando por uma chamada sem resposta (KUMAR, 2021).

3.3 **SISTEMA MONOLÍTICO - PORQUE NÃO USAR**

Ao contrário da proposta desta projeto existem os sistemas monolíticos.

> Monólito significa “obra construída em uma só pedra” por isso é utilizado para definir a arquitetura de alguns sistemas, refere-se a forma de desenvolver um sistema, programa ou aplicação onde todas as funcionalidades e códigos estejam em um único processo. “Uma aplicação monolítica é aquele tipo de aplicação na qual toda a base de código está contida em um só lugar, ou seja, todas as funcionalidades estão definidas no mesmo bloco.” (SANTOS, 2017). (MARQUES FERNANDES).

Algumas desvantagens desta estrutura:

I. Manutenibilidade: Com o aumento das funcionalidades do sistema, o código se torna maior e dificulta a manutenção e as entregas começam a ficar mais complexas e menos frequentes.

II. Escalabilidade: neste modelo é preciso escalar toda a aplicação, dificultando o aumento de alguns recursos do sistema, que começam a ficar limitados. Em alguns casos, não é possível somente adicionar mais máquinas (processador, memória, …) para aplicação, sendo necessários modelos com balanceador de carga.

III. Custos: se um ponto do sistema precisa ser escalado, não tem como fazer isso separadamente: é preciso escalar a aplicação inteira.

4 **CONCEITOS SOBRE A TECNOLOGIA**

4.1 **API - APPLICATION PROGRAMMING INTERFACE**

> Interface de Programação de Aplicações (português europeu) ou Interface de Programação de Aplicação (português brasileiro)), cuja sigla API provém do Inglês Application Programming Interface, é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços (INTERFACE..., 2021).

A comunicação entre aplicativos é feita principalmente por meio das APIs. Esta abordagem simplificou a comunicação entre sistemas.

Antigamente, outra forma de comunicação entre sistemas era o acesso direto ao banco de dados, mas essa abordagem tem muitos riscos, e cada vez mais tende a ser extinta.

A forma de comunicação via API, se usada corretamente, é muito segura. Mas, logicamente, se não forem implementadas as etapas de segurança, como token, usuário e senha, também deixará a aplicação sem segurança.

O bom do uso da API, é que a pessoa que vai desenvolver o sistema que vai consumir o serviço, não precisa conhecer a tecnologia que foi usada, e, desta forma, pode usar a linguagem que desejar

Um exemplo de uso de API é a do Google Maps, pois existem diversos sistemas que usam a funcionalidade, mas cada um disponibiliza as funcionalidades de uma forma diferente.

Vários sistemas fechados estão disponibilizando API de acesso para estender uma funcionalidade desenvolvida por outra equipe.

4.2 **PADRÃO OPEN API**

> A OpenAPI Initiative (OAI) foi criada por um consórcio de especialistas da indústria voltados para o futuro que reconhecem o imenso valor da padronização sobre como as APIs são descritas. Como uma estrutura de governança aberta sob a Linux Foundation, a OAI está focada na criação, evolução e promoção de um formato de descrição neutro do fornecedor. A especificação OpenAPI foi originalmente baseada na especificação Swagger, doada pela SmartBear Software (OPENAPI.ORG).

Atualmente, a necessidade de integrações entre sistemas está ficando cada vez maior. Por isso, o processo de padronização entre aplicações se torna uma questão de sobrevivência.

O uso do padrão OpenAPI faz com que o desenvolvedor não precise conhecer a fundo o sistema que está criando a integração. Somente pelo padrão

OpenAPI é possível fazer a integração entre sistemas independentemente da tecnologia utilizada.

O software usado para implementar este padrão é o *Swagger,* exemplificado no capítulo 5.1.

4.3 **CONTRATO PRIMEIRO - (CONTRACT FIRST)**

A abordagem de Contrato-Primeiro usado na construção da API é a definição da estrutura dos dados. Neste caso, várias equipes de desenvolvimento poderão trabalhar, simultaneamente, em partes diferentes do sistema, uma vez que já terá sido acordado como o serviço irá se comportar: o tamanho dos campos, nomes, e tipos. Geralmente, o padrão de comunicação desse contrato é um arquivo JSON. Em seguida, a equipe pode iniciar a implementação do serviço de forma paralelizada.

4.3.1 **Vantagens do contrato primeiro**

- Os desenvolvedores podem criar os serviços paralelamente, de forma simultânea;
- Todos envolvidos sabem o que esperar, e, com base no contrato, podem desenvolver no ambiente de testes, apenas inserindo os *stubs* conforme o contrato e simular o comportamento.
- Como os parâmetros do serviço dependem somente do contrato, a tecnologia usada não altera o resultado.
- Permite o reuso dos esquemas.

> STUBS - em desenvolvimento de software, é um pedaço de código usado para substituir algumas outras funcionalidades de programação. Um stub pode simular o comportamento de um código existente (como um procedimento em uma máquina remota) ou ser um substituto temporário para o código ainda a ser desenvolvido. Eles são portanto mais úteis em portabilidade, computação distribuída bem como no desenvolvimento e teste de software em geral (STUB..., 2021).

5 **ESTRUTURA DO PROJETO**

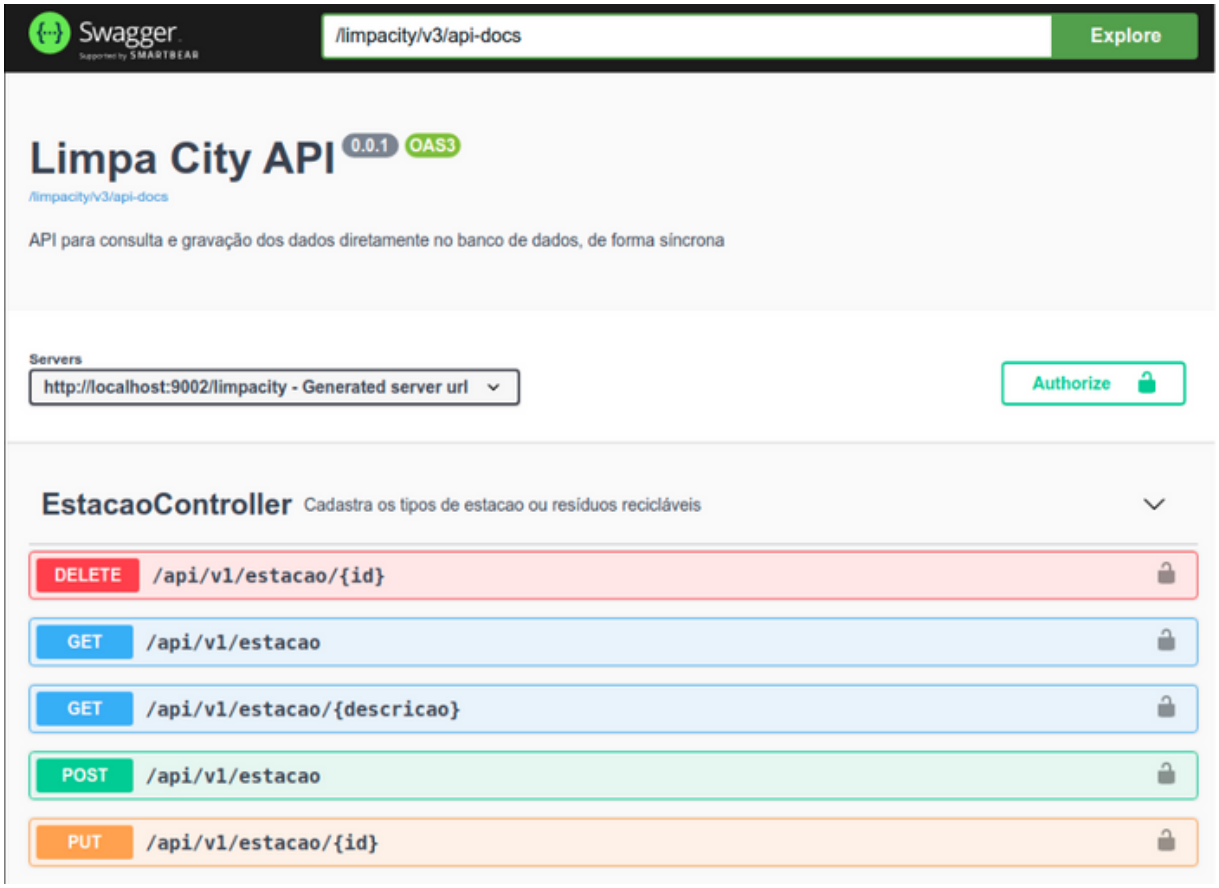
A estrutura, aqui entendida como os padrões utilizados para a elaboração do projeto, serão detalhados abaixo.

5.1 **OPENAPI**

Na figura 2, é mostrado um exemplo da API que faz a gravação no banco de dados de forma síncrona.

Com o uso desta padronização, qualquer desenvolvedor, com experiência em API, pode fazer a comunicação, independentemente da tecnologia escolhida. Isto é um bom exemplo da característica desta infraestrutura de microsserviço com camadas desacopladas e independentes.

Figura 2 — OpenAPI no sistema



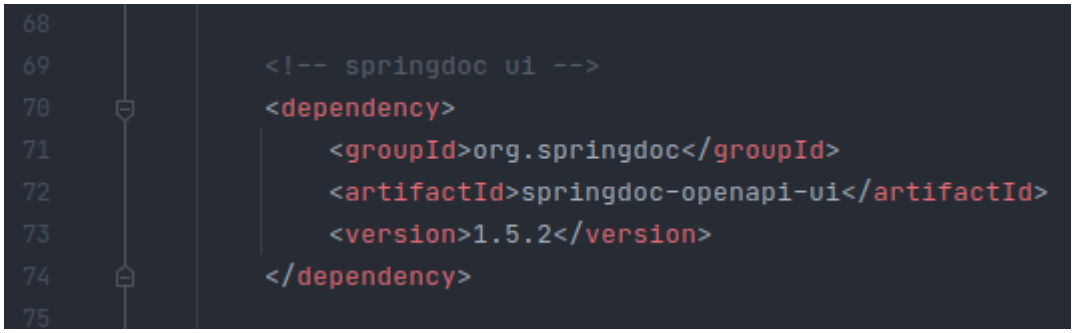
Fonte: O autor (2021)

5.1.1 **Exemplo de Implementação do Swagger**

Segue um exemplo de como foi implementado no Microsserviço, denominado Limpacity_api, o padrão citado acima: OpenAPI através do Swagger

Como o Sistema foi desenvolvido em Java/Spring, primeiramente deve ser feita a instalação do pacote necessário no arquivo pom.xml

Figura 3 — Configuração da dependência no pom.xml



Fonte: O autor (2021)

Em seguida criado o arquivo de configuração no caminho:
src/main/java/br/com/limpacity/api/config/OpenAPIConfig.java

Figura 4 — Configuração do Swagger

```
@Profile("!test")
@Configuration
public class OpenAPIConfig {

    private static final String API_KEY = "bearer-key";

    @Bean
    public io.swagger.v3.oas.models.OpenAPI customOpenAPI(@Value("${springdoc.version}") String appVersion) {
        return new io.swagger.v3.oas.models.OpenAPI()
                .components(new Components()
                        .addSecuritySchemes(API_KEY, apiKeySecuritySchema()))
                .info(new Info()
                        .title("Limpa City API")
                        .description("API para consulta e gravação dos dados diretamente no banco " +
                                "de dados, de forma síncrona ")
                        .version(appVersion))
                .security(Collections.singletonList(new SecurityRequirement().addList(API_KEY)));
    }

    public SecurityScheme apiKeySecuritySchema() {
        return
                new SecurityScheme()
                        .name("authorisation-token")
                        .description("Insira o Token sem a palavra Bearer")
                        .scheme("bearer")
                        .in(SecurityScheme.In.HEADER)
                        .bearerFormat("JWT")
                        .type(SecurityScheme.Type.HTTP);
    }
}
```

Fonte: O autor (2021)

Desta forma, através das anotações do Swagger, é possível, dentro do código, realizar a documentação da API

Exemplo de Anotação:

@Operation(description = "Insere uma nova solicitação de coleta vindo da leitura do QRCode")

Figura 5 — Documentando API no próprio código

```
    @PostMapping("/qrcode/{posto}")
    @Operation(description = "Insere uma nova solicitação de coleta vindo da leitura do QRCode ")
    public ResponseEntity<ResponseBodyDTO<ColetaQrcodeDTO>> postColetaQrcode(
            @Valid @RequestBody ColetaQrcodeDTO coleta,
            @PathVariable("posto") Long posto_id){
        logger.info("Solicitação Qrcode : Posto {} " + posto_id );
        return buildResponseBody(service.createQrcode(posto_id, coleta), HttpStatus.CREATED);
    }
```

Fonte: O autor (2021)

Nesta simples anotação dentro do código, será impressa, na tela de documentação da API, a informação para entendimento deste "endpoint"

Muitas pessoas podem confundir uma API com um outro termo muito falado ultimamente, que são os endpoints. Um endpoint é basicamente o que um serviço expõe e esse serviço pode ser acessado por uma aplicação, por isso muitas vezes acaba sendo confundido com uma API, mas vale ressaltar que não é.
Um endpoint contém três principais características: Address (onde o serviço está hospedado), Binding (como o serviço pode ser acessado) e Contract (o que tem no serviço). Além disso, uma API pode existir sem um endpoint e vice-versa (O QUE É UMA API...).

Figura 6 — informação do endpoint

Fonte: O autor (2021)

5.1.2 **Conclusão do Uso do OpenAPI/Swagger**

Da forma como foi desenvolvida esta API, para que haja a integração com outras partes do fluxo do sistema, não é necessário o desenvolvedor conhecer nada sobre a estrutura ou linguagem interna da API. Ou seja, caso o desenvolvedor for usar uma outra linguagem de programação, como nodejs, ele não precisa ter domínio sobre JAVA e Spring Boot; basta conhecer o padrão que é comum neste tipo de aplicação.

5.2 **ASSÍNCRONO VERSUS SÍNCRONO**

Para implementação desta solução, foi pensada uma abordagem na qual uma parte do fluxo seria processada e gravada no banco de dados de forma síncrona, e a outra parte, de forma assíncrona.

Processo Síncrono: Por exemplo, no caso de uma restrição no cadastro de usuário, de não ser permitida a criação de e-mail duplicado: neste caso, a API faz a conexão com o banco, e só depois da validação dos dados é que o usuário recebe o retorno, seja positivo ou não.

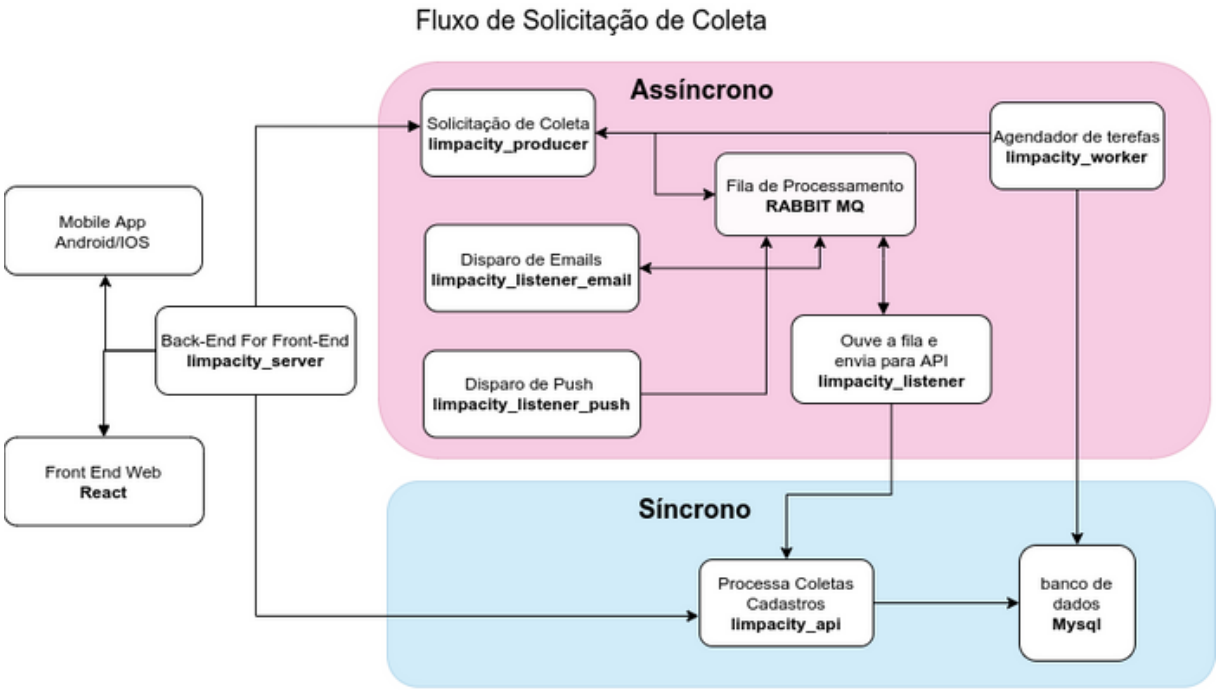
Processo Assíncrono: No caso do fluxo relacionado a solicitações de coleta por exemplo, o usuário não precisa receber a informação de que a coleta será feita e/ou quem irá coletar, de forma imediata. Isso porque, neste fluxo, depende do coletor interagir no sistema. Neste contexto, haverá um processamento inteligente para decidir para qual coletor será enviada a mensagem, e, talvez, até aguardar por alguma interação até que haja uma resposta. Nesse caso, o usuário envia, para uma fila, a sua solicitação, e enquanto isso, o sistema continua processando, buscando informações para enviar ao usuário (posto de coleta), de forma assíncrona.

5.3 **ESTRUTURA DO MICROSSERVIÇOS**

Na figura 7 está representada a relação entre os processos que são assíncronos e síncronos.

No caso, todo fluxo ligado à Fila de Mensagens (RABBIT MQ), é realizado de forma assíncrona, e o fluxo ligado à API de cadastro é feita de forma síncrona.

Figura 7 — Infraestrutura de microserviço

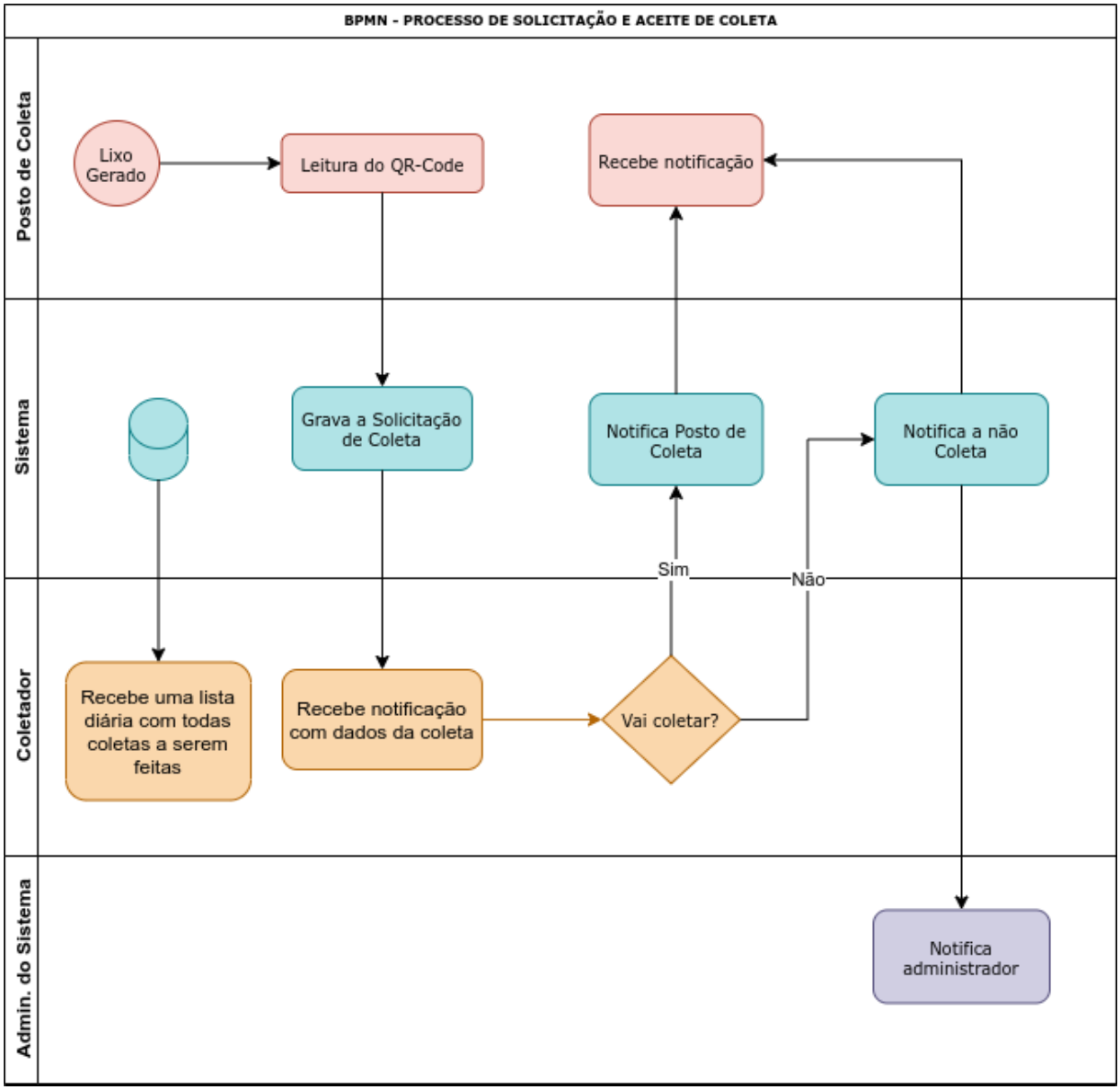


Fonte: O autor (2021)

6 **DIAGRAMAS**

6.0.1 **BPMN**

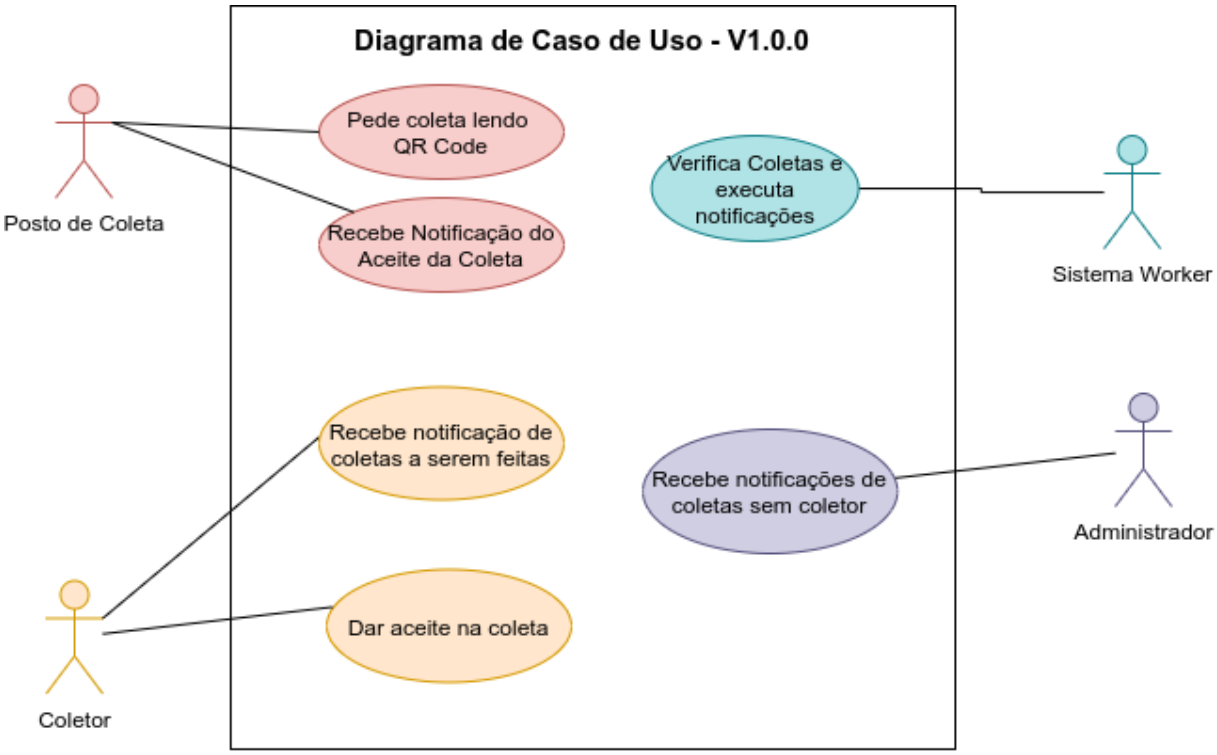
Figura 8 — Diagrama BPMN



Fonte: O autor (2021)

6.0.2 **Caso de Uso - Versão 1.0**

Figura 9 — Caso de Uso - v1.0.0

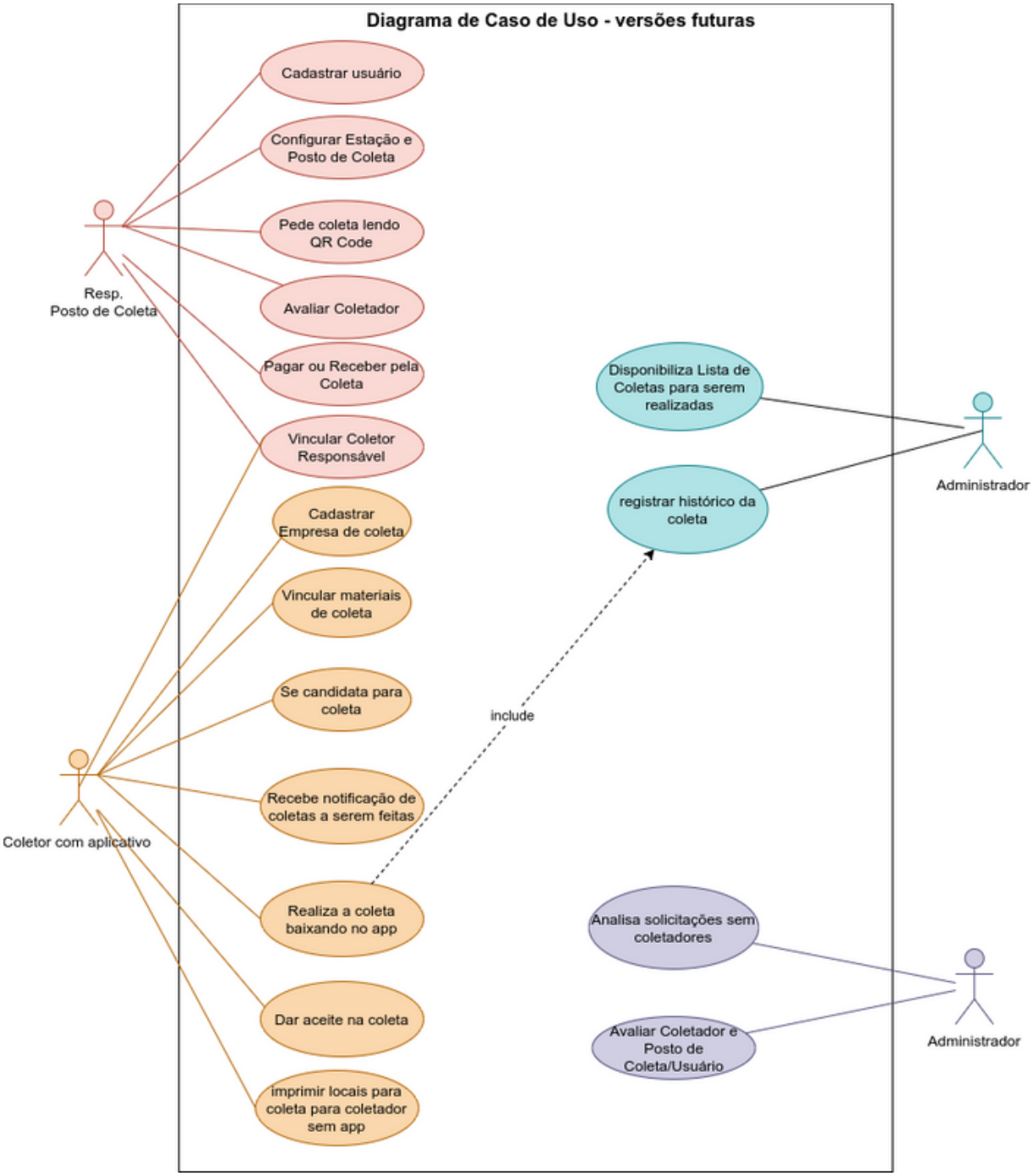


Fonte: O autor (2021)

6.0.3 **Caso de Uso - Versão 2.0**

Este diagrama possui ideias que poderão ser implementadas em versões futuras.

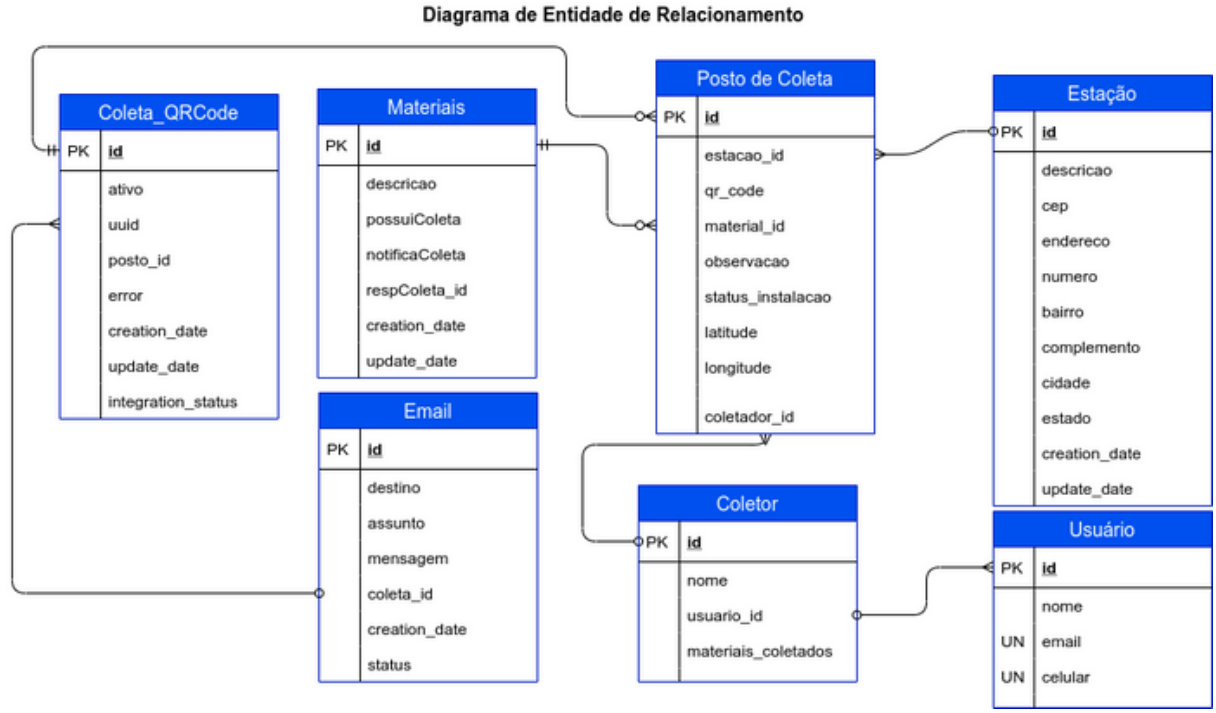
Figura 10 — Caso de Uso - Itens fora de Escopo



Fonte: O autor (2021)

6.0.4 **Diagrama de Entidade de Relacionamento - DER**

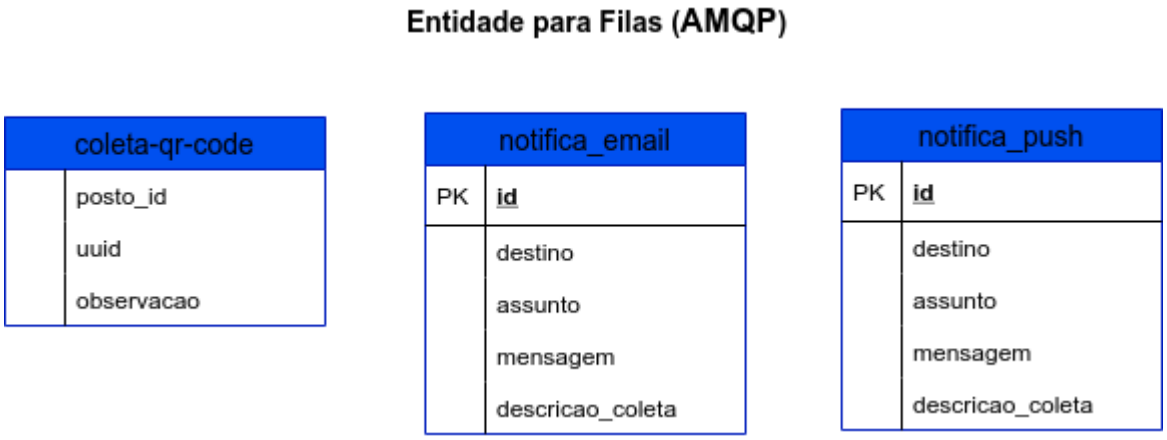
Figura 11 — Diagrama de Entidade de Relacionamento



Fonte: O autor (2021)

6.0.5 **Diagrama de Entidade da Mensageria**

Figura 12 — Entidade da Mensageria



Fonte: O autor (2021)

7 DETALHAMENTO DOS MICROSSERVIÇOS

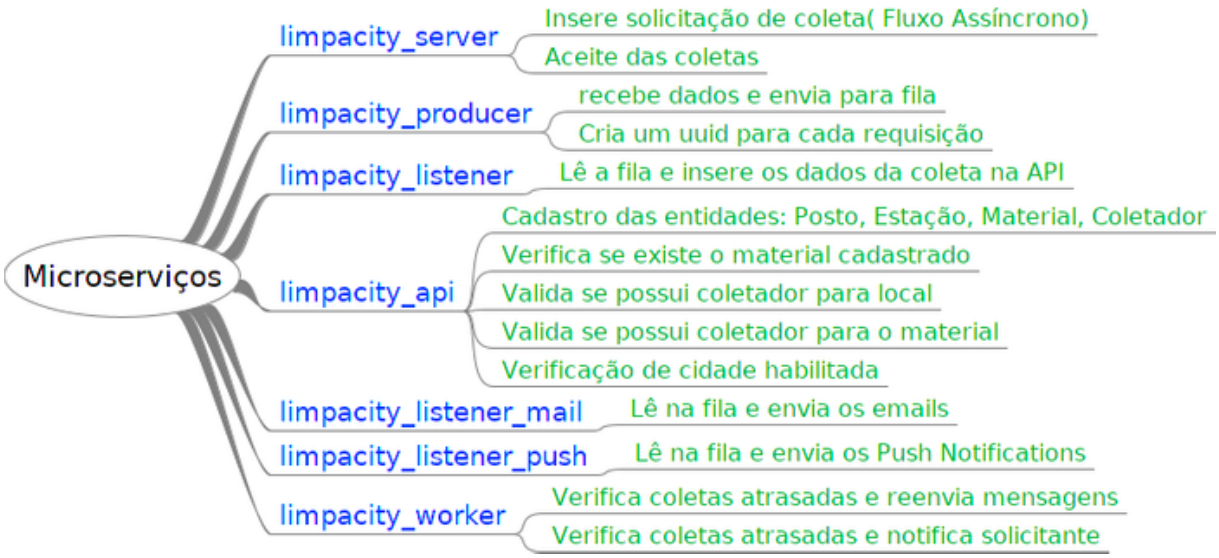
7.1 VERSIONAMENTO

A separação dos requisitos será feita por versão, de forma que a versão 2.0.0 será composta pelos requisitos que estão fora do escopo, ou seja, somente a versão 1.0.0 se refere ao que foi desenvolvido e se encontra disponível para ser apresentado. Na modelagem dos requisitos, será utilizado o padrão de versionamento SemVer (SERVER.ORG).

7.1.1 Versão 1.0.0

Trabalho concluído e pronto para ser entregue.

Figura 13 — Mapa mental dos Microserviços

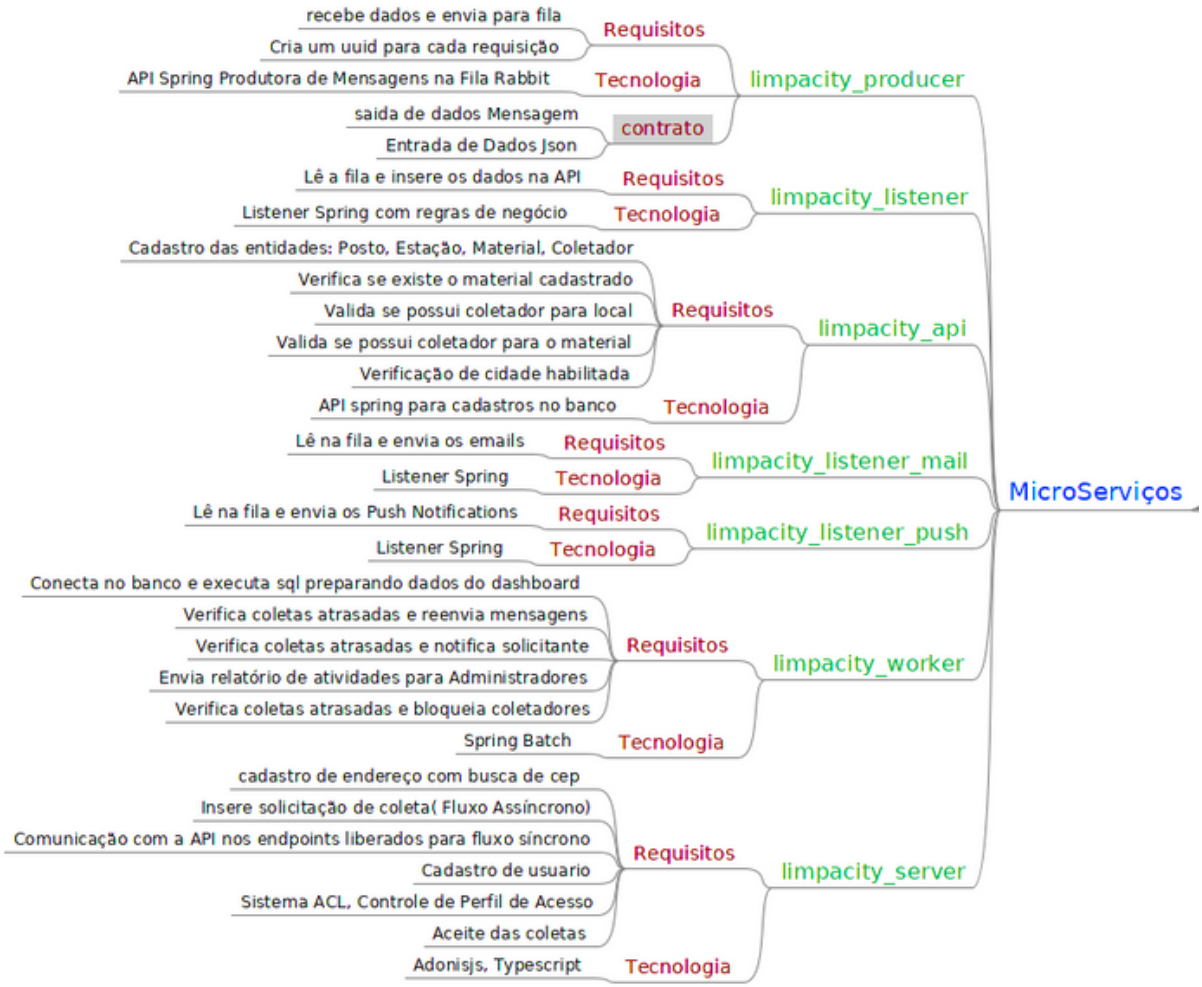


Fonte: O autor (2021)

7.1.2 Versão 2.0.0

Com alguns itens não desenvolvidos, considerados fora de escopo.

Figura 14 — Mapa mental versão 2.0.0



Fonte: O autor (2021)

A figura 13 mostra como estão organizadas as funções de cada sistema, e que estão prontas na versão 1.0.0 do sistema. Já a figura 14 é o mapa mental completo com os itens, tanto da versão 1.0, quanto da 2.0.

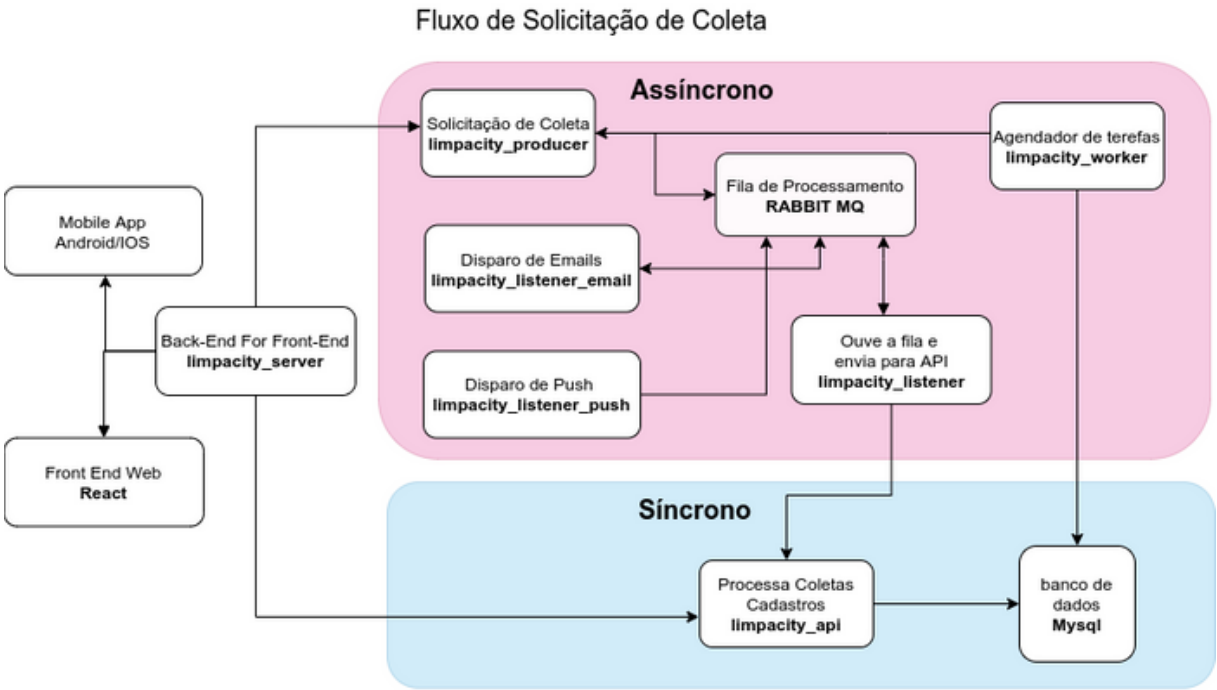
8 **REQUISITOS DOS MICROSERVIÇOS**

Estão relacionados, nas tabelas abaixo, os requisitos para cada microsserviço. É importante reforçar a informação de que na coluna Versão, quando for 1.0, é a funcionalidade já implementada e pronta para apresentação deste projeto, quando for 2.0, é para futuro desenvolvimento, e por isso, não será apresentada a funcionalidade.

Na imagem abaixo está uma visão geral do fluxo dos microsserviços,

Obs.: Os apps que estão antes do limpacity_server são somente ilustrativos (Mobile App e Front End Web).

Figura 15 — Fluxo dos Microserviços



Fonte: O autor (2021)

8.1 **LIMPACITY_SERVER**

Este sistema prepara as informações para o front-end, reunindo os dados do banco e das APIs.

O Server é a interface responsável pela comunicação entre os integradores externos e entre os microsserviços do limpacity.

Tabela 1 — Requisitos funcionais Server

Código	Tarefa	Descrição	Versão
RF01	Inserir solicitação de coleta baseado em QR Code	A API deve receber os dados vindo da leitura de um QR Code, com as informações de posto de coleta e um texto para descrever observações.	1.0
RF02	Aceitar o aceite das coletas	Baseado no código único da coleta, deve receber a mensagem de 'sim' ou 'não', e gravar no banco	1.0
RF03	Aceitar solicitação de coleta sem QRCode	Como não terá o QRCode o sistema deverá aceitar o envio de dados como, endereço, nome do solicitante e tipo de material no endpoint da solicitação de coleta.	2.0
RF04	Cadastro de endereço com busca de CEP	Deverá receber os dados de endereço e consultar se o local está habilitado para coleta	2.0
RF05	Cadastro de perfil de usuário	Deve permitir o cadastro de perfil de usuário pelo administrador do sistema	2.0
RF06	Cadastro de usuário	Deve permitir o cadastro de usuário com informações de contato como telefone e e-mail	2.0
RF07	Habilitação de Cidade	Deverá consultar no cadastro de município do sistema e permitir alterar status do município de Habilitado e não Habilitado	2.0
RF08	Permitir cadastro de Estação de Coleta	Deverá permitir que o cadastro da estação de coleta, com endereço e usuário vinculado, o endereço deve estar em um município habilitado.	2.0
RF09	Permitir cadastro de Posto de Coleta	Somente com uma Estação de coleta vinculado é que deve ser permitido o cadastro de posto de coleta, deve ser obrigatório o material.	2.0
RF10	Emissão de QRCode	Deverá ser possível enviar um código de posto de coleta e receber a imagem do QRCode para ser colocado na lixeira	2.0

Fonte: O autor (2021)

Tabela 2 — Requisitos Não Funcionais server (continua)

Código	Tarefa	Descrição	versão
RNF01	Sistema de segurança para o login	Deve ter um sistema de login baseado em token com expiração baseado em tempo de vida.	2.0
RNF02	Acesso somente com protocolo HTTPS	instalar sistema de certificado SSL para tráfego de informações criptografadas	2.0
RNF03	Tecnologia Javascript	Utilizar Javascript, Typescript com o Framework Adonisjs	1.0
RNF04	Preparar cada endpoint para controle de Perfil de	Criar um sistema de controle em cada rota, para ser definido o acesso de acordo com o perfil de acesso	2.0

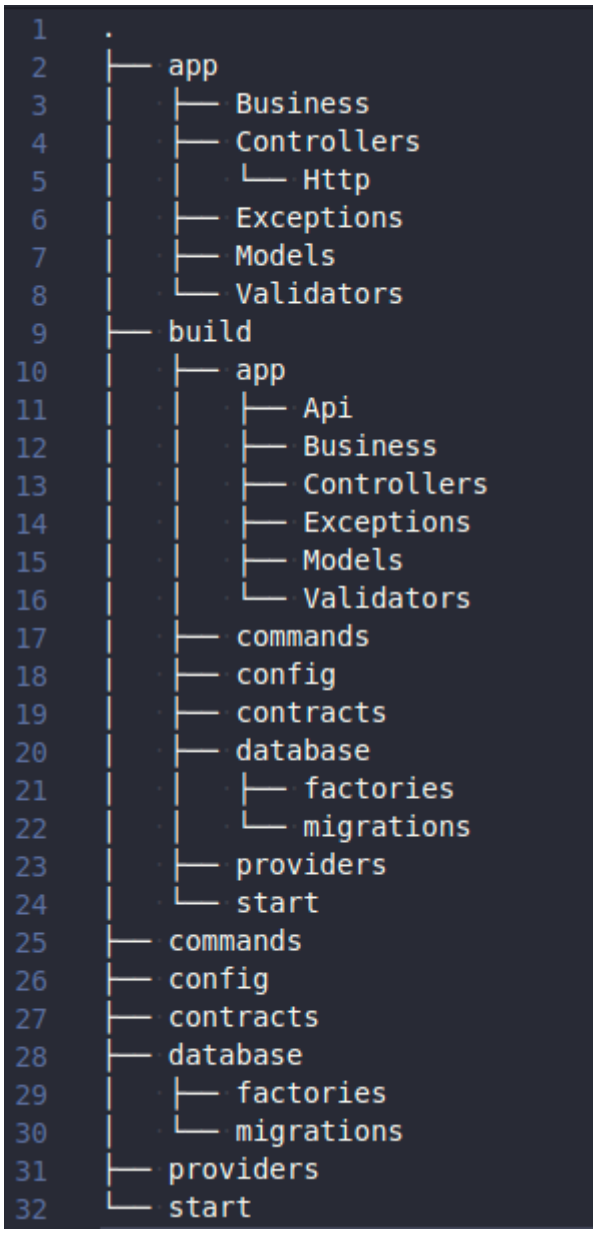
Tabela 2 — Requisitos Não Funcionais server (conclusão)

Código	Tarefa	Descrição	versão
	Acesso	do usuário	
RFN05	Não permitir cadastro de usuário duplicado	validar os dados de email e telefone e login antes de efetuar o cadastro de usuário.	2.0
RFN06	Documentação da API	Todos os endpoints deverão ser documentados, para facilitar a integração de sistemas	2.0

Fonte: O autor (2021)

8.1.1 **Estrutura de Pastas Server**

Figura 16 — Estrutura de diretório server



Fonte: O autor (2021)

8.2 **LIMPACITY_PRODUCER**

O Producer é uma API ligada no sistema de mensageria (Fila Rabbit MQ), que recebe os dados e os insere na fila. O nome faz referência a um produtor de mensagens.

Tabela 3 — Requisitos funcionais producer

Código	Tarefa	Descrição	versão
RF01	Deve receber os dados da coleta	Recebe o código do posto de coleta e as observações e cria mensagem no RabbitMQ	1.0
RF02	Gerar um UUID da coleta	para identificação de cada coleta será criado um código unico que será usado para registrar a mensagem	1.0
RF03	Recebe dados do aceite da coleta	deve receber o código uuid e gerar uma mensagem, ou de aceite ou de recusa, dependendo do status	1.0

Fonte: O autor (2021)

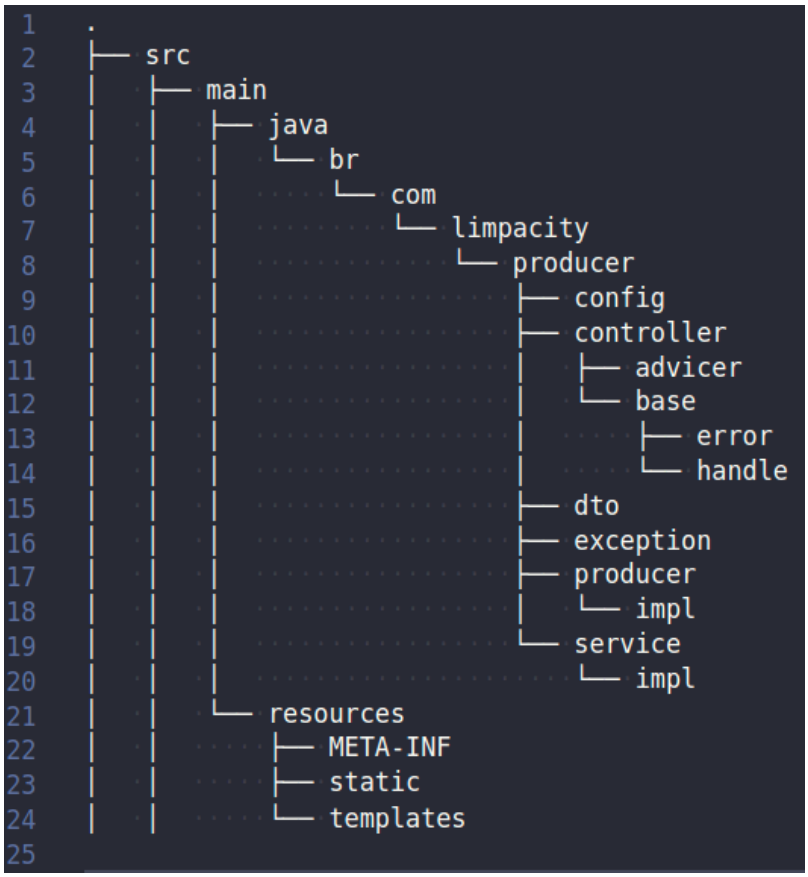
Tabela 4 — Requisitos não funcionais do producer

Código	Tarefa	Descrição	versão
RNF01	Utilizando Spring Boot criar a estrutura de API	Criar a API para receber os dados formatados e dentro do padrão para criação das mensagens	1.0
RNF02	Criar uma fila no Rabbit para gerenciamento das coletas	Deve criar uma estrutura de fila e exchange no Rabbit para receber mensagens de solicitação de coleta	1.0
RNF03	Instalar OpenAPI	Deverá ser configurado a dependência do Swagger, para implentar o padrão OpenAPI	1.0

Fonte: O autor (2021)

8.2.1 **Estrutura de diretório producer**

Figura 17 — Estrutura de diretório producer



Fonte: O autor (2021)

8.3 **LIMPACITY_LISTENER**

Este sistema é um consumidor da fila de mensagens, ele "ouve" a fila de coleta e insere os dados no banco mysql.

Tabela 5 — Requisitos funcionais listener

Código	Tarefa	Descrição	versão
RF01	Ouvir a fila de Solicitação de Coleta	deverá ficar conectado na fila de solicitação e coleta, estruturar os dados e gravar no banco mysql	1.0

Fonte: O autor (2021)

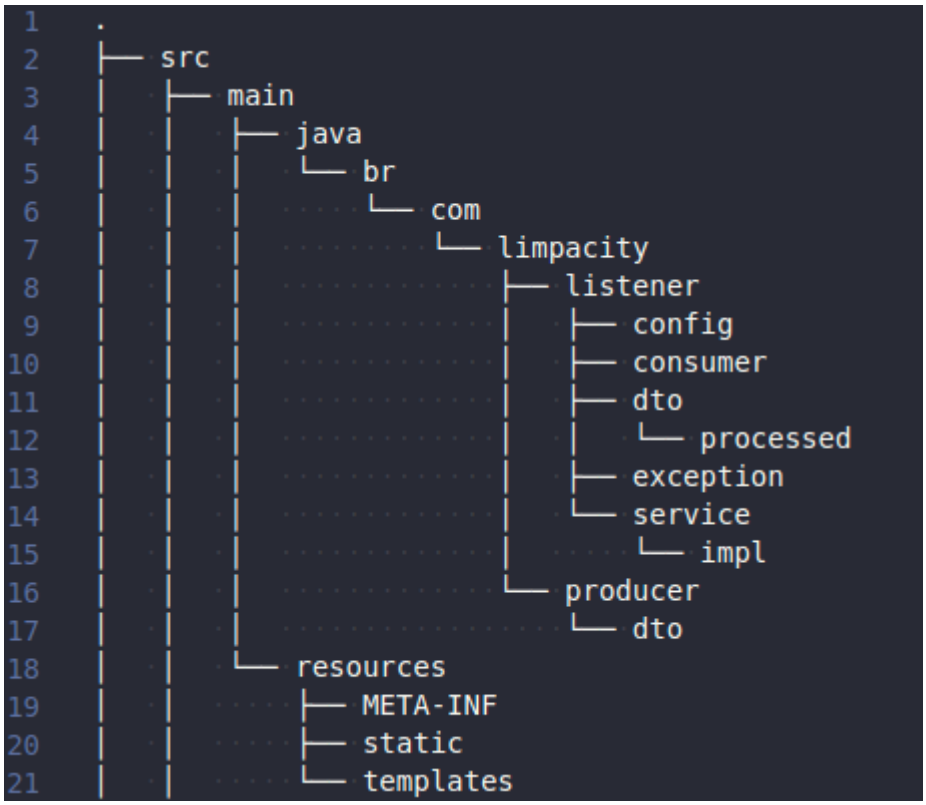
Tabela 6 — Requisitos não funcionais listener

Código	Tarefa	Descrição	versão
RNF01	Criar Listener com Spring boot	Utilizando spring boot com dependência ampq criar o listener para ficar conectado ouvindo a fila.	1.0
RNF02	Instalar sistema de Health Check	Deverá ter a dependência spring-boot-starter-actuator para checagem da saúde da aplicação	1.0
RNF03	Instalar OpenAPI	Deverá ser configurado a dependência do Swagger, para implentar o padrão OpenAPI	1.0

Fonte: O autor (2021)

8.3.1 **Estrutura de diretório listener**

Figura 18 — Estrutura de diretório listener



Fonte: O autor (2021)

8.4 **LIMPACITY_LISTENER_MAIL**

Aplicação para ouvir a fila de mensagem de e-mail e enviá-los

Tabela 7 — Requisitos funcionais listener mail

Código	Tarefa	Descrição	versão
RF01	Deverá ouvir a fila d notificação de E-mail	Criar um listener para ficar ouvindo a fila de notificação de email	1.0
RF02	Deverá enviar email pelo sistema	Criar uma classe para envio de emails com os dados da mensagem	1.0

Fonte: O autor (2021)

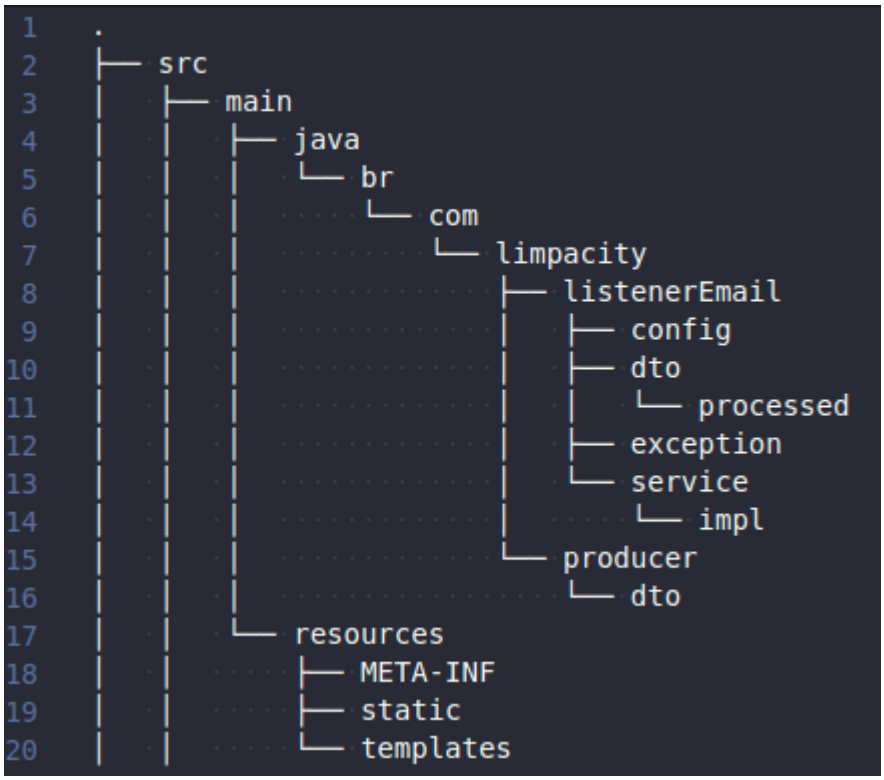
Tabela 8 — Requisitos não funcionais listener mail

Código	Tarefa	Descrição	versão
RNF01	Criar Listener com Spring boot	Utilizando spring boot com dependência ampq criar o listener para ficar conectado ouvindo a fila.	1.0
RNF02	Instalar sistema de Health Check	Deverá ter a dependência spring-boot-starter-actuator para checagem da saúde da aplicação	1.0
RNF03	Instalar OpenAPI	Deverá ser configurado a dependência do Swagger, para implentar o padrão OpenAPI	1.0

Fonte: O autor (2021)

8.4.1 **Estrutura de diretório listener email**

Figura 19 — Estrutura de diretório listener email



Fonte: O autor (2021)

8.5 **LIMPACITY_LISTENER_PUSH**

Listener Push ouve a fila de notificação de push notification e as envia.

Tabela 9 — Requisitos funcionais listener push

Código	Tarefa	Descrição	versão
RF01	Deverá ouvir a fila d notificação de Push Notification	Criar um listener para ficar ouvindo a fila de Push Notification	2.0
RF02	Deverá enviar push notification pelo sistema	Criar uma classe para envio do push notification de acordo com os dados da mensagem	2.0

Fonte: O autor (2021)

Tabela 10 — Requisitos não funcionais listener push

Código	Tarefa	Descrição	versão
RNF01	Criar Listener com Spring boot	Utilizando spring boot com dependência ampq criar o listener para ficar conectado ouvindo a fila.	2.0
RNF02	Instalar sistema de Health Check	Deverá ter a dependência spring-boot-starter-actuator para checagem da saúde da aplicação	2.0
RNF03	Instalar OpenAPI	Deverá ser configurado a dependência do Swagger, para implentar o padrão OpenAPI	1.0

Fonte: O autor (2021)

Observação: Não foi desenvolvido na versão 1.0 porque todo o microsserviço está na versão 2.0.

8.6 **LIMPACITY_API**

A API recebe os dados e os os insere no banco. Nela também existem algumas regras do negócio para manter a integridade dos dados.

Embora a solicitação de coleta passe pela fila, na API também é possível inserir essas informações de forma síncrona, isso porque este sistema é que faz a interface com o banco de dados.

Tabela 11 — Requisitos funcionais API (continua)

Código	Tarefa	Descrição	versão
RF01	Inserir cadastro de coletor	Deve permitir inserir um novo coletor	1.0
RF02	Alterar cadastro de coletor	Deve permitir alterar os dados do coletor	1.0
RF03	Remover coletor	Deve permitir inativar um coletor removendo a permissão de usar o sistema	1.0
RF04	Buscar coletor	Deve permitir fazer uma busca pelo nome do coletor	1.0
RF05	Inserir cadastro de Estação	Deve inserir uma nova estação com dados de endereço para localização	1.0
RF06	Alterar cadastro de Estação	Altera as informações da estação pelo código uuid	1.0
RF07	Remover Estação	Remove estação pelo código uuid	1.0
RF08	Buscar Estação	Busca estação com base na descrição	1.0
RF09	Inserir cadastro de Posto de Coleta	Insere um novo posto de coleta vinculando material e estação de coleta	1.0
RF10	Alterar cadastro de Posto de Coleta	Com base no código uuid do posto de coleta deve ser possível alterar os dados.	1.0
RF11	Remover Posto de	Ao inserir o código uuid deve executar o soft delete,	1.0

Tabela 11 — Requisitos funcionais API (conclusão)

Código	Tarefa	Descrição	versão
	Coleta	inativando o registro no sistema	
RF12	Buscar Posto de Coleta	Busca simples dos postos de coletas	1.0
RF13	Buscar Posto de Coleta com Estações, Coletor e Material	deve fazer a busca completa do posto de coleta e seus dados de material, endereço e coletor.	1.0
RF14	Insere uma nova solicitação de coleta com código do posto	Deve inserir uma solicitação de coleta somente informando o código uuid do posto e opcionalmente enviar uma observação	1.0
RF15	Marca o aceite na coleta	Com base no código uuid dar o aceita na solicitação, seja sim ou não.	2.0
RF16	Buscar todas as coletas em aberto	Deve fazer uma busca pelas solicitações de coleta disponíveis para serem coletadas, ativas e dentro do prazo especificado.	1.0
RF17	Buscar coleta pelo uuid	deve buscar os dados da coleta de acordo com o código uuid	1.0
RF18	Buscar nome do municipio e retornar se está habilitada	Deve inserir o nome do município e consultar no banco de dados se o registro está como habilitado = verdadeiro	1.0
RF19	Inserir cadastro de Material	Deve permitir o cadastro de material e subgrupo não permitindo duplicação de registros.	1.0
RF20	Alterar cadastro de Material	Deve permitir alterar cadastro de material com base no código	1.0
RF21	Remover Material	Deve receber o código do material e inativá-lo, para não ser possível usar me novas coletas.	1.0
RF22	Buscar Material	Deve buscar os materiais cadastrados no sistema.	1.0
RF23	Busca estação por município	deve fazer uma busca no cadastro de município conforme o nome informado	2.0
RF24	Validação de material	Deve verificar se o material já está cadastrado	2.0

Fonte: O autor (2021)

Tabela 12 — Requisitos não funcionais da API (continua)

Código	Tarefa	Descrição	versão
RNF01	Inserir todos os estados	Deve inserir os estados brasileiros com baso no código do ibge	1.0
RNF02	Inserir todos municípios	Deve inserir todos os municípios brasileiros com base no cadastro do ibge.	1.0
RNF02	Instalar sistema de Health Check	Deverá ter a dependência spring-boot-starter-actuator para checagem da saúde da aplicação	1.0

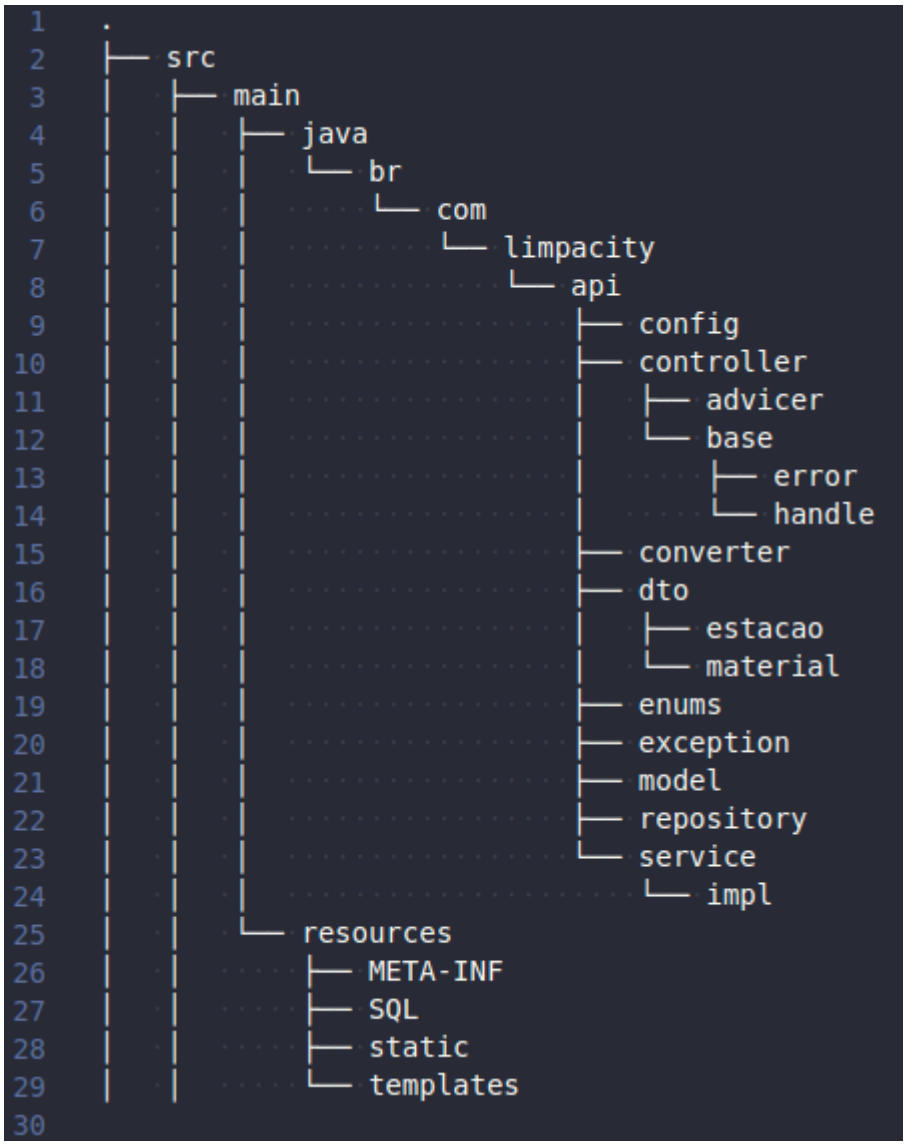
Tabela 12 — Requisitos não funcionais da API (conclusão)

Código	Tarefa	Descrição	versão
RNF03	Instalar OpenAPI	Deverá ser configurado a dependência do Swagger, para implentar o padrão OpenAPI	1.0

Fonte: O autor (2021)

8.6.1 **Estrutura de diretório API**

Figura 20 — Estrutura de diretório api



Fonte: O autor (2021)

8.7 **LIMPACITY_WORKER**

Este worker realiza de tempos em tempos a verificação de coletas não realizadas no banco de dados, e dispara novas notificações na fila de e-mail e push notification.

Tabela 13 — Requisitos funcionais worker

Código	Tarefa	Descrição	versão
RF01	Verifica solicitações em aberto	Deve fazer consultas no banco de dados de tempos em tempos e conforme parâmetros fazer envio de novas notificações	1.0

Fonte: O autor (2021)

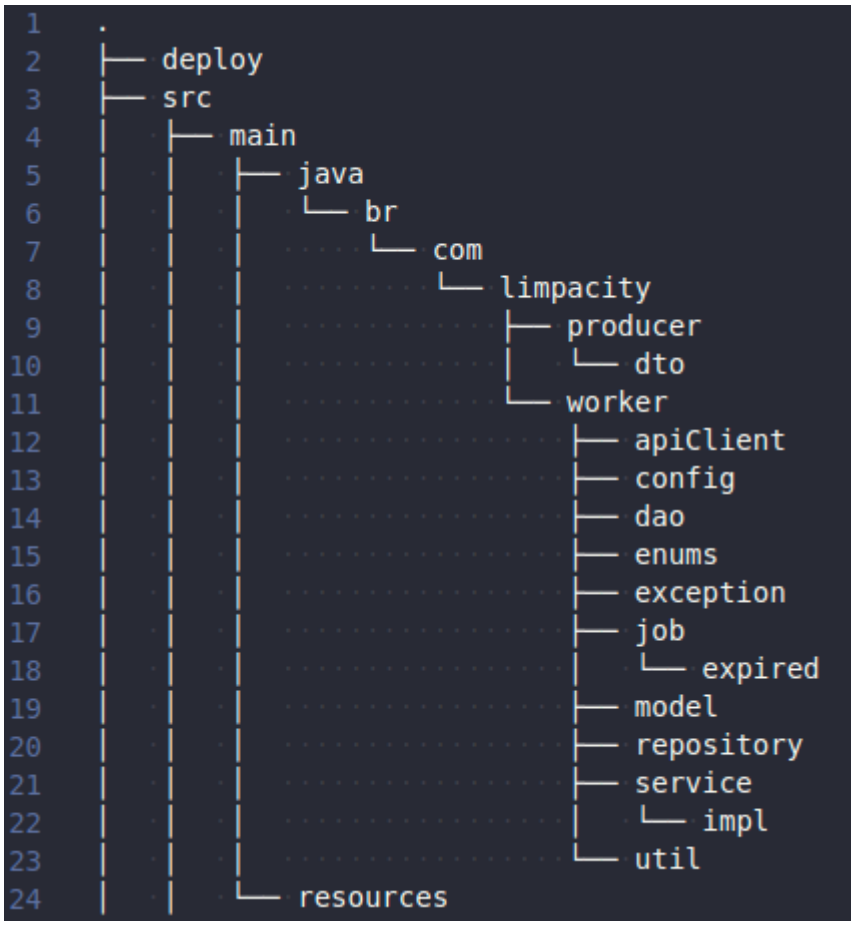
Tabela 14 — Requisitos não funcionais worker

Código	Tarefa	Descrição	versão
RNF01	Criar estrutura baseado no Spring Batch	Deve criar uma estrutura de jobs para executar a cada 10 segundos e enviar mensagens para fila através do producer	1.0
RNF02	Instalar sistema de Health Check	Deverá ter a dependência spring-boot-starter-actuator para checagem da saúde da aplicação	1.0

Fonte: O autor (2021)

8.7.1 **Estrutura de diretório Worker**

Figura 21 — Estrutura de diretório worker



Fonte: O autor (2021)

8.8 **LINK PARA OS REPOSITÓRIOS**

https://github.com/johnnyvaz1/limpacity_server
https://github.com/johnnyvaz1/limpacity_producer
https://github.com/johnnyvaz1/limpacity_listener
https://github.com/johnnyvaz1/limpacity_listener_email
https://github.com/johnnyvaz1/limpacity_api
https://github.com/johnnyvaz1/limpacity_worker

9 **CONSIDERAÇÕES FINAIS**

A criação de um sistema para gestão de solicitação de coletas, baseado em microsserviço, usando a metodologia de mensageria para o fluxo assíncrono, não foi uma tarefa fácil. Inicialmente, a estrutura dos microsserviços era bem diferente, mas com o passar do desenvolvimento foi-se percebendo a necessidade de mudança. Destacamos um ponto que foi removido, o qual foi o cacheamento das informações com um banco baseado em memória (Redis); este recurso iria aumentar a complexidade do projeto, mas não faria sentido, uma vez que o maior contato com o usuário será através do envio de mensagens. Alguns problemas encontrados durante o desenvolvimento custaram um bom investimento de tempo para sua solução, e isto gerou um aprendizado muito útil. O desenvolvimento foi feito de forma que funcionasse o fluxo principal de coleta, porém os próximos passos e novos recursos também dependerão do parceiro que irá integrar o seu frontend neste projeto. Isto ainda será objeto de pesquisa e desenvolvimento. Como um MVP, "produto mínimo viável" este projeto atende ao que promete para avançar na negociação desta parceria.

REFERÊNCIAS

7 BENEFÍCIOS de Microserviços e Arquitetura Escalável. Disponível em: https://www.deal.com.br/blog/7-beneficios-de-microservicos-e-arquitetura-escalavel/. Acesso em: 17 mai. 2021.

ALVES, Isabela. **40,1% do lixo produzido no Brasil é descartado de forma incorreta**. **Observatório do Terceiro Setor**. Disponível em: https://observatorio3setor.org.br/noticias/401-do-lixo-produzido-no-brasil-e-descartado-de-forma-incorreta/. Acesso em: 17 mai. 2021.

AMARO, Daniel. **97% do lixo produzido no Brasil não é reciclado**. **Edição do Brasil**. Disponível em: http://edicaodobrasil.com.br/2020/01/31/97-do-lixo-produzido-no-brasil-nao-e-reciclado/. Acesso em: 16 mai. 2021.

FABRO, Clara. **O que é API e para que serve? Cinco perguntas e respostas**. Disponível em: https://www.techtudo.com.br/listas/2020/06/o-que-e-api-e-para-que-serve-cinco-perguntas-e-respostas.ghtml. Acesso em: 16 mai. 2021.

INTERFACE de programação de aplicações. **Wikipedia**. 2021. Disponível em: https://pt.wikipedia.org/wiki/Interface_de_programa%C3%A7%C3%A3o_de_aplica%C3%A7%C3%B5es. Acesso em: 17 mai. 2021.

KUMAR, Amit. **Circuit Breaker Patter**. **DZone**. 2021. Disponível em: https://dzone.com/articles/circuit-breaker-pattern. Acesso em: 17 mai. 2021.

MARQUES FERNANDES, Henrique. **O que é um sistema/aplicação Monolito/Monolítica?**. Disponível em: https://marquesfernandes.com/tecnologia/o-que-e-um-sistema-aplicacao-monolito-monolitica/. Acesso em: 16 mai. 2021.

MINISTÉRIO DO MEIO AMBIENTE. **600 lixões desativados menos de um ano após sanção do Marco do Saneamento**. **gov.br/mma**. Disponível em: https://www.gov.br/mma/pt-br/assuntos/noticias/600-lixoes-desativados-menos-de-um-ano-apos-sancao-do-marco-do-saneamento. Acesso em: 17 mai. 2021.

MOTTER, Andressa. **Mais de 3 mil cidades brasileiras mantêm lixões a céu aberto**. **Folha de S. Paulo**. Disponível em: https://www1.folha.uol.com.br/seminariosfolha/2020/09/mais-de-3-mil-cidades-brasileiras-mantem-lixoes-a-ceu-aberto.shtml. Acesso em: 17 mai. 2021.

O QUE É A TECNOLOGIA Java e porque preciso dela?. **java**. Disponível em: https://www.java.com/pt-BR/download/help/whatis_java.html. Acesso em: 17 mai. 2021.

O QUE É API?. **Canal Tech**. Disponível em: https://canaltech.com.br/software/o-que-e-api/. Acesso em: 16 mai. 2021.

O QUE É UMA API. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-uma-api/. Acesso em: 15 mai. 2021.

OPENAPI.ORG. **Sobre OpenAPI**. Disponível em: https://www.openapis.org/about.

Acesso em: 15 mai. 2021.

PERLOW, Jason. **Containers: Fundamental to the cloud's evolution**. **ZDNet**. Disponível em: https://www.zdnet.com/article/containers-fundamental-to-the-evolution-of-the-cloud/. Acesso em: 17 mai. 2021.

SERVER.ORG. **Versionamento Semântico 2.0.0**. Disponível em: https://semver.org/lang/pt-BR/. Acesso em: 15 mai. 2021.

STUB. 2021. Disponível em: https://pt.wikipedia.org/wiki/Stub. Acesso em: 17 mai. 2021.

THE PRINCIPLES of OOD. **butunclebob**. Disponível em: http://www.butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod. Acesso em: 17 mai. 2021.