

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**  
**NÚCLEO DE EDUCAÇÃO A DISTÂNCIA**  
**Pós-graduação *Lato Sensu* em Arquitetura de Software Distribuído**

**Felipe Ralph da Costa Santos**

**GESTÃO DE SERVIÇOS DE LOGÍSTICA (GSL)**

Belo Horizonte

2022

**Felipe Ralph da Costa Santos**

**GESTÃO DE SERVIÇOS DE LOGÍSTICA (GSL)**

Trabalho de Conclusão de Curso de Especialização  
em Arquitetura de Software Distribuído como  
requisito parcial à obtenção do título de especialista.

Orientador: Professor Dr. Pedro Alves de Oliveira

Belo Horizonte

2022

## RESUMO

A empresa de logística Boa Entrega está passando por mudanças e deseja se manter relevante no mercado, que, atualmente, contém diversos novos concorrentes e desafios. A empresa contém uma *stack* tecnológica que necessita de atualização e da inclusão de novas ferramentas. Neste trabalho, apresentamos a proposta arquitetural GSL, que é escalável, distribuída, fácil de modificar e evoluir. Ela é baseada em microserviços, interoperável, via API, com mobile e Web, e conta com ferramentas de BI e *Big Data*. Os serviços são autônomos e se comunicam via mensagens pelo Kafka. Adicionalmente, são pequenos, tolerantes a falha e simples de manter. A proposta arquitetural GSL busca resolver os problemas do negócio da Boa Entrega, mas deixa uma série de decisões em aberto, para o futuro, com o objetivo comportar a evolução do negócio.

**Palavras-chave:** arquitetura de software, projeto de software, requisitos arquiteturais, microserviços, API, mensageria, serviços autônomos.

## SUMÁRIO

1	Apresentação.....	5
1.1	A competitividade do mercado.....	5
1.2	O problema.....	6
1.3	O objetivo do trabalho.....	6
1.4	Definições e Abreviaturas.....	7
2	Especificação da Solução.....	9
2.1	Identificação dos elementos do domínio.....	9
2.1.1	Organização dos subdomínios em módulos.....	10
2.1.2	Contexto do subdomínio principal, o <i>core</i> .....	10
2.1.3	Os atores do domínio.....	11
2.2	Requisitos Funcionais.....	12
2.2.1	Módulo de Serviços ao Cliente.....	14
2.2.2	Módulo de Informações cadastrais.....	14
2.2.3	Módulo de Gestão e Estratégia.....	15
2.2.4	Módulo de Ciência de Dados.....	15
2.3	Requisitos Não Funcionais.....	16
2.3.1	Objetivos do negócio.....	16
2.3.2	Atributos de qualidade.....	16
2.3.3	Cenários de estímulo e resposta dos atributos de qualidade.....	17
2.4	Restrições Arquiteturais.....	20
2.5	Mecanismos Arquiteturais.....	20
3	Modelagem Arquitetural.....	22
3.1	Macroarquitetura.....	22
3.2	Descrição Resumida das Histórias de Usuário.....	23
3.3	Visão Lógica.....	24
3.3.1	Diagrama de Classes.....	24
3.3.2	Diagrama de Componentes.....	24
3.3.3	Diagrama de Implantação.....	29
4	Prova de Conceito (POC) e Protótipo Arquitetural.....	31
4.1	Implementação.....	31
4.2	Interfaces e APIs.....	35
5	Avaliação da Arquitetura.....	37
5.1	Análise das abordagens arquiteturais.....	37
5.2	Cenários.....	37
5.3	Evidências da Avaliação.....	38
5.3.1	Cenário 1 - Desempenho.....	38
5.3.2	Cenário 2 - Manutenibilidade.....	41
5.3.3	Cenário 3 - Testabilidade.....	42
5.3.4	Cenário 4 - Tolerância a falhas.....	45
5.4	Resultados.....	47
6	Conclusão.....	49
	Referências.....	50
	APÊNDICES.....	54

# 1 Apresentação

A Boa Entrega é uma empresa de logística com abrangência nacional. Ela atua na modalidade rodoviária, possui centenas de empresas clientes e conta com um catálogo de milhares de rotas de entrega. Tem uma fatia de mercado considerável, estando posicionada como uma empresa de logística de grande porte.

Sua atuação é transportar a mercadoria do remetente ao destinatário, da coleta até a entrega, passando por diversos pontos de armazenamento em uma rota predefinida. O fluxo de movimentação se inicia na coleta da mercadoria em um cliente do tipo *e-commerce*, por exemplo. Em seguida, a mercadoria se torna uma encomenda e percorre diversos pontos de armazenamento até atingir a unidade mais próxima ao destinatário para uma posterior entrega. A movimentação da encomenda entre os pontos de armazenamento é realizado por empresas de transporte rodoviário, sendo que todas as mercadorias são seguradas por se tratarem de bens de terceiros. Todas as empresas de transporte e armazenamento são parceiros da Boa Entrega.

## 1.1 A competitividade do mercado

Durante o período da pandemia de COVID-19 ocorreu um aumento da demanda por entregas e mais concorrentes surgiram. Diante um ambiente mais disputado, a Boa Entrega se posicionou com o objetivo de se tornar mais eficiente e manter sua relevância no mercado.

A empresa definiu 6 metas para o triênio com foco na sustentabilidade e gestão financeira, demandando uma ampliação da capacidade de acompanhamento dos índices de negócio. As metas são as seguintes:

1. Realizar todas as entregas com tempo médio inferior a 5 dias úteis;
2. Expandir a atuação para mais 200 municípios de pequeno/médio porte;
3. Passar a atuar na região norte do Brasil, último reduto que ela ainda não cobre, fazendo parcerias com uma empresa aérea e outras empresas de transporte terrestre local;

4. Desenvolver novas parcerias com outras transportadoras, visando complementar sua atuação em lugares onde ela apresenta *marketshare* inferior a 10%;
5. Fazer convênio com no mínimo 50 novos clientes, preferencialmente do ramo supermercadista;
6. Crescer 10% em termos de faturamento global.

## 1.2 O problema

O catálogo tecnológico da Boa Entrega contém ferramentas de apoio a operação nas áreas administrativas, financeiro, gestão de serviços e operacional. São ferramentas legadas, que pecam no quesito escalabilidade e elasticidade. Elas possuem dificuldade de integrar com outras tecnologias de mercado mais recentes e não disponibilizam suporte para dispositivos móveis.

Outro ponto negativo é a inexistência de recursos de monitoramento para área financeira acompanhar o andamento do negócio e a evolução das metas, prejudicando a tomada de decisões.

## 1.3 O objetivo do trabalho

O objetivo deste trabalho é apresentar uma proposta arquitetural capaz de adequar a *stack* tecnológica atual da Boa Entrega para suportar as metas definidas para o triênio. A proposta visa melhorias para as ferramentas em uso e a inclusão de outras, como *Business Intelligence* (BI) e *Big Data*, as quais entregarão os dados para os indicadores de operação e diversos *insights* que visam colaborar com as tomadas de decisões de negócio.

A proposta do projeto arquitetural é denominada Gestão de Serviços de Logística (GSL). A GSL contempla todos os requisitos e tecnologias que serão utilizadas agrupados nos seguintes módulos:

- **Módulo de Informações Cadastrais:** desempenha o papel de obter e manter informações de todos os prestadores e associados, dentre as quais se

destacam: informações pessoais, de localização, de formação, de saúde e outras necessárias ao negócio da empresa;

- **Módulo de Serviços ao Associado:** desempenha o papel de desenhar, analisar e acompanhar todos os processos existentes na empresa, tanto os já existentes quanto os que ainda serão implantados, desta forma melhorando o desempenho e a eficiência dos mesmos. É uma solução baseada em *workflow*, com o uso de *Business Process Management* – BPM;
- **Módulo de Gestão e Estratégia:** desempenha o papel de prover a gestão estratégica de todos os projetos, produtos e serviços da empresa, com indicadores do andamento individual e global dos projetos e serviços, na forma de um *cockpit*;
- **Módulo de Ciência de Dados:** desempenha o papel de obtenção, guarda, recuperação e utilização dos dados corporativos pertinentes, com recursos para tratamento de dados massivos (*Big Data*), mineração dos dados de saúde e apoio às tomadas de decisão. É o módulo de *Data Warehouse* (DW) e *Business Intelligence* (BI).

## 1.4 Definições e Abreviaturas

- **API:** *Application Programming Interface*;
- **AWS:** *Amazon Web Services*;
- **BI:** *Business Intelligence*;
- **BPM:** *Business Process Management*;
- **DW:** *Data Warehouse*;
- **ESB:** *Enterprise Service Bus*;
- **GISA:** Gestão Integral da Saúde do Associado;
- **ORM:** *Object-Relational Mapping*;

- **SQL:** *Structured Query Language*;
- **POC:** *Proof of Concept*;
- **RNF:** Requisito Não-Funcional;
- **US:** *User story*;
- **HTTP:** *Hypertext Transfer Protocol*;
- **REST:** *Representational State Transfer*;
- **JSON:** *JavaScript Object Notation*;
- **SGBDs:** Sistemas Gerenciadores de Banco de Dados;
- **ATAM:** *Architecture Tradeoff Analysis Method*;
- **DDD:** *Test-Driven Development* ;
- **BDD:** *Behavior Driven Development*;
- **IaC:** *Infrastructure as Code*;
- **ACL:** *Anti-corruption Layer*;
- **JWT:** *JSON Web Token*;
- **ERP:** *Enterprise Resource Planning*;
- **GUI:** *Graphical User Interface*;
- **UI:** *User Interface*;
- **CI:** *Continuous Integration*;
- **CD:** *Continuous Delivery ou Continuous Deployment*;
- **QA:** *Quality Assurance*.



## 2 Especificação da Solução

Este capítulo tem como objetivo o entendimento do problema se baseando nos requisitos funcionais e não funcionais de acordo com as informações fornecidas pelo cliente e pela documentação.

### 2.1 Identificação dos elementos do domínio

Os dados fornecidos pelo cliente e pela documentação contém inúmeras informações sobre o negócio, porém sem refinamento que delimite os componentes do domínio. Utilizando DDD (ERIC EVANS, DDD) foram identificados os seguintes subdomínios: o *core*, o suporte e o genérico.

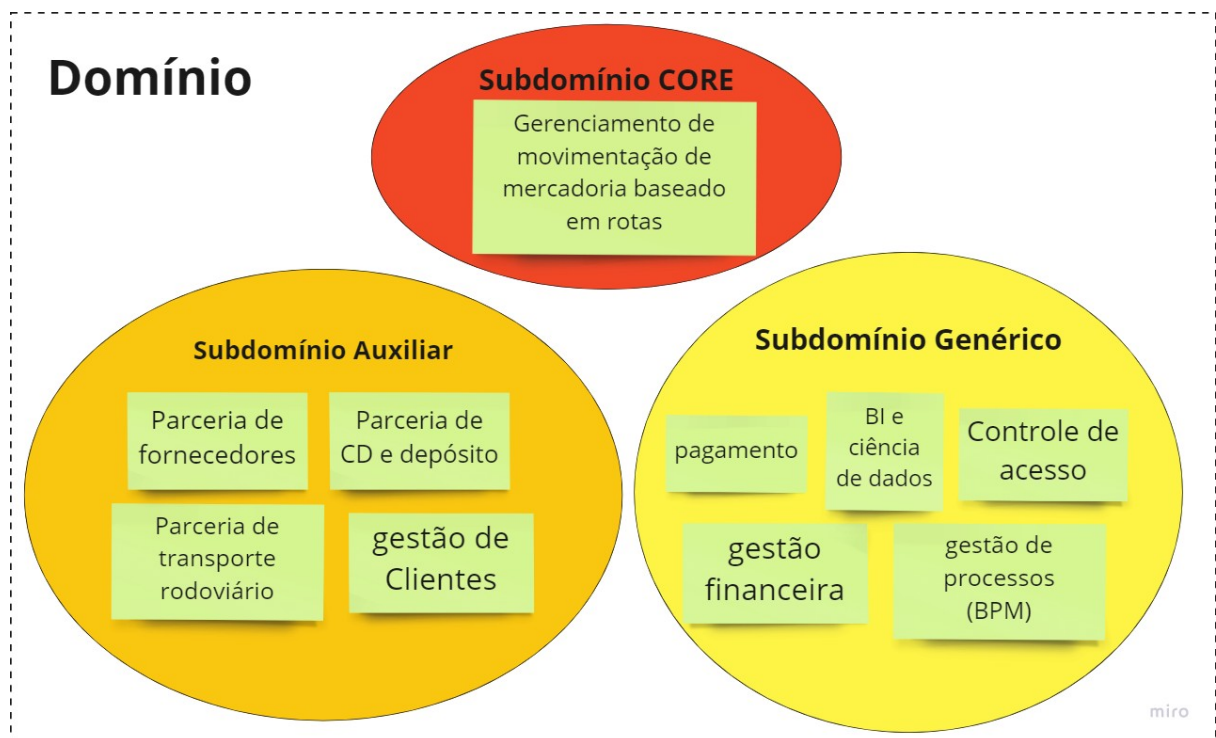


Figura 1: Domínio com os subdomínios do negócio Boa Entrega

Na Figura 1 apresenta o domínio com três subdomínios. O *core* do negócio é o gerenciamento de movimentação de mercadorias baseada em rotas. O subdomínio genérico agrupa as funções de gestão financeira, gestão de processos (BPM), BI e ciência de dados, pagamentos e controle de acesso. Por último, no conjunto do subdomínio auxiliar, estão as ferramentas para gerir parcerias com

empresas de transporte rodoviário e estoque, parceria com fornecedores e relação com empresas clientes.

### 2.1.1 Organização dos subdomínios em módulos

A prioridade no desenvolvimento da solução é o subdomínio *core*, seguido dos componentes do subdomínio auxiliar. O subdomínio genérico pode adotar soluções de prateleira, por exemplo o Power BI (MICROSOFT, PWRBI), IBM Business Process Manager (IBM, BPM), Totvs ERP Financeiro (TOTVS) e/ou contratar empresas de gestão de colaboradores e meios de pagamentos. Outra possibilidade, a qual vamos adotar é manter alguns componentes que já estão em uso, como o SAF e o SFC. A Figura 2 demonstra a relação entre o domínio de negócio e a modularização da solução. O controle de identidade e acesso é o KeyCloak (KEYCLOAK).

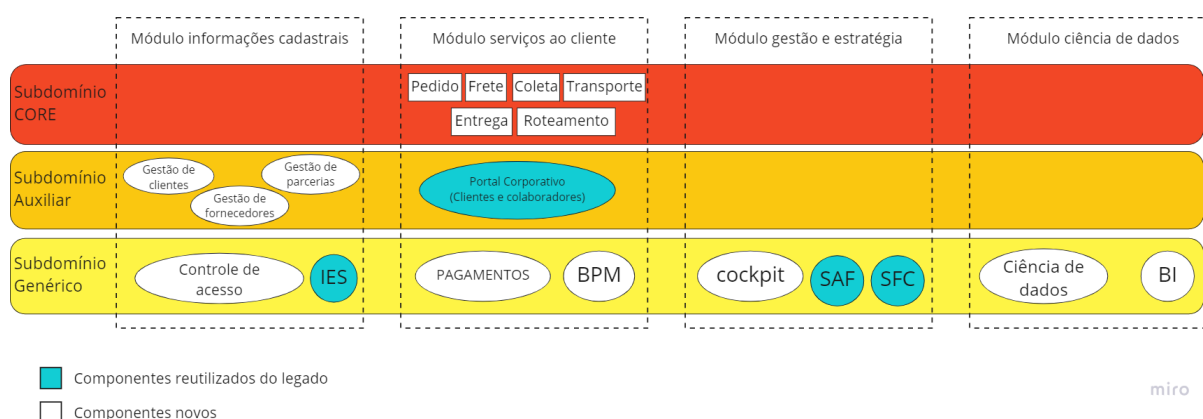


Figura 2: Visão dos módulos organizados em camadas do domínio de negócio

### 2.1.2 Contexto do subdomínio principal, o *core*

O *core* da aplicação contém os serviços de pedido, frete, coleta, transporte, entrega e roteamento. Os serviços de coleta, transporte e entrega são subprocessos do frete. O serviço de pedido tem a funcionalidade de contratar um frete. O serviço de frete é orquestrador e agregador de dados do processo de frete. O serviço de coleta gerencia o processo de coleta e preparação da mercadoria para o transporte. O serviço de transporte é o responsável por coordenar a movimentação da encomenda com deslocamentos e armazenamentos, até alcançar a área do

destinatário. O serviço de entrega gerencia a entrega da encomenda ao consumidor final. O serviço de roteamento contém informações de rotas disponibilizadas ao serviço de transporte.

A Figura 3 descreve a relação entre serviços e atores do subdomínio *core*. Esses são os serviços que devem ser atacados prioritariamente, pois são os principais do negócio.

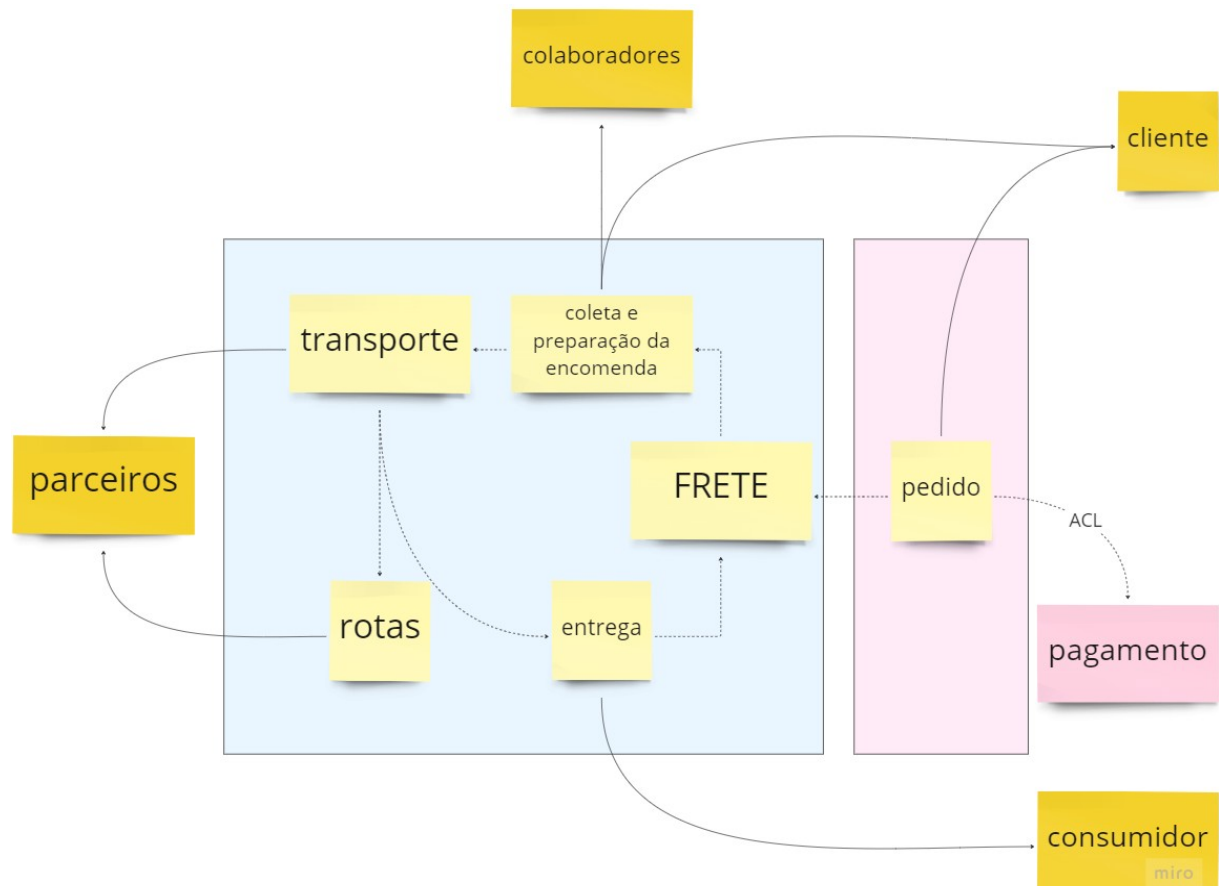


Figura 3: Organização dos agregados e atores

### 2.1.3 Os atores do domínio

Baseado nas informações do cliente e na documentação provida, os seguintes atores foram mapeados.

- **Cientes** – Empresas clientes do Boa entrega, exemplo: *e-commerces*. Os clientes que enviam a mercadoria aos consumidores;

- **Consumidores** – São as pessoas que compram nos *e-commerces* e recebem as mercadorias;
- **Parceiros** – Empresas parceiras que realizam transporte e armazenamento das encomendas. Parceiros também realizam a coleta e a entrega da mercadoria;
- **Colaboradores** – São pessoas que atuam realizando tarefas relacionadas ao processamento da mercadoria em encomenda, etapa antes do transporte, e em outras atividades mandatórias ao funcionamento da Boa Entrega. Também são denominados de associados;
- **Fornecedores** – Empresas que fornecem materiais para embalar as mercadorias, exemplo: plástico bolha, fita adesiva, etiquetas, caixas, etc.

## 2.2 Requisitos Funcionais

Os requisitos funcionais foram extraídos utilizando a técnica e *Event Storming* (BRANDOLINI, ES) (GERTEL, ES01) (GERTEL, ES02), como ilustrado na Figura 4. Segue a legenda:

- **Laranja** – Eventos;
- **Azul** – Comandos;
- **Roxo** – Política (*policy*)
- **Rosa** – Sistemas externos
- **Amarelo claro** – Agregados;
- **Amarelo escuro** – Atores;
- **Verde** – Saída para tela de visualização.

A construção deste gráfico tem o objetivo de entender quais são os eventos esperados e quem e para onde os comandos devem ser enviados.

## Atores

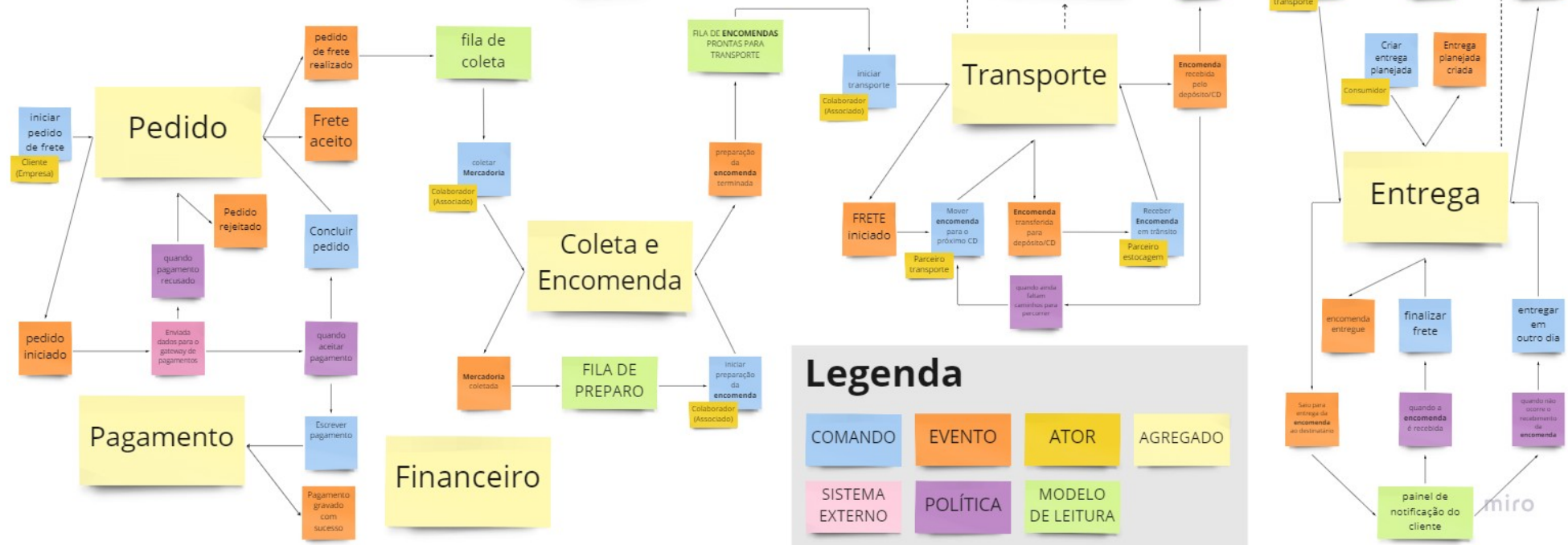
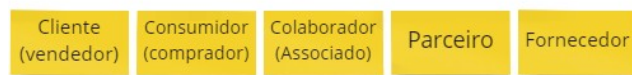


Figura 4: Mapeamento de agregados, eventos, comandos, política e modelo de leitura via técnica do Event Storming

Após o grafo completado, temos informações consistentes sobre os atores, comandos, eventos, políticas que observam eventos e utilização de recursos externos, sendo possível iniciar a confecção a solução.

A seguir, são apresentadas diversas tabelas com requisitos funcionais.

### 2.2.1 Módulo de Serviços ao Cliente

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF01	O sistema deve permitir que o cliente solicite transporte para uma ou mais mercadorias	A	A
RF02	O sistema deve prover ao cliente o acompanhamento do processo de pedido de transporte	M	M
RF03	O sistema deve prover ao destinatário (consumidor) o acompanhamento do processo de transporte da mercadoria	B	M
RF04	O sistema deve prover ao cliente o acompanhamento do processo de transporte das mercadorias	B	M
RF05	O sistema deve informar ao cliente quando a mercadoria foi entregue ao destinatário (consumidor)	M	A
RF06	O sistema deve informar ao destinatário (consumidor) quando a mercadoria saiu para entrega	M	A
RF07	O sistema deve permitir aos parceiros enviarem notificações sobre o estado das encomendas despachadas e recebidas, caso seja armazém, e em movimentação, caso seja transporte	A	A
RF08	O sistema deve permitir ao destinatário (consumidor) realizar o agendamento de entrega	B	B
RF09	O sistema deve informar ao colaborador sobre as mercadorias para a retirada nos clientes	M	A
RF10	O sistema deve informar ao colaborador sobre as mercadorias, já preparadas como encomendas, que devem ser repassadas ao parceiro do transporte	M	A
RF11	O sistema deve permitir ao colaborador gerenciar os processos de atendimento ao cliente (cliente e consumidor) via BPM	A	A

\*B=Baixa, M=Média, A=Alta.

### 2.2.2 Módulo de Informações cadastrais

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF12	O sistema deve permitir ao colaborador cadastrar clientes	B	A
RF13	O sistema deve permitir ao colaborador cadastrar parceiros	B	A
RF14	O sistema deve permitir ao colaborador cadastrar fornecedores	B	A
RF15	O sistema deve permitir ao parceiro atualizar suas informações cadastrais	M	M
RF16	O sistema deve permitir ao cliente atualizar suas informações cadastrais	M	M
RF17	O sistema deve permitir ao fornecedor atualizar suas informações cadastrais	M	M
RF18	O sistema deve consultar informações de clientes	B	A
RF19	O sistema deve consultar informações de parceiros	B	A
RF20	O sistema deve consultar informações de fornecedores	B	A
RF21	O sistema deve permitir que ao colaborador consultar informações sobre depósitos e mercadorias	B	A

### 2.2.3 Módulo de Gestão e Estratégia

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF22	O sistema prover ao colaborador realizar a gestão financeira empresa	A	A
RF23	O sistema prover ao colaborador realizar a gestão estratégica de entregas	A	A
RF24	O sistema prover ao colaborador acompanhar os indicadores dos objetivos estratégicos planejados para a empresa em forma de cockpit	M	M
RF25	O sistema prover ao colaborador visualizar todos os indicadores das entregas em andamento e realizadas em forma de cockpit	M	M

### 2.2.4 Módulo de Ciência de Dados

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
----	--------------------	----------------------	---------------------

RF26	O sistema permitir ao colaborador utilizar dados corporativos em <i>Big Data</i> para apoio às tomadas de decisão	A	B
RF27	O sistema permitir ao colaborador utilizar dados corporativos em mineração de dados para apoio às tomadas de decisão	A	B
RF28	O sistema permitir ao colaborador armazenar dados operacionais para serem analisados posteriormente	A	B

## 2.3 Requisitos Não Funcionais

Os requisitos não funcionais são apresentados a seguir.

### 2.3.1 Objetivos do negócio

Na tabela a seguir são apresentados os objetivos do negócio:

ID	Descrição	Prioridade B/M/A
RNF01	Suportar ambientes web e móveis	A
RNF02	Atender, de forma seletiva (por perfil) a clientes, fornecedores e colaboradores	A

### 2.3.2 Atributos de qualidade

A tabela a seguir apresenta os atributos de qualidade.

ID	Descrição	Prioridade B/M/A
RNF03	Estar disponível em horário integral (24H), sete dias por semana	A
RNF04	Ser de fácil implantação	A
RNF05	Apresentar bom desempenho	M
RNF06	Facilidade de modificação ou adaptação	A
RNF07	O software deve falhar durante a execução dos testes se possuir defeito	A
RNF08	Ser tolerante a falhas	A
RNF09	Possuir interface responsiva	M
RNF10	Ser de fácil utilização	A



### 2.3.3 Cenários de estímulo e resposta dos atributos de qualidade

- **RNF03** (Disponibilidade) - Estar disponível em horário integral (24H), sete dias por semana

<b>Estímulo</b>	O cliente solicita transporte para uma ou mais mercadorias em qualquer momento do dia
<b>Fonte do estímulo</b>	Cliente
<b>Ambiente</b>	Serviço em funcionamento com carga normal
<b>Artefato</b>	API REST do serviço de pedido
<b>Resposta</b>	Requisição retorna o <i>status code</i> de 2xx ou 4xx, ou seja, sucesso ou o cliente realizou a requisição incorreta
<b>Medida de resposta</b>	Porcentagem de sucesso nas respostas do serviço em relação as requisições

- **RNF04** (Portabilidade) - Ser de fácil implantação

<b>Estímulo</b>	Migração do sistema em uma outra nuvem qualquer
<b>Fonte do estímulo</b>	Equipe de engenharia de software
<b>Ambiente</b>	Ambiente de nuvem desconhecido
<b>Artefato</b>	Todos os serviços e banco de dados do sistema
<b>Resposta</b>	Deve ser possível realizar a implantação utilizando infraestrutura como código (IaC)
<b>Medida de resposta</b>	Tempo de implantação menor ou igual a 30 minutos

- **RNF05** (Desempenho) - Apresentar bom desempenho

<b>Estímulo</b>	Um cliente, parceiro ou colaborador realiza uma operação qualquer
<b>Fonte do estímulo</b>	Cliente, parceiro ou colaborador
<b>Ambiente</b>	Serviço em funcionamento com carga normal
<b>Artefato</b>	API REST de um serviço

<b>Resposta</b>	Requisição retorna sucesso ou erro entre 0,1 e 1 segundo
<b>Medida de resposta</b>	Intervalo de tempo entre 0,1 e 1 segundo em 85% dos casos

- **RNF06** (Manutenibilidade) - Facilidade de modificação ou adaptação

<b>Estímulo</b>	A equipe de engenharia deve corrigir um <i>bug</i> em produção
<b>Fonte do estímulo</b>	Equipe de engenharia
<b>Ambiente</b>	Ambiente de produção com carga normal
<b>Artefato</b>	Um serviço
<b>Resposta</b>	Deve ser possível subir uma nova versão do serviço sem impacto
<b>Medida de resposta</b>	O tempo da implementação e implantação de uma correção

- **RNF07** (Testabilidade) - O software deve falhar durante a execução dos testes se possuir defeito

<b>Estímulo</b>	Desenvolvedor realiza <i>commit</i> na <i>branch</i> Develop
<b>Fonte do estímulo</b>	Desenvolvedor
<b>Ambiente</b>	Repositório Git (GIT) e esteira do ambiente de desenvolvimento da nuvem
<b>Artefato</b>	O código de um serviço
<b>Resposta</b>	A esteira (pipeline) executando a <i>build</i> de um serviço e executando os testes
<b>Medida de resposta</b>	Os testes unitários, de integração e BDD devem ter 100% de aprovação

- **RNF08** (Tolerância a falhas) - Ser tolerante a falhas

<b>Estímulo</b>	Falha em um serviço
<b>Fonte do estímulo</b>	Diversos, podendo ser uma requisição via API ou troca de mensagens

	entre os serviços
<b>Ambiente</b>	Ambiente de produção com carga normal
<b>Artefato</b>	Um serviço do subdomínio core
<b>Resposta</b>	A falha não deve impactar no sistema por completo
<b>Medida de resposta</b>	Todas as trocas de mensagens entre os outros serviços devem ocorrer sem impacto e o serviço defeituoso deve ser reiniciado automaticamente

- **RNF09 (Usabilidade) - Possuir interface responsiva**

<b>Estímulo</b>	Clientes e parceiros interagem com o sistema via <i>smartphone</i> ou <i>notebook</i>
<b>Fonte do estímulo</b>	Clientes ou parceiros
<b>Ambiente</b>	<i>Smartphone</i> ou <i>notebook</i>
<b>Artefato</b>	Aplicativo móvel e Web
<b>Resposta</b>	O sistema deve permitir que aos clientes e parceiros interajam com o sistema via <i>smartphone</i> ou um <i>notebook</i> , sendo que interface de ambos devem ser adaptar para o tamanho de tela
<b>Medida de resposta</b>	Não deve ocorrer prejuízos nas funcionalidades por conta de um acesso mobile ou web e por diferentes tamanhos de tela

- **RNF10 (Usabilidade) - Ser de fácil utilização**

<b>Estímulo</b>	Cliente ou parceiro realizam uma operação no sistema
<b>Fonte do estímulo</b>	Cliente
<b>Ambiente</b>	<i>Smartphone</i> ou <i>notebook</i>
<b>Artefato</b>	Interface <i>mobile</i> , Web e API
<b>Resposta</b>	O sistema deve apresentar rotinas de forma simplificada para que os parceiros e clientes interajam com baixo custo cognitivo, ou seja, não fiquem dúvida se vão pressionar o botão A ou B
<b>Medida de resposta</b>	Baixo índice de ambiguidade, botões com diversas alternativas e textos muito longos. A API deve estar preparada para salvar estados parciais, no caso de formulários

## 2.4 Restrições Arquiteturais

As restrições arquiteturais identificadas estão listadas na tabela a seguir

R1	A escolha das tecnologias deve basear-se em produtos do mercado, com preferência para soluções de baixo custo e grande disseminação
R2	Utilizar APIs ou outros recursos adequados para consumo de serviços. O uso de <i>Application Programming Interfaces</i> (APIs) é mandatório
R3	Ser hospedado em nuvem híbrida, sendo a forma de hospedagem documentada.
R4	Ser desenvolvido utilizando recursos de gestão de configuração, com integração contínua
R5	Ser modular e componentizado, utilizando orientação a serviços
R7	Possuir características de aplicação distribuída: abertura, portabilidade, uso de recursos de rede
R8	Utilizar controle de versão Git

As restrições arquiteturais geralmente não são consideradas requisitos uma vez que limitam a solução, mas não constituem funcionalidades ou necessidades a serem satisfeitas. Ou seja, diferentemente dos requisitos elas servem para impor restrições que precisam ser satisfeitas.

## 2.5 Mecanismos Arquiteturais

A seguir, é apresentado a lista dos diferentes mecanismos arquiteturais propostos.

Análise	Design	Implementação
Persistência	ORM	Hibernate (HIBERNATE)
Front end	<i>Single Page Application</i>	Angular (ANGULAR) React (REACT) Vue (VUE)
Back end	<i>Microservices</i> (MICROSERVICES, MICRO)	Spring Boot (SPRING, BOOT) com Java (JAVA) e Kotlin (KOTLIN)
Integração	API e mensageria	API REST e Kafka (KAFKA)
Documentação de	<i>Schema</i>	Avro (AVRO)

mensagens		
Documentação de HTTP	API	SpringDoc (SPRING, DOC)
Interface com o cliente	API <i>gateway</i> (MICROSERVICES, APIGTWY) (MICROSERVICES, APICPSTN)	Kong (KONG)
Teste de Software	Unitários, integração e BDD	Junit (JUNIT) com Spring Boot Test (SPRING, BOOTTEST) e Postman (POSTMAN)
Banco de dados	SQL	PostgreSQL (POSTGRESQL)
Gerência e configuração	Git para código e registry para artefatos	Github (GITHUB) ou Gitlab (GITLAB) e Docker Registry (DOCKER, REGISTRY)
Escalabilidade e elasticidade	Orquestração de <i>containers</i>	Kubernetes (K8S)

### 3 Modelagem Arquitetural

Este capítulo apresenta diversos aspectos arquiteturais da solução proposta. Tais aspectos norteiam a implementação da prova de conceito, a qual será apresentada no capítulo 4 .

#### 3.1 Macroarquitetura

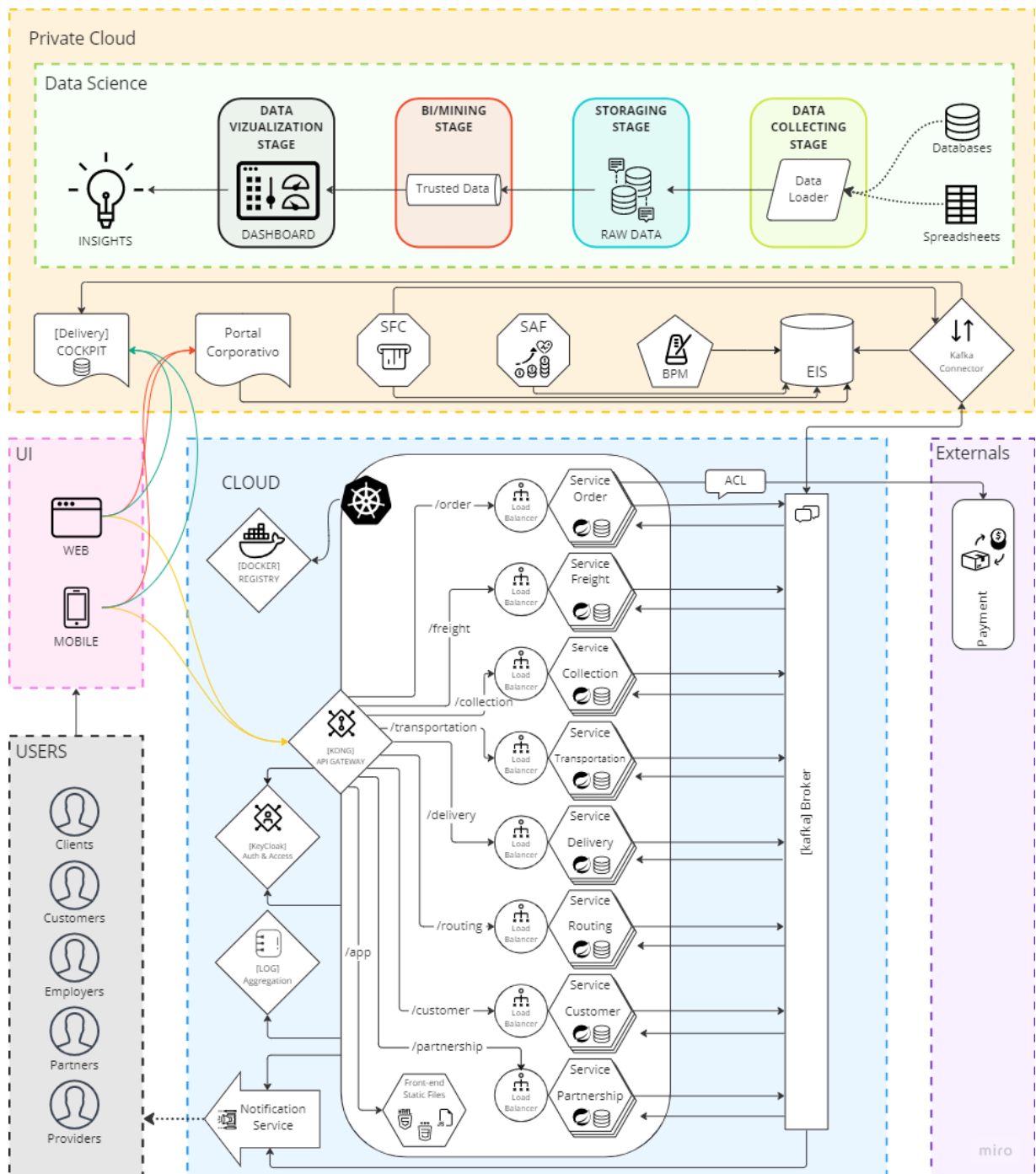


Figura 5: Visão Geral da Solução

A Figura 5 apresenta a especificação completa da solução proposta, com os contextos de nuvem comercial, nuvem privada, recursos externos, atores e o meio de interação dos atores com a solução. A especificação foi desenvolvida com em consonância com as restrições arquiteturais, com componentes confeccionados, de prateleira e recursos de externos de outras empresas. Cada serviço tem seu próprio banco de dados (MICROSERVICES, DBPERSVC).

### 3.2 Descrição Resumida das Histórias de Usuário

Nesta seção são apresentadas as histórias de usuário (US) que serão objeto de implementação na POC, capítulo 5 .

- **US1:** Como **Cliente**, eu quero **contratar um frete** para entregar uma mercadoria ao Consumidor;
  - US1.1: Como Cliente quero contratar frete por demanda e online para minha comodidade;
  - US1.2: Como Cliente quero ter a mercadoria retirada, após o frete contratado, por conta de uma necessidade do negócio;
  - US1.3: Como Cliente quero saber o estado do processo de entrega para informar ao consumidor em caso o mesmo solicite;
- **US2:** Como **Consumidor**, eu quero **acompanhar o processo de entrega** da mercadoria para poder me preparar para o recebimento;
  - US2.1: Como Consumidor, quero saber o momento da retirada;
  - US2.2: Como Consumidor, quero ter ciência da preparação da encomenda;
  - US2.3: Como Consumidor, quero acompanhar o transporte até minha localidade;
  - US2.4: Como Consumidor, quero saber quando saiu para entrega;
  - US2.5: Como Consumidor, quero confirmar o recebimento;
- **US3:** Como **Parceiro**, eu quero **registrar uma encomenda recebida e/ou em transporte** para poder realizar a cobrança posterior.
  - US3.1: Como Parceiro de transporte, quero registrar um transporte de uma encomenda;

- US3.2: Como Parceiro de armazenamento, quero registrar um armazenamento de uma encomenda.

### 3.3 Visão Lógica

Nesta seção é apresentada a especificação dos diagramas da solução proposta, com todos os seus componentes, propriedades e interfaces. Nas subseções a seguir são apresentados os diagramas de Classes, Componentes e Implantação.

#### 3.3.1 Diagrama de Classes

O diagrama de classes apresentado nesta seção demonstra apenas a relação entre as classes em um alto nível, sem detalhamento. A Figura 6 descreve o cliente (*Client*) e o consumidor (*Customer*) como os dois atores que enviam e recebem, respectivamente, uma mercadoria (*Product*).

O cliente realiza um pedido (*Order*) de frete para a mercadoria (*Product*). Após o pedido, a mercadoria é retirada (*PickupRequest*) por um colaborador (*Employee*) da Boa Entrega, junto ao cliente (*Customer*), e preparada para o envio. Neste momento, a mercadoria se torna uma encomenda (*Package*). A encomenda é repassada para diversos parceiros (*Partner*) de transporte (*Partner Store*) e armazenamento (*Partner Transporter*) até chegar no consumidor. O consumidor (*ClientCustomer*) tem por opção agendar um horário (*preferredDeliveryTime*) de entrega (*Delivery*). Todo o processo de deslocamento da mercadoria pode ser acompanhado em frete (*Freight*), via *status* (*FreightStatus*) e posicionamento (*currentPosition*).

#### 3.3.2 Diagrama de Componentes

A Figura 7 apresenta todos os componentes utilizados na solução. O contexto azul representa os componentes em nuvem e o contexto amarelo apresenta os componentes alocados em nuvem privada da Boa Entrega. Segue uma breve descrição dos componentes ilustrados na imagem.



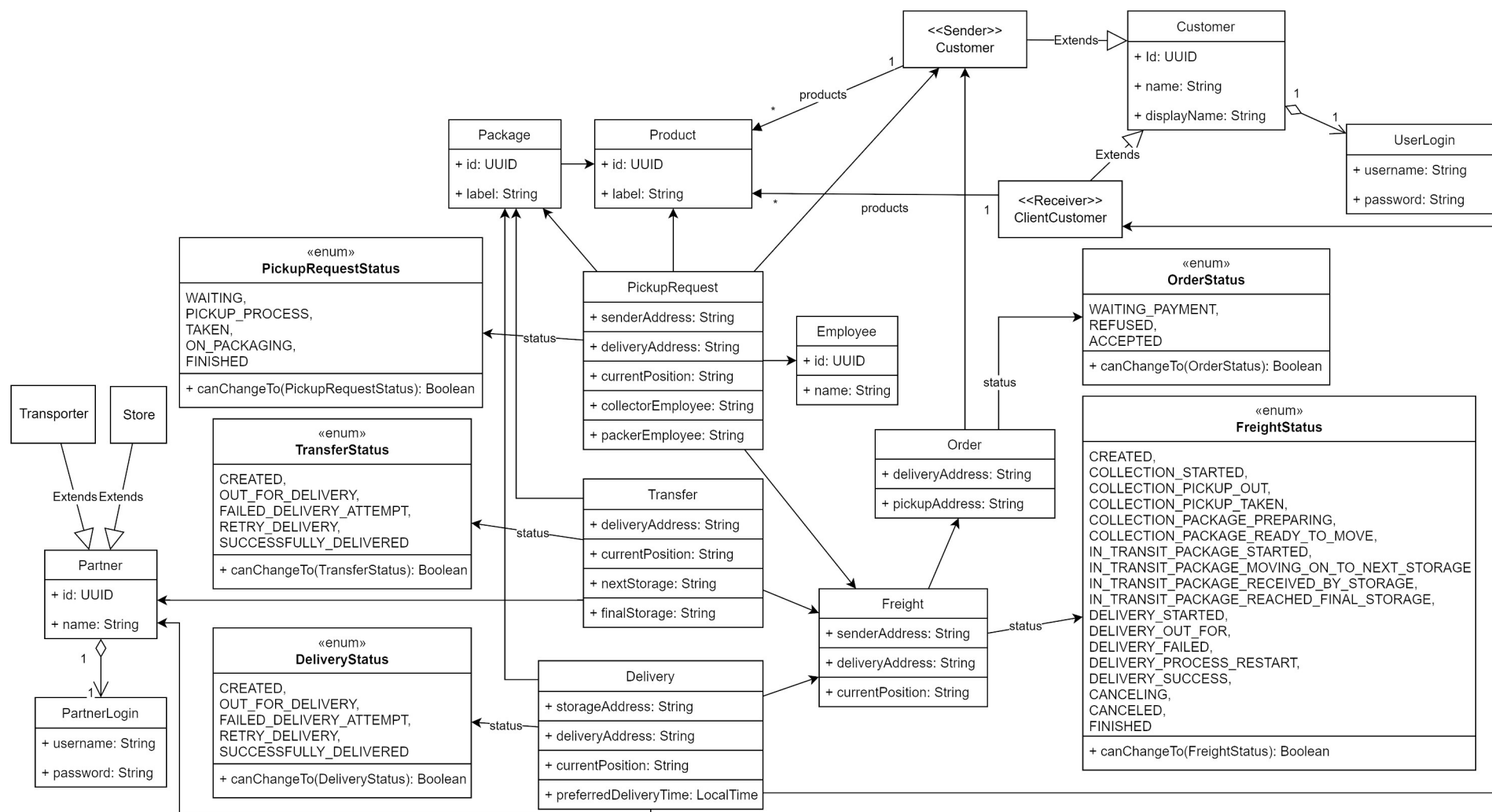


Figura 6: Diagrama de classes do core da solução em baixo detalhamento

- **Client App** – Aplicação Web e *mobile* que acessa uma API via REST;
- **Cloud** – Recursos alocados em nuvem do tipo: AWS, Google Cloud, Azure ou outra;
  - **Order** – Serviço implementado para realizar um pedido de frete;
  - **Freight** – Serviço implementado para coordenar o processo e agregar os dados de status dos fretes;
  - **Collection** – Serviço implementado para os colaboradores serem informados sobre as mercadorias pendentes de coleta no cliente e preparadas para o transporte;
  - **Transportation** – Serviço implementado para destinado ao processo de transporte da encomenda, interagindo com os parceiros de transporte e armazenamento;
  - **Delivery** – Serviço implementado destinado processo de entrega da mercadoria ao destinatário, com interações com o parceiro e o agendamento de horário para entrega pelo destinatário;
  - **Routing** – Serviço interno implementado que contém as todas as rotas predefinidas da Boa Entrega, atuando em conjunto com o serviço Transportation;
  - **Customer** – Serviço auxiliar implementado como uma interface para os clientes se comunicarem com a Boa Entrega;
  - **Partnership** – Serviço auxiliar implementado como uma interface para os parceiros de armazenamento e transporte se comunicarem com a Boa Entrega;
  - **Static Files** – Armazenamento para arquivos estáticos;
  - **Kubernetes** – O Kubernetes é um sistema de orquestração de contêineres *open-source* que automatiza a implantação, o dimensionamento e a gestão de aplicações em contêineres. Neste contexto, o componente gerenciará os serviços do subdomínio *core*;

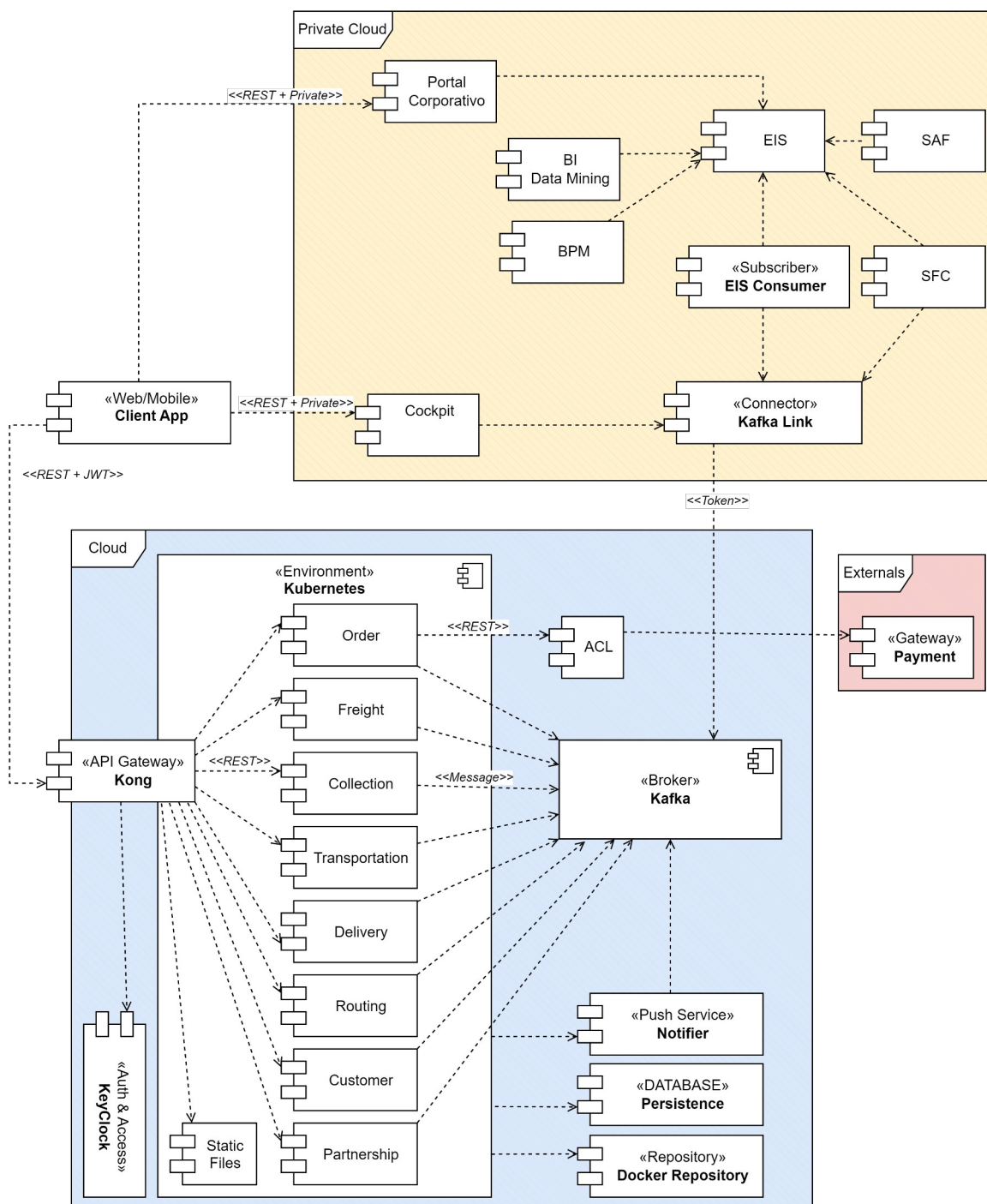


Figura 7: Diagrama de Componentes

- **Kong** – O Kong é um *API gateway open-source*, com a capacidade de verificar tokens JWT, via *plugin*. Este componente proverá acesso a API para todos os serviços alocados no *cluster* do Kubernetes;
- **KeyCloak** – O Keycloak é um produto *open-source* para permitir o login único com gerenciamento de identidade e acesso voltado para aplicativos

e serviços modernos. O componente será o responsável por autenticar e autorizar o acesso, via JWT, à API provida pelo Kong;

- **Kafka** – O Kafka é uma plataforma *open-source* de processamento de streams. Neste contexto, o componente desempenhará o papel de *message broker* (mensageria), facilitando a troca de mensagens entre os serviços implantados no Kubernetes e os componentes da nuvem privada;
- **Docker Repository** – É o repositório de imagens Docker (DOCKER), onde as imagens com os binários dos serviços implementados serão armazenadas;
- **Notifier** – É um serviço genérico de envio de notificações aos usuários;
- **ACL** – Camada anticorrupção (MICROSERVICES, ACL). Este componente é um *façade* para o serviço externo do *Payment*;
- **Persistence** – Componente de armazenamento de dados da nuvem;
- **External Services** – Serviços contratados de outras empresas;
  - **Payment** – *Gateway* de pagamento tipo mercado pago;
- **Private Cloud** – Serviços alocados em nuvem privada;
  - **SAF** – Componente legado Sistema Administrativo-Financeiro;
  - **SFC** – Componente legado Sistema de Faturamento e Cobrança;
  - **EIS** – Sistema de informação empresarial. Tem o objetivo de manter informações do cliente e fornecedores, bem como os depósitos e as mercadorias;
  - **EIS Consumer** – Componente implementado para se inscrever nos tópicos do Kafka e levar os dados gerados na nuvem para o IES no contexto privado;
  - **Kafka Link** – Conector para o Kafka implementado para ser utilizado dentro da nuvem privada;

- **Cockpit** – Um *dashboard* com as informações sobre entregas em andamento e as entregas realizadas;
- **Portal Corporativo** – *Front-end* legado onde colaboradores e clientes podem acessar recursos de rotas, entregas e veículos;
- **BI/Data Mining** – Agente para obter informações do EIS e alimentar a base de dados de BI e a mineração de dados;
- **BPM** – Componente de *Business Process Management* para desenhar, analisar e acompanhar todos os processos de atendimento ao cliente.

### 3.3.3 Diagrama de Implantação

A solução pode ser implantada em contextos híbridos, utilizando uma nuvem comercial e privada simultaneamente. A imagem a seguir, Figura 8, apresenta a implantação no contexto híbrido, com as representações físicas e suas ligações.

Os blocos azuis representam o hardware em nuvem comercial e os blocos amarelos representam o hardware utilizado em nuvem privada. O hardware pode ser um componente computacional físico ou virtualizado.

Os componentes do subdomínio *core* do negócio são implantados em nuvem comercial, dentro de um *cluster* Kubernetes, para que possam ser escalonados de acordo com a demanda. Os *Pods* são unidades de implantação do microsserviço, via imagem Docker, são independentes e gerenciados por um serviço dedicado (“K8S Service”). O *Node* é o local onde os *Pods* são alocados e ser duplicado diversas vezes. O Kubernetes também agrega o Kong como API *gateway*.

Ainda na nuvem comercial, temos um *cluster* Kafka para intermediar a troca de mensagens entre microsserviços no Kubernetes e entre os componentes alocados na nuvem privada. O microsserviço *Order* (pedido) utiliza um *gateway* de pagamentos externo, o qual possui uma interface de comunicação fora do padrão, obrigando a utilização de um *façade* (Camada Anticorrupção – ACL) (MICROSERVICES, ACL). O componente notificador (*Notifier*) realiza o *push* de mensagens aos usuários. E o repositório de imagens Docker armazena as imagens dos microsserviços que serão implantados nos *Pods* do Kubernetes.

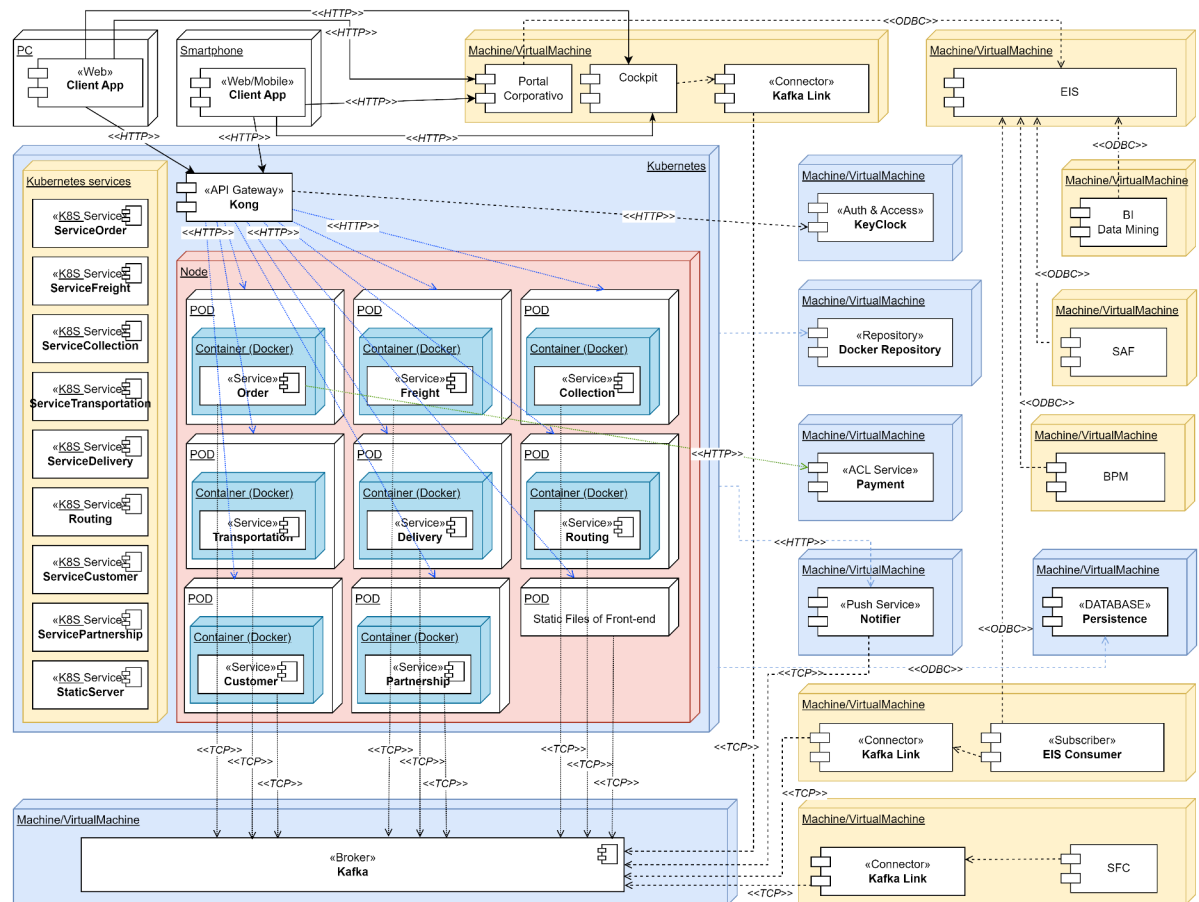


Figura 8: Diagrama de Implantação

As máquinas onde executam as aplicações clientes estão representadas por um PC/Notebook e um *smartphone*.

A estrutura da nuvem privada é composta de um hardware dedicado para os seguintes componentes: arquivos de *front-end*, *cockpit* e portal corporativo, e aos demais componentes.

## 4 Prova de Conceito (POC) e Protótipo Arquitetural

Este capítulo apresenta a POC desenvolvida com o intuito de verificar a viabilidade da proposta arquitetural apresentada neste documento. A seguir, serão demonstradas as evidências da avaliação do protótipo em relação aos requisitos não funcionais para: desempenho (RNF05), modificabilidade (RNF06) e testabilidade (RNF07).

O código fonte está disponível no Github, em <https://github.com/println/puc-tcc-gsl>.

### 4.1 Implementação

As seguintes histórias de usuários implementadas para a prova de conceito foram:

- **US1:** Como **Cliente**, eu quero **contratar um frete** para entregar uma mercadoria ao Consumidor;
  - US1.1: Como Cliente quero contratar frete por demanda e online para minha comodidade;
  - US1.2: Como Cliente quero ter a mercadoria retirada, após o frete contratado, por conta de uma necessidade do negócio;
  - US1.3: Como Cliente quero saber o estado do processo de entrega para informar ao consumidor em caso o mesmo solicite;
- **US2:** Como **Consumidor**, eu quero **acompanhar o processo de entrega** da mercadoria para poder me preparar para o recebimento;
  - US2.1: Como Consumidor, quero saber o momento da retirada;
  - US2.2: Como Consumidor, quero ter ciência da preparação da encomenda;
  - US2.3: Como Consumidor, quero acompanhar o transporte até minha localidade;
  - US2.4: Como Consumidor, quero saber quando saiu para entrega;
  - US2.5: Como Consumidor, quero confirmar o recebimento;

- **US3:** Como **Parceiro**, eu quero **registrar uma encomenda recebida e/ou em transporte** para poder realizar a cobrança posterior.
  - US3.1: Como Parceiro de transporte, quero registrar um transporte de uma encomenda;
  - US3.2: Como Parceiro de armazenamento, quero registrar um armazenamento de uma encomenda.

Os seguintes requisitos não funcionais aplicados na implementação da prova de conceito foram:

- **RNF05** (Desempenho)
  - Critérios de aceite:
    - As requisições HTTP são respondidas entre 0,1 e 1 segundo;
- **RNF06** (Manutenibilidade)
  - Critérios de aceite:
    - O sistema simples de modificar e implantar as modificações;
- **RNF07** (Testabilidade)
  - Critérios de aceite:
    - Devem existir testes unitários
    - Devem existir testes de integração;
    - Devem existir testes de BDD;
- **RNF08** (Tolerância a falhas)
  - Critérios de aceite:
    - Múltiplas instâncias do serviço no Kubernetes;
    - Fluxo linear, onde um serviço que já atuou não tem mais influência no restante do processo.

As tecnologias utilizadas foram:

- Kotlin – Linguagem de programação;
- Java – Linguagem de programação;
- Spring Boot – Framework Java/Koltin para o back-end;
- Gradle (GRADLE) – Gerenciador de dependências;
- Avro – *Schema* para mensagens utilizadas no Kafka;
- JUnit – Framework utilizado em testes unitário e integração;
- Kafka - Mensageria;



- Kafka UI (KAFKA, UI) – GUI para gerenciar o Kafka;
- Docker – Motor de *container*;
- Docker Compose – Gerenciamento de *containers*;
- PostgreSQL – Banco de dados;
- H2 (H2) – Banco de dados para testes;
- Git – Versionamento de código;
- Postman – Testes de BDD;
- Spring-Doc/Swagger-UI (SWAGGER) – Documentação de API;
- Kubernetes – Orquestrador de *containers*;
- Kong – API *gateway*;
- Konga (KONGA) – Interface para o Kong;
- SonarQube (SONAR) – Inspeção de qualidade de código;
- Ubuntu/Linux – Executar a implantação de teste
- JMeter (JMETER) - Testes de carga e de estresse.

A Figura 9 apresenta a estrutura do código fonte alocado no Github. A pasta `.dev` contém arquivos voltados para a criação do ambiente de desenvolvimento. As pastas `order`, `freight`, `collection`, `transportation` e `delivery` contém o código fonte dos serviços pedido, frete, coleta, transporte e entrega, respectivamente. A pasta `shared` é destinada para os códigos compartilhados entre os serviços. Os arquivos `docker-build-all.sh`, `docker-run-all.sh`, `docker-stop-all.sh`, `kubernetes-delete-all.sh` e `kubernetes-deploy-all.sh` estão relacionados com a execução dos serviços. O `docker-build-all.sh` é o responsável por gerar a imagem Docker de todos os serviços. O `docker-run-all.sh` e o `docker-stop-all.sh` são os responsáveis pela execução e interrupção dos serviços, eles existem apenas para testes. Para o fazer a implantação da imagem Docker no Kubernetes, deve ser utilizado o comando `kubernetes-deploy-all.sh`. E para limpar o Kubernetes de todas as criações realizadas, utiliza-se o comando `kubernetes-delete-all.sh`.

A Figura 10 ilustra a estrutura replicada para todos os serviços, com arquivos do Gradle para *build*, de implantação no Kubernetes (k8s) e o Dockerfile para gerar as imagens Docker. A pasta `src` contém o código fonte e os testes unitários e integrados.
















	println added environment variables in deployment o...	6081aad 2 days ago	 42 commits
	.dev	added kong initial config with services and routes	2 days ago
	collection	added environment variables in deployment on k...	2 days ago
	delivery	added environment variables in deployment on k...	2 days ago
	freight	added environment variables in deployment on k...	2 days ago
	order	added environment variables in deployment on k...	2 days ago
	shared	Added integration tests and outbox pattern in m...	22 days ago
	transportation	added environment variables in deployment on k...	2 days ago
	.gitignore	First commit	2 months ago
	docker-build-all.sh	added kubernetes deploy command	2 days ago
	docker-run-all.sh	added kubernetes deploy command	2 days ago
	docker-stop-all.sh	added kubernetes deploy command	2 days ago
	kubernetes-delete-all.sh	added kong initial config with services and routes	2 days ago
	kubernetes-deploy-all.sh	added kubernetes deploy command	2 days ago

Figura 9: Repositório do Github

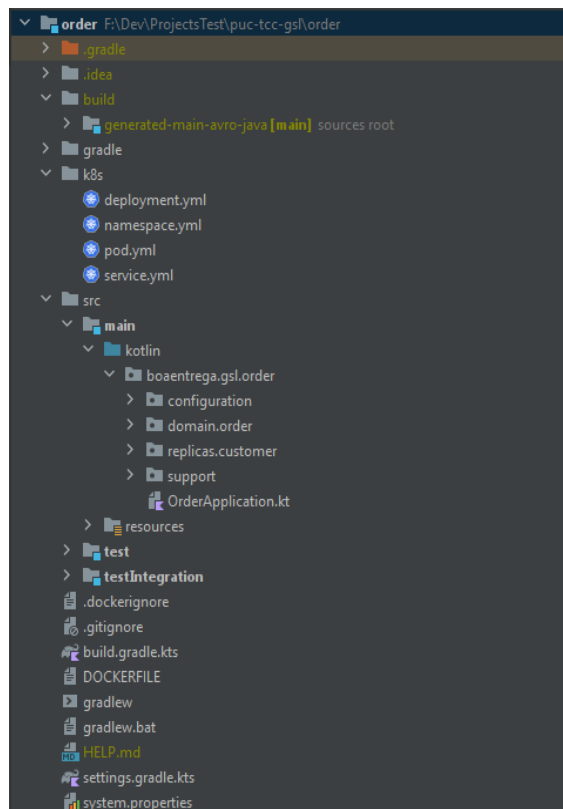


Figura 10: Estrutura física dos projetos

## 4.2 Interfaces e APIs

A POC possui interfaces para comunicação externa, do ator com o sistema, e interna, comunicação entre os serviços. A comunicação entre ator e sistema é utilizado uma API REST, como ilustrado na Figura 11. A figura contém os *endpoints* utilizados pelo Cliente, Colaborador, Parceiro e Consumidor.

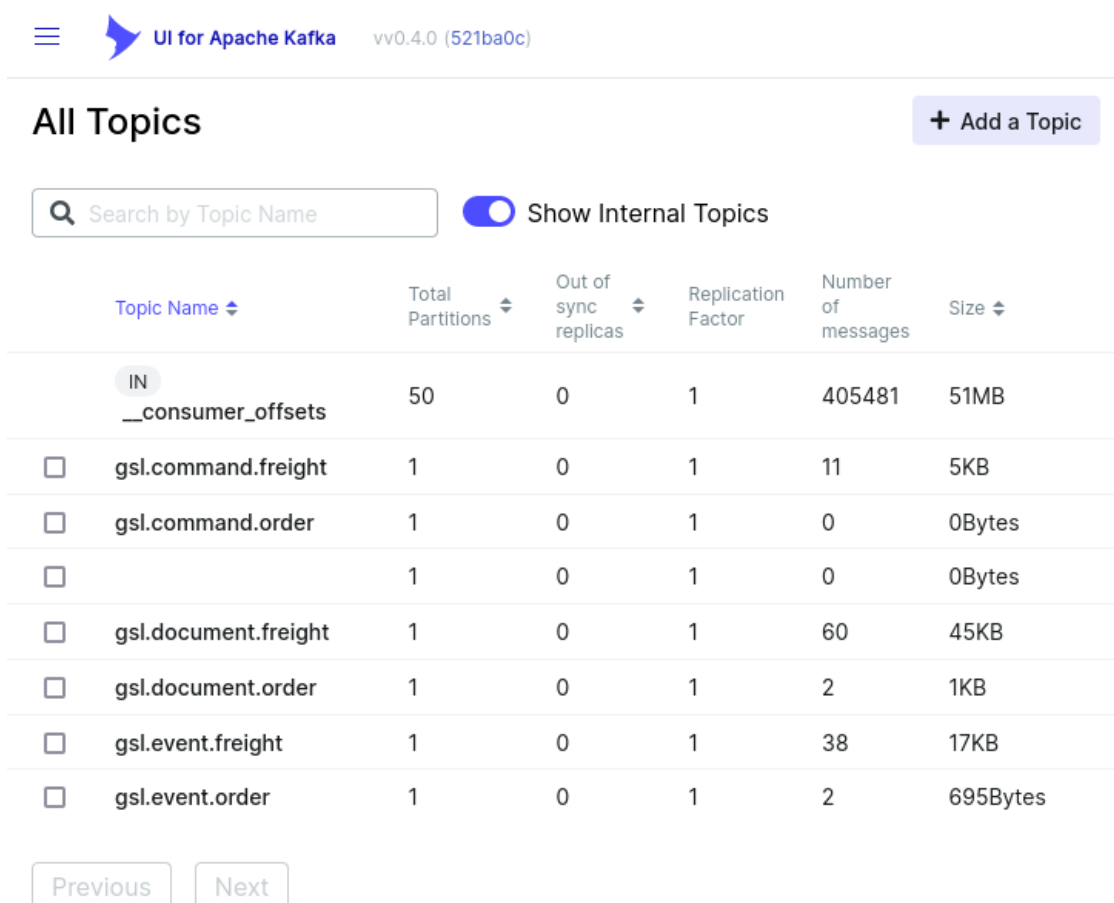
order-controller	collector-controller
PUT /order/{id}/refuse	PUT /collection/{id}/taken
PUT /order/{id}/approve	PUT /collection/{id}/ready
GET /order	PUT /collection/{id}/pickup
POST /order	PUT /collection/{id}/packaging
GET /order/{id}	GET /collection
GET /order/test	GET /collection/{id}
freight-controller	delivery-controller
GET /freight	PUT /delivery/{id}/time
GET /freight/{id}	PUT /delivery/{id}/package
	DELETE /delivery/{id}/package
transfer-controller	PUT /delivery/{id}/delivery
PUT /transport/{id}/transfer	DELETE /delivery/{id}/delivery
PUT /transport/{id}/receive	GET /delivery
GET /transport	GET /delivery/{id}
GET /transport/{id}	

Figura 11: Representação da API baseada no Swagger/OpenAPI

A comunicação entre os serviços é assíncrona, com troca de mensagens via Kafka. Os serviços podem enviar e receber comandos e eventos, seguindo o padrão *Event Sourcing* (ELEMAR, EVENT SOURCING) (MICROSERVICES, ES). Foi ainda utilizado o padrão *Saga* (ELEMAR, SAGA) (MICROSERVICES, SAGA) coreografado para organizar o fluxo de atividades, como, por exemplo, a criação de um frete após aprovação do pagamento do pedido. A Figura 12 apresenta a tela do Kafka UI com 6

filas, sendo 3 filas para o serviço pedido (*Order*) e 3 filas para o serviço frete (*Freight*). As filas são específicas para eventos, comandos e documentos.

O documento (CREDITAS, EVENT DRIVEN) é um *snapshot* do recurso REST, difundido via Kafka, para todos os interessados que desejam manter uma cópia atualizada. A combinação entre redundância de dados e o padrão Saga coreografado tem como objetivo habilitar a autonomia nos serviços e nos fluxos de atividade. Por exemplo, o frete é composto das etapas de coleta, transporte e entrega. Cada etapa tem seu próprio serviço com redundância dos dados do frete. Em caso de problema com serviço de frete, não haverá impacto nas demais atividades, pois os dados do frete estão redundantes nos serviços e cada um dispara um comando para iniciar a próxima etapa. O fim da coleta emite o comando para o início do transporte e o fim do transporte emite o comando para iniciar o processo de entrega.



UI for Apache Kafka vv0.4.0 (521ba0c)

## All Topics + Add a Topic

Search by Topic Name Show Internal Topics

Topic Name	Total Partitions	Out of sync replicas	Replication Factor	Number of messages	Size
IN __consumer_offsets	50	0	1	405481	51MB
<input type="checkbox"/> gsl.command.freight	1	0	1	11	5KB
<input type="checkbox"/> gsl.command.order	1	0	1	0	0Bytes
<input type="checkbox"/>	1	0	1	0	0Bytes
<input type="checkbox"/> gsl.document.freight	1	0	1	60	45KB
<input type="checkbox"/> gsl.document.order	1	0	1	2	1KB
<input type="checkbox"/> gsl.event.freight	1	0	1	38	17KB
<input type="checkbox"/> gsl.event.order	1	0	1	2	695Bytes

Previous Next

Figura 12: Filas no Kafka

## 5 Avaliação da Arquitetura

Este capítulo contém a avaliação da arquitetura. O objetivo é verificar se a arquitetura atende, de fato, o que foi proposto. A avaliação será realizada pela verificação dos atributos de qualidade em diferentes cenários.

### 5.1 Análise das abordagens arquiteturais

A avaliação é iniciada com a escolha de um conjunto de atributos de qualidade e definição dos níveis de importância e complexidade. O método *Architecture. Tradeoff Analysis Method* (ATAM) foi utilizado na avaliação dos cenários desta análise. A tabela a seguir contém os atributos de qualidade escolhidos, uma breve descrição dos cenários e os valores de importância e complexidade.

Atributos de Qualidade	Cenários	Importância	Complexidade
Desempenho	Cenário 1: Apresentar bom desempenho	M	M
Manutenibilidade	Cenário 2: O sistema deve possibilitar aplicar alterações de forma simples, mantendo a consistência com os demais requisitos	A	M
Testabilidade	Cenário 3: O software deve falhar durante a execução dos testes se possuir defeito	A	B
Tolerância a falhas	Cenário 4: O sistema deve se manter funcional quando serviços adjacentes param de funcionar	A	A

\*B=Baixa, M=Média, A=Alta.

### 5.2 Cenários

Esta seção contém os seguintes cenários:

- **Cenário 1** - Desempenho: Qualquer requisição ao sistema deve retornar sucesso ou erro em uma janela de tempo entre 0,1 e 1 segundo;

- **Cenário 2** - Manutenibilidade: Uma alteração deve ser fácil de implementar e implantar. Efeitos colaterais não devem existir para modificações centralizadas.
- **Cenário 3** - Testabilidade: O software deve falhar durante a execução do teste unitário, teste integrado ou teste de BDD, via Postman, se possuir defeito.
- **Cenário 4** - Tolerância a falhas: Caso o serviço de frete se torne inativo durante o fluxo de atividades, nenhum impacto deve ocorrer nos outros serviços.

### 5.3 Evidências da Avaliação

Esta seção apresenta uma avaliação geral da arquitetura indicando os pontos fortes e as limitações da arquitetura proposta. A avaliação utiliza os cenários definidos na Seção 5.2 e as evidências podem ser quantitativas ou qualitativas, desde que suportem o atendimento do requisito não-funcional.

#### 5.3.1 Cenário 1 - Desempenho

Atributo de Qualidade:	Desempenho
Requisito de Qualidade:	Apresentar bom desempenho
Preocupação:	
Qualquer requisição ao sistema deve retornar sucesso ou erro em uma janela de tempo entre 0,1 e 1 segundo	
Cenário(s):	
Cenário 1	
Ambiente:	
Sistema em operação normal, sem o processo de autenticação	
Estímulo:	
Um cliente, parceiro ou colaborador realiza uma operação qualquer	
Mecanismo:	
Através de um API <i>gateway</i> com o redirecionamento a para o serviço de pedido, frete, coleta, transporte ou entrega.	
Medida de resposta:	
O tempo da resposta HTTP em milissegundos	

Considerações sobre a arquitetura:	
Riscos:	Em períodos de muito acesso, podem ocorrer sobrecargas no API <i>gateway</i> e nos serviços, gerando um tempo de resposta fora da faixa ideal
Pontos de Sensibilidade:	Toda a parte de infraestrutura de hardware e de rede podem impactar esta medição.
Tradeoff:	Para mitigar o impacto da infraestrutura no tempo de resposta, pode ser necessário um esforço para otimizar a execução do serviço.

### Evidências:

NAME / ID	TAGS	HOSTS	SERVICE	PATHS	CREATED ^
freight-route		-	freight	/freight	Nov 27, 2022
collection-route		-	collection	/collection	Nov 27, 2022
delivery-route		-	delivery	/delivery	Nov 27, 2022
transportation-route		-	transportation	/transport	Nov 27, 2022
order-route		-	order	/order	Nov 27, 2022

Figura 13: Imagem com as rotas e serviços do Kong

	Start time	Source	Duration	All tests	Passed	Failed	Skipped	Avg. Resp. Time
✓	Nov 28, 2022 08:30:08	Runner	38m 59s	2200	2200	0	0	34 ms
	▶ Ran locally using Collection Runner · Local · 20 iterations completed							
✓	Nov 28, 2022 08:30:07	Runner	39m	2200	2200	0	0	34 ms
	▶ Ran locally using Collection Runner · Local · 20 iterations completed							
✓	Nov 28, 2022 08:30:05	Runner	38m 59s	2200	2200	0	0	32 ms
	▶ Ran locally using Collection Runner · Local · 20 iterations completed							
✓	Nov 28, 2022 08:30:04	Runner	39m 3s	2200	2200	0	0	32 ms
	▶ Ran locally using Collection Runner · Local · 20 iterations completed							

Figura 14: Relatório do Postman - 4 execuções em paralelo

A Figura 14 apresenta o relatório de execuções do Postman. Foi executado 4 instâncias em paralelo do teste de BDD, presente no código fonte, com 20 repetições, sendo 2 segundos de intervalo entre cada requisição. Esse tempo foi adicionado por conta do intervalo de execução do *schedule*, configurado para o padrão *Outbox (MICROSERVICES, OUTBOX)*, de 1 segundo. A média do tempo de resposta foi de 34 ms.

A tabela a seguir apresenta os tempos do segundo experimento com um teste de carga simples. Foi utilizado o JMeter e a configuração deste teste também está presente no repositório de código. Foram realizadas requisições para as rotas exibidas na Figura 13, ou seja, a listagem de recursos nos serviços de *Order*, *Freight*, *Collection*, *Transportation* e *Delivery*. Inicialmente foram 50 requisições simultâneas, depois aumentado para 100 e concluindo em 500. A tabela ainda contém a latência mínima, média e máxima das requisições. O pior caso de tempo de resposta foi de 22 ms com 50 usuários e de 84 ms com 100 usuários. Com 500 usuários, supõe-se que o hardware não suportou a demanda, gerando respostas com o tempo de 2,1 segundos.

Usuários	URL	Amostras	Latência (ms)		
			Média	Mínima	Máxima
50	GET /order	50	12	8	22
	GET /freight	50	9	7	17
	GET /collection	50	10	8	15
	GET /transport	50	9	8	14
	GET /delivery	50	10	7	17
	<b>TOTAL</b>	<b>250</b>	<b>10</b>	<b>7</b>	<b>22</b>
100	GET /order	100	22	8	78
	GET /freight	100	21	8	53
	GET /collection	100	23	9	84
	GET /transport	100	19	11	42
	GET /delivery	100	20	11	35
	<b>TOTAL</b>	<b>500</b>	<b>21</b>	<b>8</b>	<b>84</b>
500	GET /order	500	673	14	2109
	GET /freight	500	218	21	1184
	GET /collection	500	188	18	1183
	GET /transport	500	170	15	1183
	GET /delivery	500	153	10	1158
	<b>TOTAL</b>	<b>2500</b>	<b>280</b>	<b>10</b>	<b>2109</b>



### 5.3.2 Cenário 2 - Manutenibilidade

Atributo de Qualidade:	Manutenibilidade
Requisito de Qualidade:	Facilidade de modificação ou adaptação
Preocupação:	
Sistemas necessitam de ajustes, correções e evolução e essas alterações precisam ocorrer com menos atrito possível	
Cenário(s):	
Cenário 2	
Ambiente:	
Desenvolvimento, com IDE e elementos para testar a integração com os componentes	
Estímulo:	
Codificação e implantação de uma alteração evolutiva, correção de <i>bug</i> ou otimização	
Mecanismo:	
Codificação e teste	
Medida de resposta:	
O tempo gasto em todo o processo e complexidade da implementação	
Considerações sobre a arquitetura:	
Riscos:	O desenvolvedor precisa seguir os padrões estabelecidos para o projeto
Pontos de Sensibilidade:	O desenvolvedor precisa conhecer a arquitetura e o conceito de mensageria
<i>Tradeoff</i> :	Não tem

#### Evidências:

A primeira evidência é a Figura 15, com um painel do SonarQube listando todos os projetos analisados pela ferramenta. Na esquerda é possível observar o filtro para manutenibilidade nível “A” que representa o campo “*Code Smell*” em cada item da lista.

A segunda evidência é Figura 11, a qual apresenta quantidade de *endpoints* por serviço. É possível notar que a quantidade máxima de *endpoints* é 7.

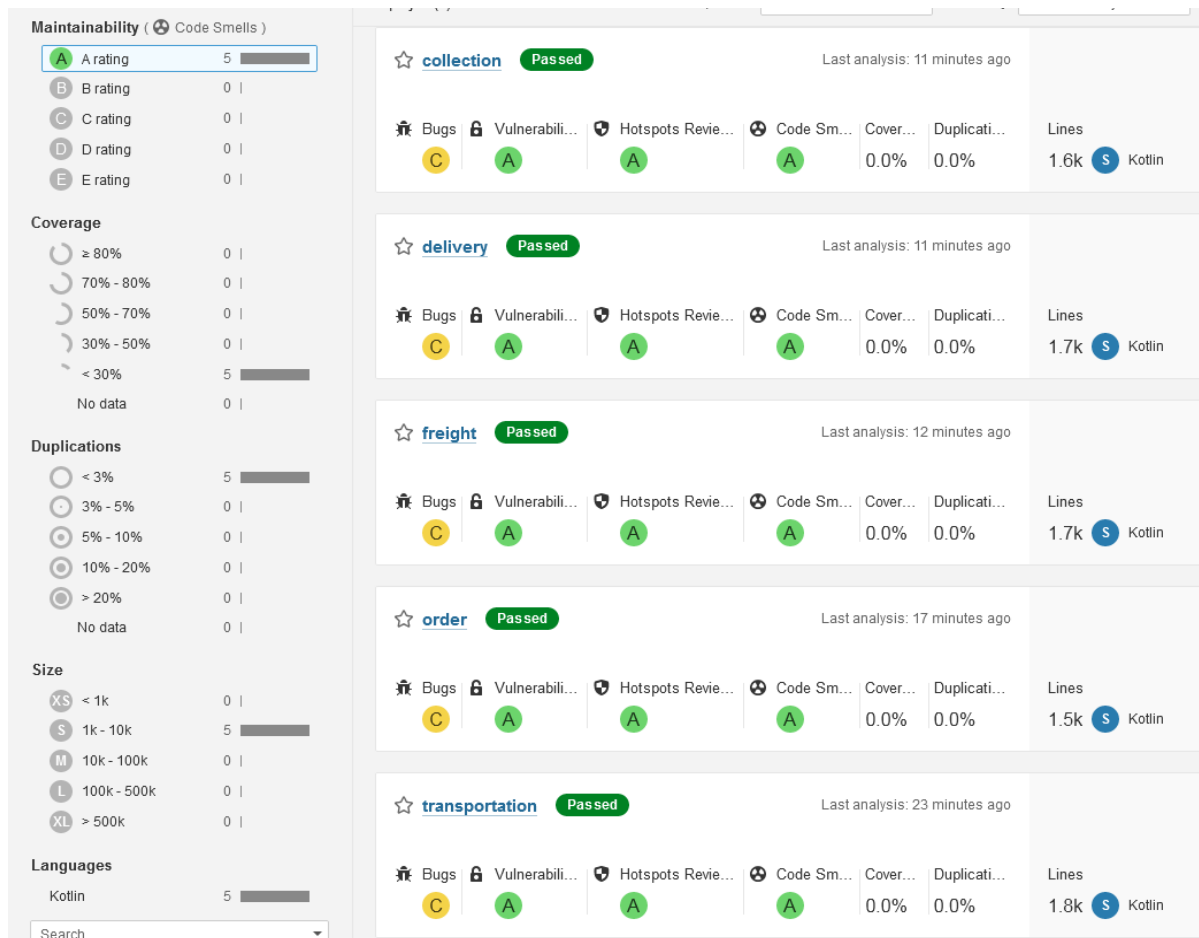


Figura 15: Relatório do SonarQube

### 5.3.3 Cenário 3 - Testabilidade

Atributo de Qualidade:	Testabilidade
Requisito de Qualidade:	O software deve falhar durante a execução dos testes se tiver defeito
Preocupação:	
O software deve falhar durante a execução do teste unitário, teste integrado ou teste de BDD, via Postman, se possuir defeito	
Cenário(s):	
Cenário 3	
Ambiente:	
Local, processo de <i>build</i> e implantado no ambiente de desenvolvimento	
Estímulo:	

Desenvolvedor, esteira e QA	
Mecanismo:	
Testes	
Medida de resposta:	
Se tiver algum defeito, o teste deve falhar	
Considerações sobre a arquitetura:	
Riscos:	Testes viciados
Pontos de Sensibilidade:	Não tem
Tradeoff:	Pode existir um balanço entre testes unitários e de integração

### Evidências:

A Figura 15 contém a captura de uma tela do Postman com o resultado dos testes de BDD . Este teste realiza o fluxo completo, iniciando pelo do pedido do frete, a coleta do produto, o transporte da encomenda, com 3 etapas de trânsito e 3 de etapas de armazenamento, e, ao final, a entrega da mercadoria ao destinatário.

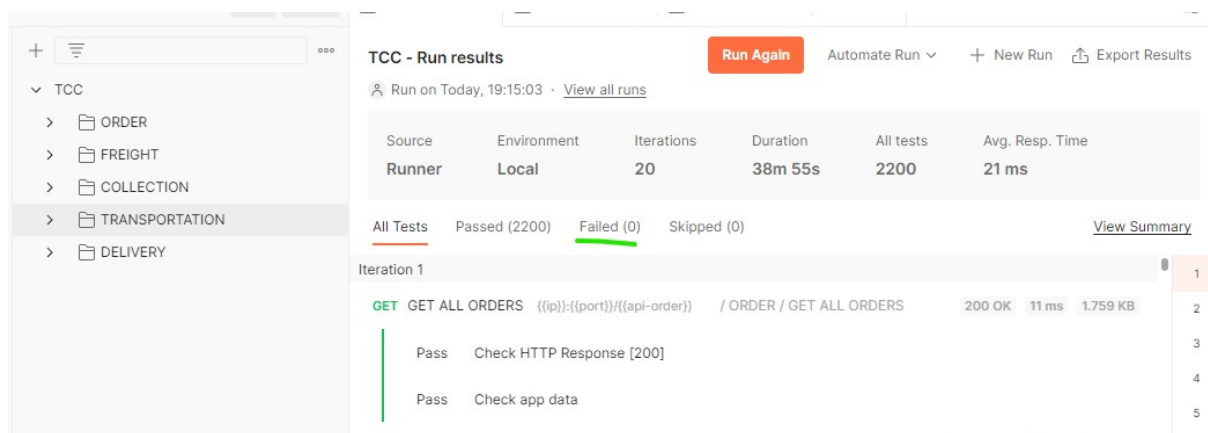


Figura 16: Testes de BDD para toda a API no Postman

A Figura 17 contém o resultado dos testes unitários e a Figura 18 apresenta o resultado dos testes de integração. Todos os testes contemplam os serviços *Order*, *Freight*, *Collection*, *Transportation* e *Delivery*.

<div>Test Results651 ms</div> <div><div>OrderApplicationUnitTests10 ms</div><div>contextLoads()10 ms</div><div>MessageDispatcherTest396 ms</div><div>dispatchByHandled()386 ms</div><div>dispatchErrorEmptyHandlers()9 ms</div><div>dispatchErrorParseFailed()1 ms</div><div>dispatchErrorNotFound()0 ms</div><div>MessageErrorDispatcherTest1 ms</div><div>dispatchErrorWithErrorTypeOnly()1 ms</div><div>dispatchReturnFalse()0 ms</div><div>dispatchErrorWithErrorTypeAndMessage()0 ms</div><div>dispatchFullError()0 ms</div><div>dispatchNoReturn()0 ms</div><div>dispatchReturnNull()0 ms</div><div>dispatchReturnTrue()0 ms</div><div>MessageTest244 ms</div><div>serializeWithCustomData()208 ms</div><div>serializeWithAvroData()36 ms</div></div>	<div>Test Results645 ms</div> <div><div>FreightApplicationUnitTests10 ms</div><div>contextLoads()10 ms</div><div>MessageDispatcherTest387 ms</div><div>dispatchByHandled()374 ms</div><div>dispatchErrorEmptyHandlers()10 ms</div><div>dispatchErrorParseFailed()2 ms</div><div>dispatchErrorNotFound()1 ms</div><div>MessageErrorDispatcherTest4 ms</div><div>dispatchErrorWithErrorTypeOnly()1 ms</div><div>dispatchReturnFalse()1 ms</div><div>dispatchErrorWithErrorTypeAndMessage()0 ms</div><div>dispatchFullError()1 ms</div><div>dispatchNoReturn()0 ms</div><div>dispatchReturnNull()0 ms</div><div>dispatchReturnTrue()1 ms</div><div>MessageTest244 ms</div><div>serializeWithCustomData()211 ms</div><div>serializeWithAvroData()33 ms</div></div>	<div>Test Results669 ms</div> <div><div>CollectionApplicationUnitTests10 ms</div><div>contextLoads()10 ms</div><div>MessageDispatcherTest401 ms</div><div>dispatchByHandled()390 ms</div><div>dispatchErrorEmptyHandlers()9 ms</div><div>dispatchErrorParseFailed()1 ms</div><div>dispatchErrorNotFound()1 ms</div><div>MessageErrorDispatcherTest3 ms</div><div>dispatchErrorWithErrorTypeOnly()1 ms</div><div>dispatchReturnFalse()1 ms</div><div>dispatchErrorWithErrorTypeAndMessage()0 ms</div><div>dispatchFullError()0 ms</div><div>dispatchNoReturn()0 ms</div><div>dispatchReturnNull()1 ms</div><div>dispatchReturnTrue()0 ms</div><div>MessageTest255 ms</div><div>serializeWithCustomData()218 ms</div><div>serializeWithAvroData()37 ms</div></div>
<div>Test Results648 ms</div> <div><div>TransportationApplicationUnitTests10 ms</div><div>contextLoads()10 ms</div><div>MessageDispatcherTest394 ms</div><div>dispatchByHandled()383 ms</div><div>dispatchErrorEmptyHandlers()9 ms</div><div>dispatchErrorParseFailed()2 ms</div><div>dispatchErrorNotFound()0 ms</div><div>MessageErrorDispatcherTest3 ms</div><div>dispatchErrorWithErrorTypeOnly()0 ms</div><div>dispatchReturnFalse()0 ms</div><div>dispatchErrorWithErrorTypeAndMessage()1 ms</div><div>dispatchFullError()0 ms</div><div>dispatchNoReturn()1 ms</div><div>dispatchReturnNull()0 ms</div><div>dispatchReturnTrue()1 ms</div><div>MessageTest241 ms</div><div>serializeWithCustomData()206 ms</div><div>serializeWithAvroData()35 ms</div></div>	<div>Test Results769 ms</div> <div><div>DeliveryApplicationUnitTests10 ms</div><div>contextLoads()10 ms</div><div>MessageDispatcherTest487 ms</div><div>dispatchByHandled()476 ms</div><div>dispatchErrorEmptyHandlers()10 ms</div><div>dispatchErrorParseFailed()1 ms</div><div>dispatchErrorNotFound()0 ms</div><div>MessageErrorDispatcherTest2 ms</div><div>dispatchErrorWithErrorTypeOnly()0 ms</div><div>dispatchReturnFalse()0 ms</div><div>dispatchErrorWithErrorTypeAndMessage()0 ms</div><div>dispatchFullError()1 ms</div><div>dispatchNoReturn()1 ms</div><div>dispatchReturnNull()0 ms</div><div>dispatchReturnTrue()0 ms</div><div>MessageTest270 ms</div><div>serializeWithCustomData()231 ms</div><div>serializeWithAvroData()39 ms</div></div>	

Figura 17: Testes unitários dos serviços: Order, Freight, Collection, Transportation e Delivery

<div><div>Test Results1 sec 523 ms</div><div><div>OrderCommandTest883 ms<ul style="list-style-type: none"><li>checkApprovedDuplication()762 ms</li><li>refused()48 ms</li><li>approved()42 ms</li><li>checkRefusedDuplication()31 ms</li></ul></div><div>OrderApiTests640 ms<ul style="list-style-type: none"><li>getByWrongIdNotFound()181 ms</li><li>getAll(String, boolean, int, boolean)341 ms<ul style="list-style-type: none"><li>[1] page=, first=true, index=0, last=false144 ms</li><li>[2] page=?page=1, first=true, index=0, 69 ms</li><li>[3] page=?page=2, first=false, index=1, 67 ms</li><li>[4] page=?page=10, first=false, index=1, 61 ms</li></ul></li><li>getById()48 ms</li><li>createOrder()70 ms</li></ul></div></div></div>	<div><div>Test Results1 sec 832 ms</div><div><div>FreightCommandTest807 ms<ul style="list-style-type: none"><li>checkFinishDuplication()667 ms</li><li>create()65 ms</li><li>finish()36 ms</li><li>checkCreateDuplication()39 ms</li></ul></div><div>FreightEventTest490 ms<ul style="list-style-type: none"><li>changeStatus(FreightEventStatus, String)469 ms<ul style="list-style-type: none"><li>[1] eventStatus=COLLECTION_STAR43 ms</li><li>[2] eventStatus=COLLECTION_PICKUP27 ms</li><li>[3] eventStatus=COLLECTION_STAR28 ms</li><li>[4] eventStatus=COLLECTION_PICKUP29 ms</li><li>[5] eventStatus=COLLECTION_PICKUP29 ms</li><li>[6] eventStatus=COLLECTION_PACK27 ms</li><li>[7] eventStatus=COLLECTION_PACK24 ms</li><li>[8] eventStatus=IN_TRANSIT_PACK25 ms</li><li>[9] eventStatus=IN_TRANSIT_PACK29 ms</li><li>[10] eventStatus=IN_TRANSIT_PACK23 ms</li><li>[11] eventStatus=IN_TRANSIT_PACK25 ms</li><li>[12] eventStatus=DELIVERY_START51 ms</li><li>[13] eventStatus=DELIVERY_OUT_FCN29 ms</li><li>[14] eventStatus=DELIVERY_FAILED, 28 ms</li><li>[15] eventStatus=DELIVERY_PROCESS25 ms</li><li>[16] eventStatus=DELIVERY_SUCCESS27 ms</li></ul></li><li>deliverySuccessfully()21 ms</li></ul></div><div>FreightApiTests535 ms<ul style="list-style-type: none"><li>getByWrongIdNotFound()152 ms</li><li>getAll(String, boolean, int, boolean)275 ms<ul style="list-style-type: none"><li>[1] page=, first=true, index=0, last=98 ms</li><li>[2] page=?page=1, first=true, index=63 ms</li><li>[3] page=?page=2, first=false, index=62 ms</li><li>[4] page=?page=10, first=false, index=52 ms</li></ul></li><li>getById()54 ms</li><li>checkStatusById()54 ms</li></ul></div></div></div>	<div><div>Test Results1 sec 794 ms</div><div><div>CollectionCommandTest773 ms<ul style="list-style-type: none"><li>commandDuplication()735 ms</li><li>command()38 ms</li></ul></div><div>CollectionApiTests1 sec 21 ms<ul style="list-style-type: none"><li>getByWrongIdNotFound()207 ms</li><li>getAll(String, boolean, int, boolean)343 ms<ul style="list-style-type: none"><li>[1] page=, first=true, index=0, last:133 ms</li><li>[2] page=?page=1, first=true, index:96 ms</li><li>[3] page=?page=2, first=false, index=61 ms</li><li>[4] page=?page=10, first=false, index=53 ms</li></ul></li><li>getById()58 ms</li><li>markAsTaken()116 ms</li><li>markAsReadyToStartDelivery()112 ms</li><li>markAsOutToPickupTheProduct()56 ms</li><li>markAsOnPackaging()85 ms</li><li>checkStatusById()44 ms</li></ul></div></div></div>
<div><div>Test Results1 sec 693 ms</div><div><div>TransferCommandTest710 ms<ul style="list-style-type: none"><li>commandDuplication()676 ms</li><li>command()34 ms</li></ul></div><div>TransferApiTests983 ms<ul style="list-style-type: none"><li>getByWrongIdNotFound()210 ms</li><li>getAll(String, boolean, int, boolean)372 ms<ul style="list-style-type: none"><li>[1] page=, first=true, index=0, last:132 ms</li><li>[2] page=?page=1, first=true, index=104 ms</li><li>[3] page=?page=2, first=false, index=73 ms</li><li>[4] page=?page=10, first=false, index=63 ms</li></ul></li><li>getById()58 ms</li><li>receiveInFinalStorage()125 ms</li><li>receive()101 ms</li><li>transfer()61 ms</li><li>checkStatusById()56 ms</li></ul></div></div></div>	<div><div>Test Results1 sec 873 ms</div><div><div>DeliveryCommandTest752 ms<ul style="list-style-type: none"><li>commandDuplication()712 ms</li><li>command()40 ms</li></ul></div><div>DeliveryApiTests1 sec 121 ms<ul style="list-style-type: none"><li>getByWrongIdNotFound()211 ms</li><li>getAll(String, boolean, int, boolean)369 ms<ul style="list-style-type: none"><li>[1] page=, first=true, index=0, last:146 ms</li><li>[2] page=?page=1, first=true, index:88 ms</li><li>[3] page=?page=2, first=false, index=73 ms</li><li>[4] page=?page=10, first=false, index=62 ms</li></ul></li><li>getById()54 ms</li><li>takePackageToDelivery()107 ms</li><li>markAsDeliveryFailed()94 ms</li><li>addPreferredTimeForDelivery()62 ms</li><li>returnPackage()87 ms</li><li>markAsDeliverySuccessfully()92 ms</li><li>checkStatusById()45 ms</li></ul></div></div></div>	

Figura 18: Testes de integração dos serviços: Order, Freight, Collection, Transportation e Delivery

### 5.3.4 Cenário 4 - Tolerância a falhas

Atributo de Qualidade:	Tolerância a falhas
Requisito de Qualidade:	Ser tolerante a falhas
Preocupação:	
Se um serviço que possui dependentes falha por algum motivo, e este serviço não será mais necessário no fluxo de processamento, os outros serviços devem ser capazes de finalizar o processo sem prejuízos	
Cenário(s):	
Cenário 4	
Ambiente:	
Sistema em operação normal, sem o processo de autenticação	
Estímulo:	
Um cliente, parceiro ou colaborador realiza uma operação qualquer	
Mecanismo:	
Através de um API <i>gateway</i> com o redirecionamento a para o serviço de pedido, frete, coleta, transporte ou entrega.	
Medida de resposta:	
O processo é executado até o fim	
Considerações sobre a arquitetura:	
Riscos:	Durante o fluxo do processo, o serviço falho já deverá ter executado sua tarefa, caso seja antes, todos os processos serão represados
Pontos de Sensibilidade:	Os serviços são implantados no Kubernetes com 2 instâncias cada um
<i>Tradeoff</i> :	Não tem

#### Evidências:

A Figura 19 apresenta a primeira evidência de tolerância a falha, a qual é a redundância na quantidade de serviços ativos. A POC está configurada para ter 2 instâncias de cada serviço. Os serviços estão em imagens Docker executando de um *Pod*. Caso o serviço tenha algum problema e venha a falhar, um outro *Pod* é criado automaticamente. Na imagem é possível observar este comportamento. Repare que o *Pod* `gsl-freight-7959bb844f-gcdvm` foi apagado manualmente e o *Pod* `gsl-freight-7959bb844f-bm7p8` foi imediatamente criado, mantendo as duas instâncias ativas do serviço frete no *cluster* do Kubernetes.

```

proto@docker-vm:~/Dev/puc-tcc-gsl$ kubectl get pods -n gsl
NAME                                READY   STATUS    RESTARTS   AGE
gsl-collection-6457dd56f5-9qmgw     1/1     Running   0           21s
gsl-collection-6457dd56f5-hfcqt     1/1     Running   0           21s
gsl-delivery-665d745747-flblv       1/1     Running   0           19s
gsl-delivery-665d745747-qvq62      1/1     Running   0           19s
gsl-freight-7959bb844f-gcdvm        1/1     Running   0           22s
gsl-freight-7959bb844f-xb88p        1/1     Running   0           22s
gsl-order-5c9d57958c-c6jhf          1/1     Running   0           22s
gsl-order-5c9d57958c-k9bzw          1/1     Running   0           22s
gsl-transportation-55cff86975-s7q5k 1/1     Running   0           20s
gsl-transportation-55cff86975-zqsnq 1/1     Running   0           20s
proto@docker-vm:~/Dev/puc-tcc-gsl$ kubectl delete pod gsl-freight-7959bb844f-gcdvm -n gsl
pod "gsl-freight-7959bb844f-gcdvm" deleted
proto@docker-vm:~/Dev/puc-tcc-gsl$ kubectl get pods -n gsl
NAME                                READY   STATUS    RESTARTS   AGE
gsl-collection-6457dd56f5-9qmgw     1/1     Running   0           77s
gsl-collection-6457dd56f5-hfcqt     1/1     Running   0           77s
gsl-delivery-665d745747-flblv       1/1     Running   0           75s
gsl-delivery-665d745747-qvq62      1/1     Running   0           75s
gsl-freight-7959bb844f-bm7p8        1/1     Running   0           2s   Nova instância!
gsl-freight-7959bb844f-xb88p        1/1     Running   0           78s
gsl-order-5c9d57958c-c6jhf          1/1     Running   0           78s
gsl-order-5c9d57958c-k9bzw          1/1     Running   0           78s
gsl-transportation-55cff86975-s7q5k 1/1     Running   0           76s
gsl-transportation-55cff86975-zqsnq 1/1     Running   0           76s
proto@docker-vm:~/Dev/puc-tcc-gsl$

```

Figura 19: Kubernetes criando um novo POD Freight após uma falha

A segunda evidência é o fato do processo ser linear, onde se inicia na contratação do frete pelo cliente e termina com a encomenda entregue ao destinatário. A Figura 20 contém uma ilustração do padrão *piper-and-filters* (MICROSOFT, P&F), o qual tem como característica um fluxo sequencial, onde o dado é processado diversas vezes na cadeia de filtros. Caso um filtro falhe depois da sua atuação, o mesmo não causará impacto nos processamentos posteriores. A Figura 21 mostra a similaridade do *piper-and-filters* com o fluxo pedido-frete. Lembrando que existe a redundância dos dados através da abordagem dos documentos (*snapshot*). Foi aplicado o padrão Saga coreografado, onde cada nó sabe qual será o próximo nó a ser chamado, ou seja, cada serviço sabe qual o comando deve ser enviado. E a comunicação entre os serviços é realizada via mensagens de eventos e comandos.



Figura 20: Padrão pipe and filter (imagem da internet)

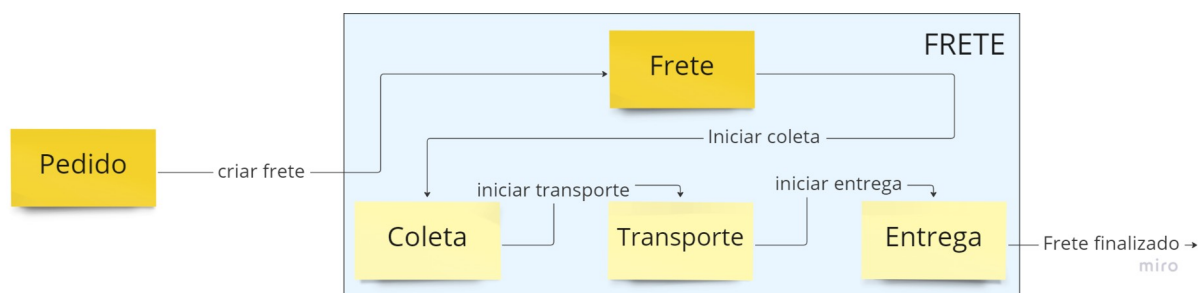


Figura 21: Fluxo do pedido-frete

## 5.4 Resultados

Com base nas evidências produzidas pelo processo da avaliação da proposta arquitetural, considerando a escolha dos 4 cenários relacionados aos requisitos não funcionais, é possível concluir que a solução é válida e atende aos requisitos, com a ressalva para a atributo de desempenho, pois este está diretamente relacionado à capacidade de entrega do hardware, gerando variações lineares do resultado. A tabela a seguir apresenta o resultado para todos os requisitos avaliados.

Requisitos Não Funcionais	Testado	Homologado
RNF05: Apresentar bom desempenho.	SIM	SIM
RNF06: Facilidade de modificação ou adaptação	SIM	SIM
RNF07: O software deve falhar durante a execução dos testes se possuir defeito	SIM	SIM
RNF08: Ser tolerante a falhas	SIM	SIM

Os requisitos não funcionais escolhidos foram atendidos e homologados. Para trabalhos futuros, é interessante avaliar os demais requisitos não funcionais que não foram contemplados nesta avaliação.

A parte de controle de acesso ficou de fora da avaliação, porém a abordagem proposta tem a característica de manter a camada de segurança desacoplada da camada de negócio, gerando um trabalho mais sucinto em ambas as partes.

A mesma filosofia da camada de segurança foi aplicada para separação dos serviços. O principal conceito é: o usuário contrata um frete para levar a mercadoria

até o cliente. Posteriormente foi refinado em: pedido e frete. O conceito de frete foi subdividido em: coleta, transporte e entrega. O sub-conceito coleta foi dividido novamente em: retirada e preparação da encomenda. Essa abordagem recursiva visa gerar nichos menores, especializados e desacoplados, que podem evoluir independente e geram baixa complexidade de manutenção. Quando bem organizados, estes nichos devem ser transparentes para o contexto de mais alto nível. Por exemplo, os processos da coleta são indiferentes para o frete. O mandatório para o frete é realizar coleta, prosseguir para o transporte e depois efetuar a entrega. Com esta abordagem é possível utilizar a melhor tecnologia para resolver um problema específico. É possível ter uma gama de tecnologias heterogêneas atuando em conjunto.

A comunicação por mensagens tem a vantagem de não precisar de um serviço de descoberta, o que facilita distribuir a aplicação para outros ambientes. A desvantagem é o custo de manter uma mensageria. Inicialmente, é mais barato criar um monolito (MICROSERVICES, MONO), com todos os serviços no mesmo binário se comunicando por filas internas. O custo é somente a hospedagem para o *deploy* do binário e a contratação de um banco de dados.

Os pontos de melhoria são a definição de uma *pipeline* para o CI/CD, a parte de log e rastreabilidade, apesar que é possível rastrear as mensagens via *trackId*, que é existente em todas as mensagens. Faltou uma ferramenta de APM, tipo New Relic (NEWRELIC) e também testar o KeyClock na camada de segurança.



## 6 Conclusão

O objetivo da proposta arquitetural foi adequar a *stack* tecnológica atual da Boa Entrega para suportar o aumento da demanda, a integração com outras tecnologias e a geração de dados de operação que devem alimentar os sistemas de tomada de decisões. A POC trouxe vislumbre de como, na prática, isso pode funcionar e possibilitou validar se o que está sendo entregue realmente atende a demanda da Boa Entrega.

É necessário ter ciência que existem alguns fatores conhecidos que não foram avaliados e, certamente, aparecerão novas questões no decorrer do processo. Não foi realizada nenhuma menção sobre as restrições jurídicas, específicas no nicho do negócio, que poderiam impactar no desenvolvimento da solução. Outra questão é que o valor do *budget* disponível para o projeto é desconhecido, lembrando que os valores das ferramentas e serviços de nuvem geralmente são cobrados em dólar. Se for o caso, é possível, inicialmente, gerar uma versão da solução mais barata, um monólito, substituindo os recursos de mensageria do Kafka por alternativas embutidas na linguagem de programação, justamente para possibilitar implantar e entrar em operação o mais rápido e barato possível. A medida que a demanda aumente, será necessário retornar com o projeto original, em microsserviços.

A proposta arquitetural apresentada é escalável, distribuída, fácil de modificar e acompanhar o crescimento do negócio. Tem interoperabilidade com outras tecnologias como *mobile*, Web e entre outras. Ela possui a característica de serviços autônomos, tolerantes à falha, que são simples de manter e evoluir. Resumindo, resolve os problemas do negócio descritos pelo cliente e deixa uma série de decisões em aberto, para o futuro, com o objetivo de comportar toda a evolução do negócio que ocorrerá com o avanço tecnológico do mundo.

## Referências

Evans, Eric. **Domain-Driven Design: Atacando as complexidades no coração do software**. 3ª edição. Rio de Janeiro/Brasil: Alta Books, 2016

Microsoft. Power BI. **Microsoft**, -. Disponível em: <https://powerbi.microsoft.com/pt-br/>. Acesso em: 30/11/2022

IBM. IBM Business Process Manager. **IBM**, -. Disponível em: <https://www.ibm.com/docs/pt-br/bpm/8.5.7?topic=manager-business-process-overview>. Acesso em: 30/11/2022

Totvs. Totvs ERP. **Totvs**, -. Disponível em: <https://www.totvs.com/>. Acesso em: 30/11/2022

KeyCloak. KeyCloak. **KeyCloak.org**, -. Disponível em: <https://www.keycloak.org/>. Acesso em: 30/11/2022

Brandolini, Alberto. Event Storming. **Event Storming**, -. Disponível em: <https://www.eventstorming.com/>. Acesso em: 30/11/2022

Gertel, Lucas. EVENT STORMING - DOMAIN DRIVEN DESIGN, EVENT SOURCING E CQRS!. **YouTube**, 2020. Disponível em: <https://www.youtube.com/watch?v=s8cyn2TUXoM>. Acesso em: 30/11/2022

Gertel, Lucas. DOMINE O EVENT STORMING!. **YouTube**, 2022. Disponível em: <https://www.youtube.com/watch?v=Y31kprvAfIE>. Acesso em: 30/11/2022

Git. Git. **Git**, -. Disponível em: <https://git-scm.com/>. Acesso em: 30/11/2022

Hibernate. Hibernate. **Hibernate.org**, -. Disponível em: <https://hibernate.org/>. Acesso em: 30/11/2022

Angular. Angular. **Angular**, -. Disponível em: <https://angular.io/>. Acesso em: 30/11/2022

React. React. **React.org**, -. Disponível em: <https://pt-br.reactjs.org/>. Acesso em: 30/11/2022

Vue. Vue. **Vue.org**, -. Disponível em: <https://vuejs.org/>. Acesso em: 30/11/2022

Microservices. Pattern: Microservice Architecture. **Microservices.io**, -. Disponível em: <https://microservices.io/patterns/microservices.html>. Acesso em: 30/11/2022

Spring. Spring Boot. **Spring**, -. Disponível em: <https://spring.io/projects/spring-boot>. Acesso em: 30/11/2022

Java. Java. **Java**, -. Disponível em: <https://www.java.com/pt-BR/>. Acesso em: 30/11/2022

Kotlin. Kotlin. **Kotlinlang.org**, -. Disponível em: <https://kotlinlang.org/>. Acesso em: 30/11/2022

Apache. Kafka. **Apache.org**, -. Disponível em: <https://kafka.apache.org/>. Acesso em: 30/11/2022

Apache. Avro. **Apache.org**, -. Disponível em: <https://avro.apache.org/>. Acesso em: 30/11/2022

Spring. springdoc-openapi. **Spring**, -. Disponível em: <https://springdoc.org/>. Acesso em: 30/11/2022

Microservices. Pattern: API Gateway / Backends for Frontends. **Microservices.io**, -. Disponível em: <https://microservices.io/patterns/apigateway.html>. Acesso em: 30/11/2022

Microservices. Pattern: API Composition. **Microservices.io**, -. Disponível em: <https://microservices.io/patterns/data/api-composition.html>. Acesso em: 30/11/2022

Kong. Kong. **Konghq**, -. Disponível em: <https://konghq.com/community>. Acesso em: 30/11/2022

Junit. Junit. **Junit.org**, -. Disponível em: <https://junit.org/>. Acesso em: 30/11/2022

Spring. Spring Boot Test. **Spring**, -. Disponível em: <https://docs.spring.io/spring-boot/docs/1.5.2.RELEASE/reference/html/boot-features-testing.html>. Acesso em: 30/11/2022

Postman. Postman. **Postman**, -. Disponível em: <https://www.postman.com/>. Acesso em: 30/11/2022

PostgreSQL. PostgreSQL. **PostgreSQL.org**, -. Disponível em: <https://www.postgresql.org/>. Acesso em: 30/11/2022

Github. Github. **Github**, -. Disponível em: <https://github.com/>. Acesso em: 30/11/2022

Gitlab. Gitlab. **Gitlab**, -. Disponível em: <https://gitlab.com/>. Acesso em: 30/11/2022

Docker. Docker Registry. **Docker**, -. Disponível em: <https://docs.docker.com/registry/>. Acesso em: 30/11/2022

Kubernetes. Kubernetes. **Kubernetes**, -. Disponível em: <https://kubernetes.io/pt-br/>. Acesso em: 30/11/2022

Microservices. Pattern: Database per service. **Microservices.io**, -. Disponível em: <https://microservices.io/patterns/data/database-per-service.html>. Acesso em: 30/11/2022

Docker. Docker. **Docker**, -. Disponível em: <https://www.docker.com/>. Acesso em: 30/11/2022

Microservices. Pattern: Anti-corruption layer. **Microservices.io**, -. Disponível em: <https://microservices.io/patterns/refactoring/anti-corruption-layer.html>. Acesso em: 30/11/2022

Gradle. Gradle. **Gradle.org**, -. Disponível em: <https://gradle.org/>. Acesso em: 30/11/2022

Provectus . Kafka UI. **Github**, -. Disponível em: <https://github.com/provectus/kafka-ui>. Acesso em: 30/11/2022

H2 Database. H2. **H2 Database**, -. Disponível em: <https://www.h2database.com/>. Acesso em: 30/11/2022

Swagger. Swagger-UI. **Swagger**, -. Disponível em: <https://swagger.io/>. Acesso em: 30/11/2022

Pantsel. Konga. **Github**, -. Disponível em: <https://pantsel.github.io/konga/>. Acesso em: 30/11/2022

SonarQube. SonarQube. **SonarQube.org**, -. Disponível em: <https://www.sonarqube.org/%20>. Acesso em: 30/11/2022

Apache. JMeter. **Apache.org**, -. Disponível em: <https://jmeter.apache.org/>. Acesso em: 30/11/2022

JR, Elemar. Descomplicando "Event Sourcing". **YouTube**, 2020. Disponível em: <https://www.youtube.com/watch?v=f4GolliNlvc>. Acesso em: 29/11/2022

Microservices. Pattern: Event sourcing. **Microservices.io**, -. Disponível em: <https://microservices.io/patterns/data/event-sourcing.html>. Acesso em: 30/11/2022

JR, Elemar. Descomplicando "Sagas". **YouTube**, 2020. Disponível em: <https://www.youtube.com/watch?v=jMBfO52FttY>. Acesso em: 29/11/2022

Microservices. Pattern: Saga. **Microservices.io**, -. Disponível em: <https://microservices.io/patterns/data/saga.html>. Acesso em: 30/11/2022

Campos, Camila. Event Driven Architecture na Credits. **YouTube**, 2019. Disponível em: <https://www.youtube.com/watch?v=pRG0Phb8ZJM>. Acesso em: 30/11/2022

Microservices. Pattern: Transactional outbox. **Microservices.io**, -. Disponível em: <https://microservices.io/patterns/data/transactional-outbox.html>. Acesso em: 30/11/2022

DevOps, Azure . Padrão de pipes e filtros. **Microsoft**, -. Disponível em: <https://learn.microsoft.com/pt-br/azure/architecture/patterns/pipes-and-filters>. Acesso em: 30/11/2022

Microservices. Pattern: Monolithic Architecture. **Microservices.io**, -. Disponível em: <https://microservices.io/patterns/monolithic.html>. Acesso em: 30/11/2022

New Relic. New Relic. **New Relic**, -. Disponível em: <https://newrelic.com/>. Acesso em: 30/11/2022

## APÊNDICES

- Código fonte da POC: <https://github.com/println/puc-tcc-gsl>;
- Apresentação cenário 1 - Desempenho:
  - [https://drive.google.com/file/d/1iAyH8E8KhluQoNz\\_EnyCP2mbi-fh0-kG/view?usp=sharing](https://drive.google.com/file/d/1iAyH8E8KhluQoNz_EnyCP2mbi-fh0-kG/view?usp=sharing)
- Apresentação cenário 2 - Manutenibilidade:
  - <https://drive.google.com/file/d/1snEA8CDAozyKJldkCz52Mid5DHE3Y17K/view?usp=sharing>
- Apresentação cenário 3 - Testabilidade:
  - [https://drive.google.com/file/d/1YtuhUaq8U75TRdYqaXvL1yr6\\_qBO2dn3/view?usp=share\\_link](https://drive.google.com/file/d/1YtuhUaq8U75TRdYqaXvL1yr6_qBO2dn3/view?usp=share_link)
- Apresentação cenário 4 – Tolerância a falhas:
  - [https://drive.google.com/file/d/1WLMXLyUx\\_QeE9AcsTL-ynblHaNg\\_bcSB/view?usp=share\\_link](https://drive.google.com/file/d/1WLMXLyUx_QeE9AcsTL-ynblHaNg_bcSB/view?usp=share_link)
- POC monolito implantado: <https://boa-entrega.herokuapp.com/>