PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

PUC Minas Virtual

Pós-graduação *Lato Sensu* em Arquitetura de *Software* Distribuído

Projeto Integrado

Relatório Técnico

Gestão de Serviços de Logística (GSL)

Gilliard Jefferson da Costa

Belo Horizonte
Junho de 2023.

Projeto Integrado – Arquitetura de Software Distribuído

Sumário

1. Introdução

A Boa Entrega é uma empresa de grande porte no ramo de transportes e possui muitos clientes em Municípios que tem sua atuação. A Boa Entrega possui tecnologia, parceiros e experiencia em entregas ágeis e de baixo custo para seus clientes, acompanhada e avaliada por seu setor financeiro. Em seu próximo triênio, a empresa tem como objetivo melhorar suas entregas, aumentar sua área de atuação territorial e de suas parcerias, conseguir mais clientes, preferencialmente no ramo de supermercadista, e por fim aumentar dez por centro seu faturamento global. Com esses objetivos, a Boa Entrega fez um levantamento e identificou pontos para melhoria, tais como automatizar todo o processo de entrega, oferecer integrações dos seus sistemas com suas parcerias, utilizar geotecnologias em todos os processos que envolvam localização e utilizar toda tecnologia para atender os objetivos da empresa.

A empresa fez um levantamento interno de seus próprios sistemas, e nestes detectou ineficiência e limitação de determinadas funcionalidades. Os sistemas atuais da empresa são: Sistema Administrativo-Financeiro (SAF) baseado em solução desktop da Microsoft .Net Core e com banco de dados em MS SQL Server, é um sistema legado que atende a área administrativa e financeira, mas não atende a todas as necessidades do negócio atualmente devido a limitação de integração com outros sistemas; Sistema de Gestão de Entregas (SGE) é um sistema web, com front-end em JavaScript e HTML com algumas funcionalidades em Flutter e back-end com PHP, tem uma estrutura mais organizada com módulos independentes e ele fornece a gestão completa de logística; Sistema de Faturamento e Cobrança (SFC) é um sistema desenvolvido pela própria equipe de TI da empresa que realiza todo o processo de faturamento e cobrança; e Portal Corporativo é um sistema web e mobile, componetizado em JavaScript, HTML e CSS, com autenticação, onde colaboradores, parcerias e clientes tem acesso a diversas informações.

Para atingir estes objetivos, a Boa Entrega pretende desenvolver um novo projeto baseado em arquitetura de microsserviços, chamada Gestão de Serviços de Logística (GSL), no qual foram planejados quatro módulos novos, que poderá poderão incorporar e integrar aos sistemas existentes. Os módulos previstos são: Módulo de Informações Cadastrais, pertinentes as informações de seus clientes, fornecedores, depósitos e mercadorias, e também são mantidas essas informações pelos sistemas legados da empresa; Módulo de Serviços ao Cliente será baseado em uma solução de workflow, com o intuito de ter toda a observabilidade dos processos da empresa; Módulo de Gestão e Estratégia terá como finalidade gerir toda a atividade estratégica da empresa, com indicadores e benchmarking das entregas; Módulo de Ciência de Dados (Data Warehouse (DW) e Business Intelligence (BI)) onde serão utilizadas diversas ferramentas de dados para tratamento dos mesmos, utilizando bancos relacionais e não relacionais. Esses módulos garantem que os objetivos desejados pela empresa sejam alcançados. Com a implantação de uma arquitetura distribuída de microsserviços, a manutenção e escalabilidade? da empresa se torna mais viável, utilizando novas tecnologias como serverless ou gerenciamento e orquestração de aplicativos em contêineres, poderemos utilizar APIs REST e gRPC, workers para processamentos longos e pesados, mensageria para garantir a comunicação entre os microsserviços de forma assíncrona, utilização de bancos de dados não relacionais para maior flexibilidade e garantia no desempenho. Outra grande vantagem de utilização de microsserviços será a facilitação da integração e comunicação com outros sistemas, seguindo o objetivo de substituição do sistema legado, poderemos planejar a sua migração, baseando-se no padrão de projeto Strangler Fig, gradativamente substituindo o legado.

O objetivo deste trabalho é apresentar o projeto de uma aplicação baseada em arquitetura distribuída de microsserviços, contemplando todo o ecossistema, definindo os recursos tecnológicos, garantindo o pleno funcionamento, permitindo flexibilidade com seu crescimento, ajustando os melhores investimentos para evitar gastos desnecessários.

Os objetivos específicos propostos são:

- Examinar minuciosamente os requisitos, integrando-os com as expectativas e estabelecer um planejamento estratégico preciso;
- Descrição dos requisitos funcionais e não funcionais;
- Realizar o desenho e detalhamento da nova arquitetura, bem como analisar como seus diversos artefatos se integram harmoniosamente para formar um sistema coeso e funcional;
- Providenciar POCs das soluções propostas.

2. Cronograma do Trabalho

A seguir é apresentado o cronograma proposto para as etapas deste trabalho.

Datas		**Atividade / Tarefa**	**Produto / Resultado**
De	**Até**		
01 /06 / 23	02 /06 /23	1. Leitura e conhecimento das necessidades e requisitos.	Documento de Requisitos.
03 / 06 / 23	06 /06 /23	2. Elaboração da solução.	Proposta do projeto.
07 /06 /23	07 /06 / 23	3. Reunião de alinhamento.	Termo de aceite.
08 / 06 / 23	09 / 06 / 23	4. Reunião técnica.	Termo de aceite.
10 / 06 / 23	14 / 06 / 23	5. Desenhar a solução.	Diagramas.
15 / 06 / 23	17 / 06 / 23	6. Preparar ambiente.	Publicação/Infra.
18 / 06 / 23	24 / 06 / 23	7. Desenvolver os protótipos de API, Worker e Infra.	POC.
25 / 06 / 23	27 / 06 / 23	8. Escrever o documento. Preparar homologação.	Relatório Técnico ASD.
28 / 06 / 23	29 / 06 / 23	9. Preparar vídeos e apresentação.	Vídeos e Apresentação.
30 / 06 / 23	30 / 06 / 23	10. Envio dos materiais.	Entrega dos documentos.

3. Especificação Arquitetural da solução

A arquitetura apresentada será arquitetura distribuída de microsserviços, onde o ecossistema consiste em microsserviços independentes que se comunicam através de APIs e mensagens. Eles são implantados em infraestruturas em nuvem e trabalham em conjunto para fornecer funcionalidades específicas, garantindo escalabilidade, flexibilidade e modularidade ao sistema.

3.1 Restrições Arquiteturais

O GSL precisa seguir as restrições da área de TI da empresa. A arquitetura proposta deve garantir requisitos essenciais, como baixo custo, facilidade de uso para colaboradores e associados, acesso em tecnologias web e mobile, e manutenção dos componentes sem gerar altos custos futuros. No entanto, trade-offs podem dificultar a definição da solução, levando a escolhas parciais para atender aos requisitos desejados. Os padrões tecnológicos devem permitir a integração com sistemas legados, com baixo acoplamento e sem substituir os ativos existentes.

R1: Possuir características de aplicação distribuída: abertura, portabilidade, uso de recursos de rede.
R2: Atender, de forma seletiva (por perfil) a clientes, fornecedores e colaboradores.
R3: Ser modular e componetizado, utilizando orientação a serviços.
R4: Ser de fácil implantação e utilização.
R5: Ser hospedado em nuvem híbrida, sendo a forma de hospedagem documentada.
R6: Suportar ambientes web e móveis.
R7: Possuir interface responsiva.
R8: Apresentar bom desempenho.
R9: Apresentar boa manutenibilidade.
R10: Ser testável em todas as suas funcionalidades.
R11: Ser recuperável (resiliente) no caso da ocorrência de erro.
R12: Utilizar APIs ou outros recursos adequados para consumo de serviços.
R13: Estar disponível em horário integral (24 H), sete dias por semana.
R14: Ser desenvolvido utilizando recursos de gestão de configuração, com integração contínua.

3.2 Requisitos Funcionais

Segue todos os requisitos funcionais previstos para a esta aplicação.

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF01	Automatizar os processos de entrega para aprimorar a apuração, conferência, faturamento e remuneração.	M	A
RF02	Implementar integrações com sistemas de parceiras para permitir entregas em parceria em diferentes etapas do processo.	A	A
RF03	Adaptar os sistemas atuais e incorporar novos componentes para alcançar maior abertura, utilizando uma arquitetura orientada a serviços.	A	M
RF04	Utilizar geotecnologias para facilitar a identificação e atualização de informações de entregas agendadas e realizadas.	M	M
RF05	Viabilizar o uso de todas as tecnologias da informação e softwares necessários para atender às demandas dos clientes, fornecedores e parceiros, conforme definido no documento.	B	B
RF06	Possibilitar a rastreabilidade completa dos processos de entrega, desde o agendamento até a conclusão.	M	M
RF07	Integrar os sistemas internos da Boa Entrega com os sistemas dos clientes, fornecedores e parceiros, permitindo troca de informações em tempo real.	A	B
RF08	Permitir o agendamento de entregas de forma online, com opções de escolha de data, horário e local.	B	M
RF09	Gerar relatórios e métricas sobre o desempenho e eficiência dos processos de entrega, incluindo indicadores de qualidade, prazos e custos.	B	B
RF10	Disponibilizar um portal de autoatendimento para clientes, onde eles possam acompanhar o status das entregas, solicitar alterações e obter suporte.	M	M
RF11	Facilitar a comunicação e colaboração entre os diversos envolvidos no processo de entrega, por meio de notificações, mensagens e recursos de chat.	B	B
RF12	Garantir a segurança e proteção dos dados sensíveis envolvidos nos processos de entrega, em conformidade com as regulamentações e políticas de privacidade aplicáveis.	A	B

*B=Baixa, M=Média, A=Alta.

3.3 Requisitos Não-funcionais

Abaixo, são apresentados os requisitos não funcionais identificados para a implantação da nova arquitetura distribuída de microsserviços na empresa Boa Entrega.

ID	Descrição	Prioridade B/M/A
RNF01	Realizar todas as entregas com tempo médio inferior a 5 dias úteis.	A
RNF02	Expandir a atuação para mais 200 municípios de pequeno/médio porte.	A
RNF03	Atuar na região norte do Brasil, último reduto que a empresa ainda não cobre, estabelecendo parcerias com uma empresa aérea e outras empresas de logística terrestres locais.	A
RNF04	Desenvolver novas parcerias com outras transportadoras para complementar a atuação em lugares onde a empresa apresenta market share inferior a 10%.	M
RNF05	Fazer convênio com no mínimo 50 novos clientes, preferencialmente do ramo supermercadista.	M
RNF06	Crescer 10% em termos de faturamento global.	B
RNF07	Garantir a rastreabilidade de 100% das operações realizadas.	B
RNF08	Assegurar uma prestação de contas adequada 100% e uma gestão financeira transparente.	B
RNF09	Gerenciar os estoques de forma 100% eficiente	B

3.4 Mecanismos Arquiteturais

Abaixo, apresentamos uma visão geral dos mecanismos que compõem a arquitetura do software, abrangendo os estados de análise, design e implementação. Na fase de análise, são levantados os requisitos e definidas as necessidades do sistema. No design, são estabelecidos os padrões. Por fim, na implementação, o design é transformado em produto. Considerações como manutenibilidade, extensibilidade e controle de versão são essenciais ao longo do processo.

Análise	*Design*	Implementação
Persistência	ORM	Entity Framework Core
Front-end	Single Page Application	Vue.js
Back end	API Rest	NET 6
Back end	Worker	NET 6

Repositório	Git	Github
Log do sistema	Collector/Tracing	Jaeger
Teste Unitário	TDD	NUnit
Teste de Cargas	Carga Constante	K6
Deploy	Container	Kubernetes
Mensageria	Message Broker	ActiveMQ Artemis
Integração	API Gateway	Kong
Componente mensageria	Message Broker Middleware	MassTransit
Componente para log e tracing	Collector	OpenTelemetry
Software Geolocalização	GPS	Google Maps Platform

4. Modelagem Arquitetural

Para esta modelagem arquitetural optou-se por utilizar o modelo C4 para documentação de arquitetura de software e os desenhos serão feitos por código. Dos quatro níveis que compõem o modelo C4, três serão apresentados aqui, e somente o Código será apresentado na próxima seção (5).

4.1 Diagrama de Contexto

Nesta seção apresentaremos o diagrama de contexto referente a proposta da nossa arquitetura. É uma visão macro de quais serão os sistemas, suas integrações com outros sistemas, seus tipos de usuários.

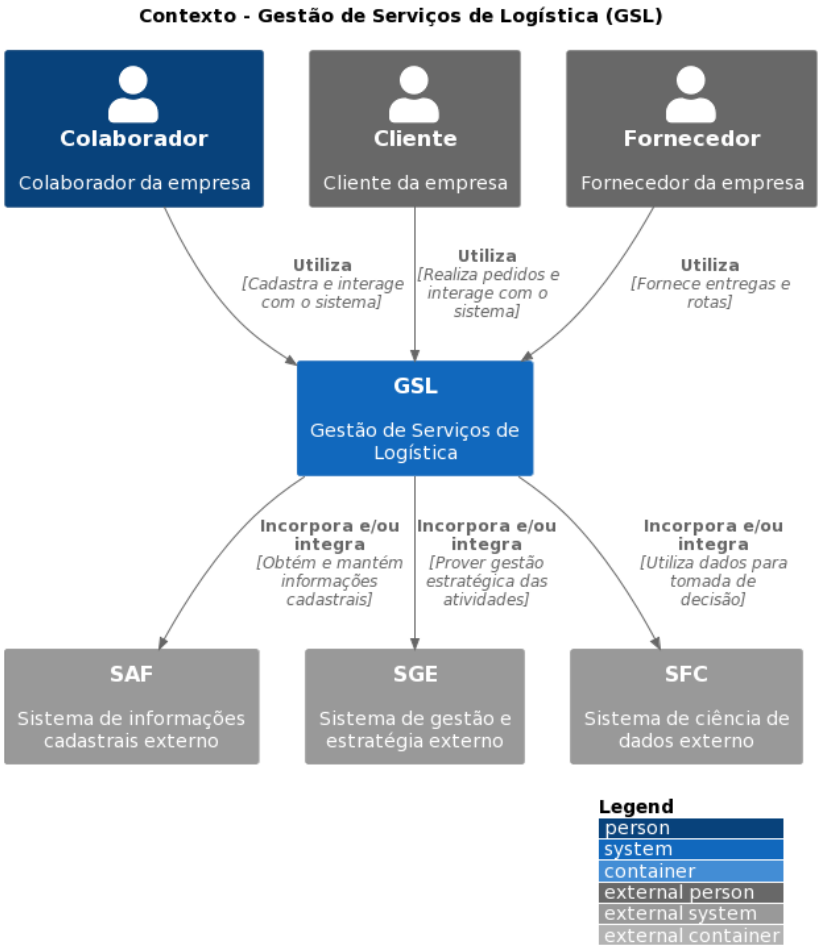

Figura 1 - Visão Geral da Solução.

A figura 1 mostra a especificação do diagrama geral da solução proposta, com todos seus principais módulos e suas interfaces. As imagens com qualidade melhor e o código delas, estarão disponíveis no repositório.

4.2 Diagrama de Container

Nesta seção apresentamos o diagrama de container, o qual contempla todos os componentes, seus tipos e tecnologias.

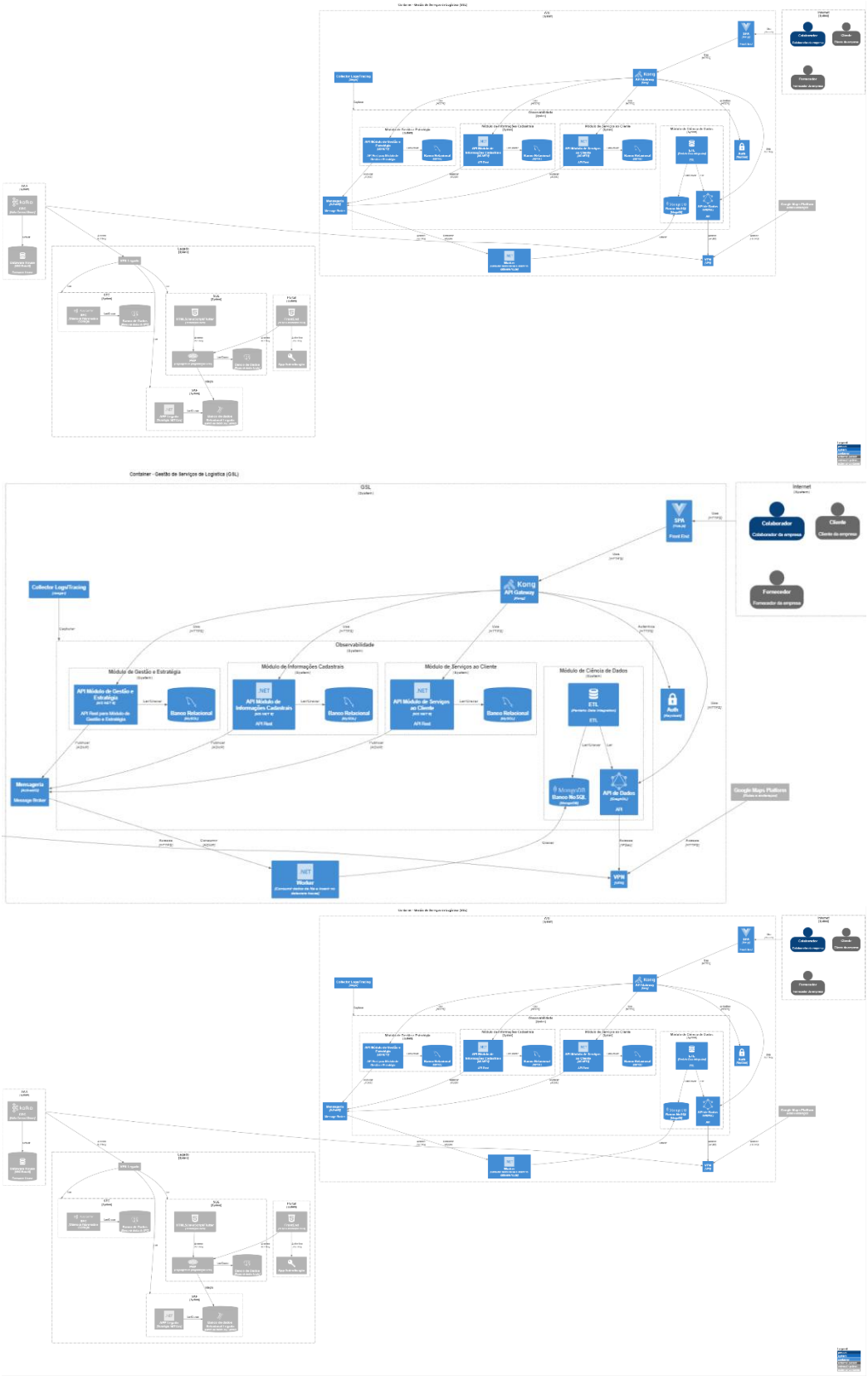

Figura 2 – Diagramas de container.

A figura 2 apresenta os *containers* da aplicação do novo sistema GSL.

4.3 Diagrama de Componentes

O sistema é composto por vários microsserviços independentes que se comunicam entre si por meio de APIs REST. Essa arquitetura permite que os microsserviços sejam desenvolvidos, implantados e dimensionados de forma independente, facilitando a escalabilidade e a manutenção do sistema.

A seguir, encontra-se uma descrição suscinta dos componentes e seus respectivos papéis dentro da arquitetura:

A1. SPA Vue.js (Single-Page Application): É a interface de usuário do sistema, desenvolvida usando o framework Vue.JS. Comunica-se com o backend por meio de APIs REST e exibe os dados aos usuários.

A2. API Gateway Kong: É responsável por receber as solicitações dos clientes e encaminhá-las para os microsserviços adequados. Atua como um ponto de entrada único para o sistema, simplificando o roteamento e a autenticação.

A3. API Auth Keycloak: É responsável pela autenticação e autorização dos usuários. Fornece recursos de gerenciamento de identidade, como login, registro e controle de acesso.

A4. Entity Framework Core: É um framework de mapeamento objeto-relacional utilizado para acessar o banco de dados MySQL. Fornece uma interface de programação para consulta e manipulação de dados.

A5. MySQL: É um Sistema de Gerenciamento de Banco de Dados Relacional (SGBDR) utilizado para armazenar os dados persistentes do sistema.

A6. API Rest dotNet: É um microsserviço desenvolvido em .NET que expõe uma API REST para interagir com o sistema. Comunica-se com o Entity Framework Core para acessar o banco de dados MySQL.

A7. ActiveMQ Artemis: É um sistema de mensageria utilizado para comunicação assíncrona entre os microsserviços. Fornece uma fila de mensagens para troca de informações entre os componentes.

A8. Worker dotNet: É um microsserviço que consome mensagens da fila do ActiveMQ Artemis e executa tarefas assíncronas. Pode ser usado para processamento de longa duração ou para a execução de trabalhos em segundo plano.

A9. MongoDB: É um banco de dados NoSQL utilizado para armazenar dados não relacionais. É acessado pelo Worker dotNet por meio do driver MongoDB. Em relação aos componentes, podemos identificar os seguintes:

Reutilizados: Navegadores web (utilizados para acessar a interface SPA Vue.js) e SGBD MySQL (usado pelo Entity Framework Core).

Adquiridos por serem proprietários: API Gateway Kong, API Auth Keycloak, ActiveMQ Artemis e MongoDB. Esses componentes são fornecidos por terceiros e são adquiridos para uso no sistema.

Precisam ser desenvolvidos: SPA Vue.JS, Entity Framework Core, API REST dotNet e Worker dotNet. Esses componentes são desenvolvidos internamente para atender às necessidades específicas do sistema.

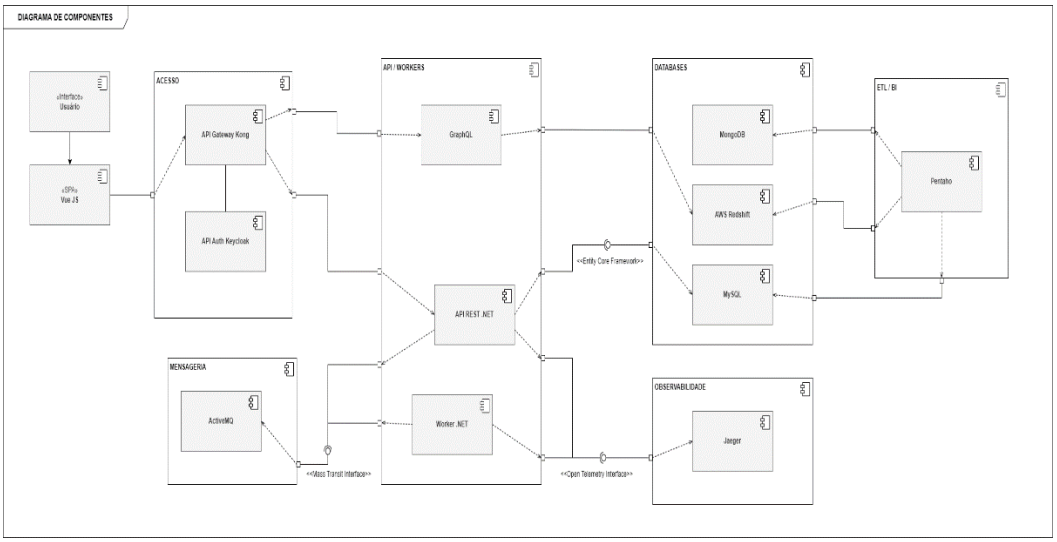

Figura 3 – Diagrama de Componentes.

5. Prova de Conceito (PoC)

Para validar e homologar a solução proposta, desenvolvemos uma infra similar a sugerida, sendo criada por container através de um arquivo yaml baseado em IaC (Infraestrutura como código), onde é possível subir de forma ágil um ambiente. A segunda validação foi referente ao framework de desenvolvimento Microsoft Net 6, onde foi possível criar microsserviços como API REST e Workers, com a facilidade dos componentes disponibilizados para a comunidade através do gerenciador de pacotes Nuget, onde foi comprovado com os componentes Entity Framework, MassTransit e OpenTelemetry, na qual também foi utilizado em outra prova de conceito. E por fim, validamos a coleta de logs e tracing, auxiliando na observabilidade de todo o ecossistema.

5.1 Integrações entre Componentes

Na prova de conceito, validamos o uso de uma API REST, onde outros aplicativos (SPA, APIs, etc) podem se comunicar e usufruir de suas funcionalidades. Outro conceito utilizado foi o de mensageria, o qual pode ser utilizado para distribuição de cargas, workflow, processamento assíncrono e outras utilidades.

5.2 Código da Aplicação

Abaixo, segue o modelo C4 de diagrama de código.

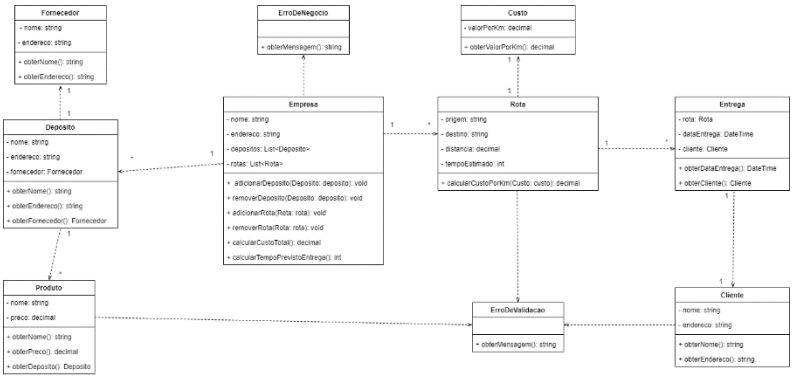

Figura 4 – Estrutura de código da aplicação.

A estrutura da aplicação apresentada na Figura consiste em uma representação de classes e relacionamentos que descrevem um software de gerenciamento logístico para uma empresa. Os componentes de código e suas funções são definidos da seguinte forma:

A classe "Empresa" representa uma empresa e possui atributos como nome, endereço, uma lista de depósitos e uma lista de rotas. Essa classe é responsável por adicionar e remover depósitos e rotas, além de calcular o custo total e o tempo previsto de entrega.

A classe "Rota" descreve uma rota específica, com atributos como origem, destino, distância e tempo estimado. Essa classe possui um método para calcular o custo por quilômetro, utilizando a classe "Custo".

A classe "Custo" representa o valor do custo por quilômetro de uma rota. Ela possui um único atributo, "valorPorKm", e um método para obter esse valor.

A classe "Cliente" descreve um cliente, com atributos como nome e endereço. Ela possui métodos para obter o nome e o endereço do cliente.

A classe "Entrega" representa uma entrega específica, com atributos como a rota a ser seguida, a data de entrega e o cliente associado. Ela possui métodos para obter a data de entrega e o cliente.

A classe "Deposito" descreve um depósito, com atributos como nome, endereço e fornecedor associado. Ela possui métodos para obter o nome, o endereço e o fornecedor do depósito.

A classe "Fornecedor" representa um fornecedor, com atributos como nome e endereço. Ela possui métodos para obter o nome e o endereço do fornecedor.

A classe "Produto" descreve um produto, com atributos como nome, preço e depósito associado. Ela possui métodos para obter o nome, o preço e o depósito do produto.

As classes "ErroDeValidacao" e "ErroDeNegocio" são classes de exceção que podem ser lançadas no código para tratar erros de validação ou erros de negócio, respectivamente.

Os relacionamentos entre as classes são estabelecidos por meio de associações. Por exemplo, a classe "Empresa" possui uma associação de um para muitos com a classe "Deposito", indicando que uma empresa pode ter vários depósitos. Da mesma forma, a classe "Rota" tem uma associação de um para um com a classe "Custo", indicando que uma rota está associada a um único custo.

Essa estrutura de classes e relacionamentos forma a base do software implementado, permitindo o gerenciamento de depósitos, rotas, entregas, clientes, produtos e fornecedores, e fornecendo funcionalidades como cálculo de custo e tempo de entrega.

Os vídeos, as POCs e outros documentos, estarão disponíveis no seguinte repositório: https://github.com/gilliardj/pucminas. Para acesso ao sistema será disponibilizado no endereço http://gilliard.codigolimpo.com.br/, com usuário pucminas e senha pucminas.

6. Avaliação da Arquitetura (ATAM)

A avaliação da arquitetura desenvolvida neste trabalho é abordada nesta seção visando avaliar se ela atende ao que foi solicitado pelo cliente, segundo o método Architecture Tradeoff Analysis Method (ATAM).

6.1. Análise das abordagens arquiteturais

Segue um breve resumo das principais características da proposta arquitetural.

Atributos de Qualidade	Cenários	Importância	Complexidade
Interoperabilidade	Cenário 1: A arquitetura distribuída visa uma grande comunicação entre todos os microsserviços, com o intuito de prover mais funcionalidade e maior escalonamento.	A	M
Usabilidade	Cenário 2: A usabilidade do sistema foi desenvolvida para que seja de fácil entendimento e absorção, para reduzir treinamentos e tempo de preparo para seus usuários.	M	B
Manutenibilidade	Cenário 3: O ecossistema foi arquitetado para prover uma facilidade em sua manutenção, considerando o isolamento de cada contexto, onde se uma ponta falhar, não afeta ao todo.	A	A
Eficiência	Cenário 4: O sistema não pode sofrer paralizações, com essa preocupação aplicamos a observabilidade em todo o ecossistema.	M	M
Confiabilidade	Cenário 5: O sistema deve garantir o seu uso contínuo, com isso implementamos boas práticas de segurança e de recuperação de desastre, além de realizar testes	A	M

	em todo ciclo de vida do sistema.		
Portabilidade	Cenário 6: Toda a infra será criada seguindo o padrão de código, dessa forma poderemos migrar, aumentar ou reduzir os recursos de forma ágil.	B	B

6.2. Cenários

Abaixo apresentaremos os cenários e seus testes.

Cenário 1 - Interoperabilidade:

Evidência: Registro de trocas de mensagens entre os microsserviços, demonstrando a comunicação efetiva e a capacidade de interoperabilidade.

Teste: Realizar testes de integração que validem a comunicação entre os microsserviços, garantindo que as mensagens sejam transmitidas corretamente e as funcionalidades estejam bem integradas.

Cenário 2 - Usabilidade:

Evidência: Coletar feedback dos usuários sobre a facilidade de uso do sistema, avaliando a eficiência em realizar tarefas e a satisfação geral dos usuários.

Teste: Realizar testes de usabilidade com usuários representativos, fornecendo tarefas para serem realizadas no sistema e observando o tempo necessário e a facilidade em completar cada tarefa.

Cenário 3 - Manutenibilidade:

Evidência: Documentação atualizada dos microsserviços, incluindo detalhes sobre sua arquitetura, dependências e procedimentos de manutenção.

Teste: Realizar testes de manutenção, como atualização de versões de microsserviços individuais e verificar se o sistema como um todo continua funcionando corretamente e sem interrupções.

Cenário 4 - Eficiência:

Evidência: Monitoramento do desempenho dos microsserviços, incluindo tempos de resposta, consumo de recursos e escalabilidade.

Teste: Realizar testes de carga e desempenho, simulando uma carga de trabalho elevada e avaliando o desempenho dos microsserviços, garantindo que eles sejam capazes de lidar com a demanda esperada.

Cenário 5 - Confiabilidade:

Evidência: Registro de incidentes e respostas de recuperação, incluindo tempo de recuperação, impacto no sistema e medidas tomadas para evitar futuros problemas.

Teste: Realizar testes de recuperação de falhas, como simular a queda de um microsserviço e verificar se o sistema é capaz de se recuperar e continuar operando de forma confiável.

Cenário 6 - Portabilidade:

Evidência: Registro de implantações bem-sucedidas em diferentes ambientes e plataformas, demonstrando a capacidade do sistema de ser implantado e executado em diferentes contextos.

Teste: Realizar testes de implantação em diferentes ambientes, como ambientes de desenvolvimento, homologação e produção, verificando se o sistema pode ser implantado de maneira consistente e executado corretamente em cada ambiente.

6.3. Evidências da Avaliação

Segue relatório de medidas registradas na coleta de dados.

Atributo de Qualidade:	Interoperabilidade
Requisito de Qualidade:	O sistema deve se comunicar com outras tecnologias.
Preocupação:	
O sistema deve ter como resposta a uma requisição uma saída de fácil leitura por outro componente.	

Cenário(s):	
Para permitir se integrar ou compartilhar seus recursos, é necessária uma opção de comunicação com nossos microsserviços.	
Ambiente:	
Microsservicos em orquestração, mas disponibilizado seus acessos via API Gateway	
Estímulo:	
O sistema de monitoramento envia uma requisição para o serviço REST do módulo de informações cadastrais.	
Mecanismo:	
Criar um serviço REST para atender às requisições do sistema de monitoramento	
Medida de resposta:	
Retornar os dados requisitados no formato JSON	
Considerações sobre a arquitetura:	
Riscos:	Alguma instabilidade na rede pode deixar a conexão lenta ou mesmo a perda de pacotes.
Pontos de Sensibilidade:	Não há
Tradeoff:	Não há

6.4. Resultados Obtidos

Segue os resultados referente aos requisitos não funcionais.

Requisitos Não Funcionais	Teste	Homologação
RNF01: Realizar todas as entregas com tempo médio inferior a 5 dias úteis.	Ok	Ok
RNF02: Expandir a atuação para mais 200 municípios de pequeno/médio porte.	Ok	N.A.
RNF03: Atuar na região norte do Brasil, último reduto que a empresa ainda não cobre, estabelecendo parcerias com uma empresa aérea e outras empresas de logística terrestres locais.	Ok	Ok
RNF04: Desenvolver novas parcerias com outras transportadoras para complementar a atuação em lugares onde a empresa apresenta market share inferior a 10%.	Ok	N.A.
RNF05: Fazer convênio com no mínimo 50 novos clientes, preferencialmente do ramo supermercadista.	N.A.	N.A.

RNF06: Crescer 10% em termos de faturamento global.	N.A.	N.A.
RNF07: Garantir a rastreabilidade de 100% das operações realizadas.	Ok	Ok
RNF08: Assegurar uma prestação de contas adequada 100% e uma gestão financeira transparente.	N.A.	N.A.
RNF09: Gerenciar os estoques de forma 100% eficiente	N.A.	N.A.

7. Avaliação Crítica dos Resultados

Um novo projeto de arquitetura distribuída de microsserviços para uma empresa de transportes que tem por objetivo expandir seus negócios e ao mesmo tempo ter a missão de integrar ou substituir seus sistemas legados, chegamos na conclusão abaixo após este trabalho.

Pontos Positivos:

Escalabilidade: A arquitetura distribuída de microsserviços permite que a empresa de logística se expanda de forma flexível, adicionando ou removendo microsserviços de acordo com as necessidades. Isso possibilita a escalabilidade do sistema, suportando o aumento da demanda e adaptando-se facilmente às mudanças no mercado.

Modularidade: Ao dividir o sistema em microsserviços independentes, é possível obter um alto grau de modularidade. Cada microsserviço pode ser desenvolvido, testado e implantado separadamente, o que facilita a manutenção e a evolução do sistema como um todo. Além disso, a modularidade permite que diferentes equipes trabalhem em paralelo, acelerando o processo de desenvolvimento.

Resiliência: A arquitetura distribuída de microsserviços torna o sistema mais resiliente a falhas. Caso um microsserviço apresente algum problema, apenas essa parte específica será afetada, enquanto os demais continuarão funcionando normalmente. Isso reduz o impacto de falhas e ajuda a manter a continuidade das operações da empresa de logística.

Pontos Negativos:

Complexidade de integração: Um dos principais desafios desse projeto é a integração dos microsserviços com o ecossistema legado da empresa de logística. A complexidade aumenta à medida que são necessárias adaptações para conectar os novos microsserviços com os sistemas existentes. É importante realizar um planejamento cuidadoso e investir em ferramentas e tecnologias adequadas para facilitar essa integração.

Sobrecarga inicial: A implementação de uma arquitetura distribuída de microsserviços requer um investimento inicial considerável. É necessário desenvolver e implantar os microsserviços, configurar a infraestrutura de suporte, estabelecer mecanismos de comunicação entre os serviços e treinar a equipe responsável. Essa sobrecarga inicial pode demandar recursos financeiros e tempo antes de se obter os benefícios completos do projeto.

Complexidade operacional: A operação de um sistema distribuído de microsserviços exige um nível de expertise técnica mais elevado por parte da equipe de TI. A gestão dos microsserviços, monitoramento, depuração de problemas de comunicação e coordenação de diferentes equipes de desenvolvimento podem se tornar desafios adicionais. É importante investir em treinamento adequado para garantir uma operação eficiente e minimizar possíveis problemas.

8. Conclusão

Ao finalizar esse trabalho de oferecer uma nova solução de arquitetura, identificamos que as novas tecnologias e implantações facilitara e ajudara a empresa a crescer. Mas, não podemos apenas se preocupar em fazer novas entregas, mas conviver com o que já está no ar, mesmo tendo suas fraquezas, o sistema legado ajudou a empresa a chegar aonde está.

Em se falando de crescimento pessoal e profissional, eu aprendi muito e além, pois colocando em prática todo o conhecimento adquirido com a pós-graduação em Arquitetura de Software, posso afirmar que estarei preparado para os novos desafios.

Eu acredito que o trabalho entregue precisa ser melhorado, alguns pontos podem ter falhas ou não serem eficientes ao problema proposto. As imagens não ficaram boas ao serem importadas para o Word. Eu identifiquei que poderia ter mais testes ou aprofundar mais nas seções melhoria da qualidade deste trabalho. E no item 6.3, não consegui entregar todos os pontos solicitados a esta atividade.

Lições aprendidas:

Aprendizado de Tecnologia:

Meu conhecimento de tecnologias de microsserviços: Durante o projeto, adquiri conhecimentos e habilidades relacionados a tecnologias específicas de microsserviços, como contêineres, orquestração de contêineres (por exemplo, Kubernetes), balanceamento de carga, serviços de descoberta e gateway de API. Falo

sobre como essas tecnologias foram usadas no contexto da empresa de transporte e os desafios que enfrentei ao estudá-las.

Minha experiência com ferramentas de desenvolvimento: Ao trabalhar com uma arquitetura distribuída de microsserviços, utilizei uma variedade de ferramentas para desenvolvimento, implantação, monitoramento e teste. Descrevo as ferramentas específicas que usei, como ambientes de desenvolvimento integrados (IDEs), sistemas de controle de versão, infraestrutura como código, ferramentas de monitoramento e registro de logs, entre outras.

Lidar com o Tempo de Projeto:

Meu gerenciamento de projeto ágil: Ao lidar com um projeto de arquitetura de software de grande escala, é importante considerar uma abordagem de gerenciamento de projeto ágil.

Minha análise e mitigação de riscos: Durante o desenvolvimento da arquitetura distribuída de microsserviços, certamente identifiquei e evitei riscos.

Similaridades com outras Rotinas de Trabalho e Estudo:

Aplicabilidade em outras áreas: Embora meu estudo de caso tenha sido na área de transporte, destaco como os conceitos e as lições aprendidas com a arquitetura de microsserviços podem ser aplicados em outras áreas.

Relevância acadêmica: Percebo a importância do estudo realizado em minha pesquisa acadêmica. Como a adoção de uma arquitetura distribuída de microsserviços contribui para o campo de arquitetura de software e quais são as implicações práticas e teóricas desse estudo.

Referências

MARTIN, Robert C. Arquitetura limpa: O guia do artesão para estrutura e design de software. São Paulo: Novatec Editora, 2018.

FEATHERS, Michael C. Trabalho Eficaz com Código Legado. São Paulo: Alta Books, 2011.

GAMMA, Erich et al. Padrões de Projetos: Soluções Reutilizáveis de Software Orientados a Objetos. Porto Alegre: Bookman, 2000.

EVANS, Eric. Domain-Driven Design: Atacando as complexidades no coração do software. São Paulo: Casa do Código, 2015.

MARTIN, Robert C. Código limpo: Habilidades práticas do Agile Software. São Paulo: Alta Books, 2019.