UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Leonardo Leite

Migração e replicação de dados de uma plataforma de financiamento de veículos

Florianópolis
2023

Leonardo Leite

Migração e replicação de dados de uma plataforma de financiamento de veículos

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Felipe Gomes de Oliveira Cabral, Dr.
Supervisor: Gabriel José Prá Gonçalves, Eng.

Florianópolis
2023

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.
Orientações em:
http://portalbu.ufsc.br/ficha

Leonardo Leite

Migração e replicação de dados de uma plataforma de financiamento de veículos

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 14 de dezembro de 2023.

Prof. Marcelo de Lellis Costa de Oliveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Felipe Gomes de Oliveira Cabral, Dr.
Orientador
UFSC/CTC/DAS

Gabriel José Prá Gonçalves, Eng.
Supervisor
Jungsoft GmbH

Prof. Publio Macedo Monteiro Lima, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

Ao meu amado avô, cuja memória permanece viva em nossos corações, e à minha família, fonte constante de amor e apoio, mesmo nas ausências que a vida nos impõe.

AGRADECIMENTOS

Agradeço à minha amada família, minha mãe Roseli, minha irmã Daiane e ao sempre jovem Vicente, que, mesmo diante das dificuldades impostas pela vida, foram meu alicerce e suporte ao longo desses anos de faculdade e na experiência de morar em outra cidade.

A Gabriel Prá e Rafael Jung, meus companheiros na empresa, agradeço por terem confiado no meu trabalho desde o início e por serem fundamentais para as oportunidades incríveis que surgiram em minha vida.

Ao grupo dos "Jantadores", amigos que tornaram minha jornada na faculdade inesquecível, meu coração se enche de gratidão. Especialmente a Gustavo, Yuri, Diego e Maurici, responsáveis pelas melhores memórias da minha vida. Agradeço por cada aventura e por transformarem os desafios acadêmicos em conquistas coletivas.

Aos mentores que me guiaram até aqui, meu especial reconhecimento. Thiago Prati, professor da vida desde o ensino médio, sua orientação foi um farol constante em meu caminho. Expresso profunda gratidão a Felipe Cabral, que, mesmo atravessando um dos momentos mais importantes de sua vida, ofereceu um apoio inabalável no desenvolvimento deste projeto.

Cada um de vocês é uma peça preciosa na construção da minha história. Muito obrigado por fazerem parte dela.

“Elixir foi desenvolvimento para ser extensível,
e a maneira de como ele vai ser estendido
sempre foi e será um esforço de comunidade.”
(Valim, 2022)

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 14 de dezembro de 2023.

Na condição de representante da Jungsoft GmbH na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.

Documento assinado digitalmente
GABRIEL JOSE PRA GONCALVES
Data: 03/12/2023 18:41:55-0300
CPF: ***.817.149-**
Verifique as assinaturas em https://v.ufsc.br

Gabriel José Prá Gonçalves
Jungsoft GmbH

RESUMO

O projeto em questão centra-se na complexa integração entre uma startup especializada em leasing de veículos e uma empresa consolidada no comércio automobilístico alemão. O principal desafio envolve a migração de mais de 10 mil ofertas do banco de dados da startup para a plataforma da empresa compradora. Essa operação é desafiadora devido às características distintas de cada plataforma e à divergência arquitetônica que as diferencia. A tarefa não se limita à movimentação das ofertas existentes, mas também à garantia de uma sincronização contínua para manter ambas as plataformas atualizadas e operacionais. A coexistência de ambas é crucial, pois certas funcionalidades exclusivas da startup devem continuar a operar mesmo após a migração. O desenvolvimento do projeto é separado em 3 fases distintas, sendo primeira a de análise de ambas as arquiteturas, buscando identificar pontos de convergência e divergência, a segunda a de implementação de um programa capaz de migrar as ofertas de maneira contínua, e por fim, a terceira sendo a construção de um plano estratégico para a migração das funcionalidades visando projetos futuros. O programa foi implementado utilizando a linguagem de programação Elixir, com a arquitetura baseada no formato Umbrella. A solução implementada faz o uso de Cronjobs, mecanismo para agendamento de tarefas periódicas, e do envio de arquivos em formato de CSV, que é bastante utilizado pela empresa receptora. O programa desenvolvido cumpre o requisito de enviar as informações das ofertas de maneira contínua para a plataforma receptora. O resultado obtido para migração foi satisfatório, visto que aproximadamente 95% das ofertas foram migradas com sucesso. Contudo, alguns problemas de desempenho foram identificados. Por fim, o projeto apresenta um possível plano estratégico que pode ser utilizado para uma futura migração de funcionalidades de uma plataforma para outra.

Palavras-chave: Migração. Leasing. Elixir. Umbrella. Cronjobs. CSV.

ABSTRACT

The project in question focuses on the complex integration between a startup specialized in vehicle leasing and an established company in the German automobile trade. The main challenge involves migrating more than 10,000 offers from the startup's database to the purchasing company's platform. This operation is challenging due to the distinct characteristics of each platform and the architectural divergence that differentiates them. The task is not limited to moving existing offerings, but also ensuring continuous synchronization to keep both platforms up to date and operational. The coexistence of both is crucial, as certain features exclusive to the startup must continue to operate even after the migration. The development of the project is separated into 3 distinct phases, the first being the analysis of both architectures, seeking to identify points of convergence and divergence, the second the implementation of a program capable of migrating offers continuously, and finally, the third being the construction of a strategic plan for the migration of functionalities for future projects. The program was implemented using the Elixir programming language, with an architecture based on the Umbrella format. The implemented solution makes use of Cronjobs, a mechanism for scheduling periodic tasks, and sending files in CSV format, which is widely used by the receiving company. The developed program meets the requirement of sending offer information continuously to the receiving platform. The migration result was satisfactory, as approximately 95% of the offers were successfully migrated. However, some performance issues were identified. Finally, the project presents a possible strategic plan that can be used for a future migration of functionalities from one platform to another.

Keywords: Migration. Leasing. Elixir. Umbrella. Cronjobs. CSV.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

No cenário dinâmico do comércio automotivo online, fusões e aquisições frequentemente configuram momentos cruciais na evolução de organizações e na busca pela eficiência operacional. Neste contexto, há a trajetória de duas entidades, ambas atuando como portais de venda de veículos, cada uma contribuindo com uma abordagem distinta, agora convergindo em um objetivo comum. Essa história destaca como a visão estratégica e o foco nas necessidades do mercado podem catalisar mudanças profundas em empresas e setores inteiros.

Ambas as entidades compartilham um objetivo: servir como plataformas de veiculação de ofertas automotivas, combinando a funcionalidade de um classificado automotivo com recursos projetados de forma exclusiva para a realidade da compra e venda de veículos. É importante ressaltar que essas empresas não vendem veículos diretamente, mas fornecem suas plataformas para que fornecedores e concessionárias ofereçam seus serviços.

Neste contexto de compra e venda de veículos, dois tipos de contratos têm sido tradicionalmente predominantes: o contrato de venda e o contrato de locação. No contrato de venda, o cliente adquire o veículo como seu ativo, enquanto no contrato de locação, o veículo é alugado e a propriedade permanece com a concessionária. No entanto, uma terceira opção ganhou destaque e atrai cada vez mais consumidores na Europa: o contrato de *leasing* de carros.

O contrato de *leasing* combina elementos dos contratos de venda e locação, oferecendo vantagens distintas que têm conquistado os motoristas europeus. Em um contrato de *leasing*, o cliente tem o uso do veículo, mas não se torna o proprietário legal. Em vez disso, o carro permanece propriedade da empresa de *leasing* durante o contrato.

Em suma, em um contrato de venda, o cliente adquire o veículo como seu ativo, tornando-se o proprietário legal, sendo responsável por todos os custos de manutenção, seguro e depreciação do veículo. A flexibilidade para trocar de veículo é limitada, uma vez que o cliente geralmente precisa vendê-lo ou fazer uma troca direta.

Em um contrato de locação, o veículo é alugado, e o cliente não se torna o proprietário. A manutenção e os custos de seguro podem ser incluídos no contrato, simplificando a experiência de uso. Além disso, a troca de veículo é mais fácil no final do contrato, pois o cliente pode simplesmente devolvê-lo e escolher um novo.

Em um contrato de *leasing*, o cliente tem o uso do veículo, mas a propriedade permanece com a empresa fornecedora de *leasing*. Os custos de manutenção e seguro também podem ser incluídos no contrato, proporcionando uma experiência conveniente. A troca de veículo também é facilitada no final do contrato, oferecendo uma maior flexibilidade para experimentar diferentes modelos. Os custos iniciais são frequente-

mente mais baixos do que a compra, tornando o leasing acessível para um público mais amplo.

Enquanto o contrato de *leasing* oferece diversas vantagens em relação ao contrato de locação, é importante observar alguns pontos negativos associados à locação que não estão presentes no *leasing*. Por exemplo, no contrato de locação, o cliente não possui a flexibilidade de escolher diferentes modelos de veículos com a mesma facilidade do *leasing*, pois está preso ao contrato de locação por um período determinado. Além disso, os custos totais de propriedade podem ser mais altos em um contrato de locação devido à necessidade de realizar manutenção e serviços de seguro separadamente.

Os benefícios do contrato de *leasing* têm sido uma razão convincente para a crescente popularidade dessa modalidade na Europa. De acordo com dados da Associação Alemã de Leasing (BDL), em 2020, cerca de 42% dos veículos novos registrados na Alemanha foram financiados por meio de contratos de *leasing* (BDL, 2020). Além disso, em nível europeu, estatísticas da *Leaseurope* indicam um aumento significativo na adoção deste tipo de contrato, com uma média de 12% de aumento em 2023 comparado ao mesmo período em 2022 (LEASEUROPE. . . , 2023). Isso demonstra como este produto se tornou uma escolha preferencial entre os consumidores alemães e europeus em geral.

Por estes motivos, uma entidade já consolidada no ramo de venda de veículos decidiu adquirir uma *startup* especializada em *leasing* de carros e ingressar nesse mercado. O *leasing* emergiu como uma escolha inteligente e conveniente, e a empresa viu na aquisição da *startup* a oportunidade de diversificar seus serviços, e consequentemente, atender a uma gama mais ampla de clientes e expandir sua presença no mercado europeu de veículos.

1.1 A MIGRAÇÃO DOS DADOS

A aquisição da *startup* pela empresa de vendas automobilística impôs um desafio premente: a integração dos dados internos nos sistemas das empresas. Essa necessidade emerge da difícil situação enfrentada pelos fornecedores, que agora se veem sobrecarregados com a gestão de duas plataformas distintas pertencentes à mesma marca.

A união dessas duas empresas trouxe consigo uma convergência de abordagens complementares no mercado automotivo online, prometendo uma experiência mais abrangente e diversificada para os clientes. No entanto, essa convergência também resultou na coexistência de sistemas de informação e fluxos de dados previamente independentes.

Desta forma, a integração e migração entre os dois sistemas se tornam necessárias principalmente pelos seguintes motivos:

- **Complexidade Técnica:** Os sistemas das duas plataformas, até então operando de maneira autônoma, possuem estruturas técnicas distintas, formatos de dados diferenciados e regras de validação específicas. A integração interna é vital para superar essas diferenças e garantir a coerência e a precisão das informações.
- **Gerenciamento de Dados:** Com informações provenientes de duas fontes diferentes, a gestão de dados torna-se consideravelmente mais complexa para os fornecedores. Isso abrange registros de transações, detalhes dos clientes, dados de inventário e outros aspectos cruciais do negócio.
- **Integridade e Qualidade dos Dados**: Assegurar a integridade e a qualidade dos dados é imperativo para evitar erros e inconsistências. A integração interna requer medidas rigorosas de validação e verificação de dados.
- **Eficiência Operacional:** A integração interna visa também melhorar a eficiência operacional, uma vez que os fornecedores não precisam duplicar esforços ao lidar com ambas as plataformas de forma separada.
- **Alívio para Fornecedores:** O principal objetivo da integração interna é aliviar a carga de trabalho dos fornecedores, permitindo que eles se concentrem em suas operações principais, sem a necessidade de realizar a integração no lado deles.

Nesse contexto amplo, a migração de dados transcende as barreiras meramente técnicas, assumindo uma dimensão estratégica que visa forjar uma conexão entre a inovação da *startup* e a tradição da empresa compradora. O projeto aspira a criar um futuro coeso e eficiente no cenário de venda de veículos na Alemanha, onde a convergência dessas duas entidades resultará em uma oferta mais robusta e adaptável para os clientes, combinando o melhor de ambos os mundos.

Além desta migração de dados, há a intenção da remoção completa do banco de dados da *startup* e a consolidação de funcionalidades internas. Essas etapas não apenas exigem uma abordagem técnica meticulosa, mas também demandam uma compreensão profunda das implicações estratégicas e operacionais dessa integração.

Embora os desafios sejam notáveis, esta aquisição também oferece oportunidades consideráveis. A integração bem-sucedida das duas plataformas pode resultar em uma oferta mais robusta para os clientes, maior eficiência operacional e o aumento da base de clientes. À medida que os fornecedores enfrentam esses desafios, eles também adquirem conhecimentos valiosos em integração de sistemas e operações complexas, que podem ser aplicados em outros contextos futuros.

1.2 OBJETIVOS

Baseando-se no problema apresentado, o projeto de migração possui os seguintes objetivos principais:

- Transferência das ofertas da *startup* para a plataforma da empresa compradora de maneira eficiente e sem perda de dados.
- Garantir a integridade e consistência dos dados durante e após o processo de migração.
- Estabelecer mecanismos para garantir a sincronização contínua entre as duas plataformas, permitindo a coexistência de ambas.
- Desenvolver um planejamento estratégico para lidar com as implicações de longo prazo, considerando a eventual desativação do *backend* da *startup* e a consolidação total de funcionalidades.

1.3 ORGANIZAÇÃO DO DOCUMENTO

Este documento é organizado da seguinte forma:

- No Capítulo 2, Empresas Envolvidas e Características, são detalhadas as empresas envolvidas e o contexto no qual o problema se encontra.
- No Capítulo 3, Fundamentação Teórica, são apresentados os detalhes de como as ferramentas que foram utilizadas para o desenvolvimento da solução funcionam.
- No Capítulo 4, Desenvolvimento, é detalhado um plano estratégico para resolução do problema apresentado.
- No Capítulo 5, Implementação, é detalhado como a solução é implementada no contexto da *startup*.
- No Capítulo 6, Análise de Resultados, é apresentado uma análise dos resultados obtidos a partir da solução implementada.
- No Capítulo 7, Conclusão, é apresentado um resumo do projeto e um planejamento futuro.

2 EMPRESAS ENVOLVIDAS E CARACTERÍSTICAS

O projeto foi realizado na empresa de desenvolvimento web chamada Jungsoft. A Jungsoft foi contratada para a realização de uma migração de dados entre as plataformas de duas empresas de venda e *leasing* de veículos, que estão passando por um processo de fusão.

2.1 JUNGSOFT

A *Jungsoft* é uma empresa de desenvolvimento de softwares web modernos, com funcionamento 100% remoto. A empresa foi estabelecida em 2018 e hoje conta com aproximadamente 15 funcionários, divididos entre desenvolvedores, engenheiros e designers. Por ser totalmente remota, a empresa contrata funcionários de todas as regiões do Brasil e também da Alemanha.

O modelo de negócio atual da *Jungsoft*, se baseia em ser responsável por todo o setor de tecnologia de uma empresa contratante. É um produto interessante principalmente para *startups*, pois a Jungsoft arca com toda a responsabilidade de contratação de desenvolvedores e de engenheiros, além do desenvolvimento e manutenção de seus sistemas internos.

Neste projeto, a *Jungsoft* é a empresa responsável pelo desenvolvimento e manutenção de toda a infraestrutura tecnológica da *startup* que foi vendida, e portanto, é um elemento essencial neste processo de fusão.

2.2 EMPRESA COMPRADORA

A empresa compradora, que terá seu nome anônimo durante o desenvolvimento deste projeto, é uma renomada organização com presença sólida no mercado de veículos na Alemanha. Com uma trajetória marcada por anos de atuação e reconhecimento nacional, ela figura entre as marcas mais conhecidas e confiáveis pelos alemães.

Especializada no ramo de vendas de automóveis, a empresa disponibiliza uma plataforma digital que serve como um ambiente de destaque para a exposição de ofertas de veículos. Seu foco principal é apresentar essas ofertas, funcionando como uma espécie de classificados automotivos, mas com recursos e funcionalidades projetados sob medida para atender as necessidades específicas do mercado de venda de veículos. As ofertas na plataforma podem ser tanto de veículos novos quanto de usados, e são criadas por fornecedores, que podem ser concessionárias ou empresas revendedoras de veículos.

É importante ressaltar que a empresa atua predominantemente como intermediária na transação de veículos, fornecendo sua plataforma para que fornecedores ofereçam seus serviços. Ela não realiza diretamente a venda, mas proporciona um

ambiente digital onde as ofertas são apresentadas, facilitando assim a conexão entre compradores e vendedores.

A empresa compradora opera em um ambiente tecnológico robusto baseado em uma arquitetura de microsserviços, que em teoria, fornece eficiência e escalabilidade à plataforma. Um componente essencial dessa arquitetura é o banco de dados NoSQL MongoDB, que desempenha um papel central na gestão e armazenamento eficaz dos dados.

A estrutura de microsserviços da empresa compradora é separada em diversas áreas que refletem também, a forma como a mesma é organizada. Neste sentido, há microsserviços criados para lidar com as informações de contas, usuários, fornecedores, segurança, infraestrutura interna, dentre outros. Neste projeto em questão, será tratado apenas de uma área específica deste sistema, que é chamada de *Händlerbereich*, que em tradução direta significa "Área dos fornecedores". Esta área do sistema engloba todos os microsserviços utilizados para todas as funcionalidades dos fornecedores, incluindo os responsáveis pela criação de ofertas na plataforma. Tais microsserviços são os que terão maior impacto durante a migração.

2.3 STARTUP

A *startup*, que também terá seu nome anônimo, é uma empresa especializada na área de financiamento de veículos. Ela apresenta um sistema de negócio bem semelhante ao da empresa compradora, fornecendo uma plataforma com veículos novos e usados, e que é alimentada por fornecedores, que podem ser concessionárias ou empresas revendedoras de veículos.

Contudo, diferentemente da empresa compradora, a *startup* concentra seus esforços em um nicho específico do mercado automotivo. Seu principal produto e área de atuação são os contratos de *leasing* de veículos. Nesta área, ela possui diversos concorrentes diretos, que também possuem uma plataforma somente para este fim. O ponto-chave que a diferencia das demais plataformas, é a flexibilidade desse modelo, pois não exige que os clientes façam um pagamento inicial em dinheiro no momento da solicitação do contrato.

Por ser uma *startup* pequena, ela resolveu focar a sua organização interna em analistas da área e de profissionais da área comercial. Neste sentido, a *Jungsoft* foi contratada para ser responsável por todo o setor tecnológico, que envolve a contratação de desenvolvedores e de engenheiros, além do desenvolvimento e manutenção de todos os seus sistemas internos.

A *startup* opera em um ambiente tecnológico robusto baseado em uma arquitetura *Umbrella*, que é proporcionada pelo uso da linguagem de programação *Elixir*. A utilização destes elementos fornecem, em teoria, eficiência e escalabilidade à plataforma. Um componente essencial dessa arquitetura é o banco de dados *PostgreSQL*,

que desempenha um papel central na gestão e armazenamento eficaz dos dados.

2.4 CARACTERÍSTICAS DAS OFERTAS

As ofertas em ambas as plataformas, independentemente do tipo de contrato, referem-se a veículos. Estes que podem ser novos ou usados, conforme divulgados pelos fornecedores.

Dado o propósito de exibir opções de venda de veículos ao usuário final, ambas as plataformas devem aderir às diretrizes alemãs para venda de veículos. Isso implica que as ofertas devem conter informações essenciais sobre o veículo, como potência do motor, tipo de combustível, modelo, ano de lançamento e dados sobre consumo energético. Para carros usados, é necessário fornecer detalhes sobre o estado atual, como a quilometragem percorrida.

A observância da nova lei de proteção de dados da União Europeia é crucial. Ambas as plataformas são obrigadas a apresentar essas informações de maneira clara e concisa, em conformidade com os requisitos legais estabelecidos.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os componentes que integram a arquitetura interna e que também serão utilizados no desenvolvimento e implementação da solução mais adiante.

3.1 MIGRAÇÃO POR REPLICAÇÃO

A migração por replicação refere-se a um processo no qual dados, sistemas ou aplicativos são transferidos de um ambiente para outro por meio da replicação contínua de informações. Esse método é frequentemente usado em situações em que é necessário minimizar o tempo de inatividade e garantir a consistência dos dados durante a transição.

No contexto de banco de dados, por exemplo, uma migração por replicação pode envolver a criação de uma cópia em tempo real dos dados de um banco de dados em um servidor para outro. Isso é alcançado por meio da replicação contínua de alterações feitas no banco de dados de origem para o banco de dados de destino. Durante esse processo, as atualizações, inserções e exclusões são replicadas de forma que o banco de dados de destino permaneça sincronizado com o banco de dados de origem.

A migração por replicação é frequentemente escolhida quando é essencial manter a continuidade do serviço ou minimizar o impacto nas operações regulares. Esse método permite que as duas instâncias (origem e destino) operem simultaneamente durante o período de migração, reduzindo assim o tempo de inatividade e garantindo que as informações mais recentes estejam disponíveis no novo ambiente.

Além de migrações de banco de dados, o conceito de migração por replicação pode ser aplicado a outros contextos, como a transferência de sistemas ou aplicativos entre servidores ou ambientes de nuvem, mantendo a sincronização contínua para garantir uma transição suave e sem perda de dados.

3.2 METODOLOGIA *SCRUM*

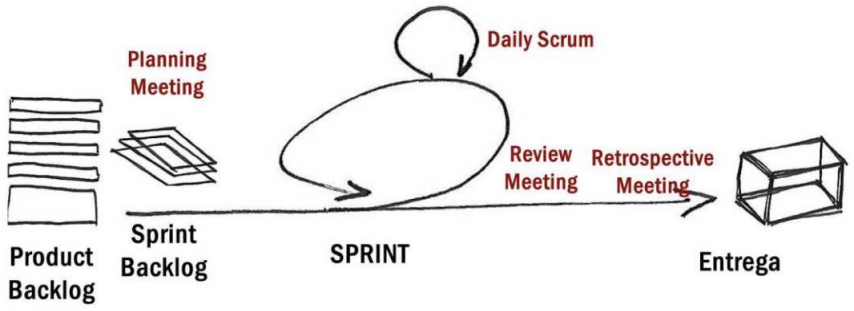
Segundo (RAFAEL DIAS RIBEIRO, 2015), a Metodologia *Scrum* é um método ágil empírico, iterativo com entregas incrementais. É reconhecida por sua flexibilidade e ênfase no desenvolvimento e manutenção de projetos/produtos complexos. Inicialmente desenvolvido para projetos de desenvolvimento de software, o *Scrum* ganhou ampla aceitação em diversos setores devido à sua eficácia em lidar com mudanças constantes.

As atividades do *Scrum* são subdivididas em *Sprints*, ciclos de desenvolvimento com duração fixa, geralmente de duas a quatro semanas. Ao iniciar cada *Sprint*, a

equipe seleciona itens do *Backlog* do Produto para desenvolver durante a *Sprint*, visando entregar um projeto funcional ao término.

O *Scrum Master* desempenha o papel de facilitador, eliminando obstáculos e promovendo a conformidade com os princípios e práticas do *Scrum*. O conjunto de reuniões *Scrum*, como *Sprint Planning*, *Daily Scrum*, *Sprint Review* e *Sprint Retrospective*, estabelece um processo estruturado para a comunicação eficaz e a melhoria contínua. Na Figura 1 abaixo, é possível observar os detalhes de como a metodologia é organizada

Figura 1 – Formato de organização da metodologia *Scrum*.



Fonte: (RAFAEL DIAS RIBEIRO, 2015).

Os principais artefatos incluem o *Backlog* do Produto, que abriga os requisitos priorizados, e o Incremento, a soma de todas as funcionalidades concluídas até o momento. A Metodologia *Scrum* enfatiza transparência, inspeção e adaptação, permitindo que as equipes respondam de forma ágil às mudanças nos requisitos e nas condições do projeto.

3.3 METODOLOGIA MOSCOW

Segundo (OLIVEIRA, R. R., 2014), "MoSCoW é uma maneira de classificar e priorizar requisitos para inclusão em um sistema de informação. Este mecanismo é utilizado no desenvolvimento no qual o escopo pode precisar ser redefinido de acordo com a evolução de progresso."

A metodologia utiliza como base a classificação de requisitos ou funcionalidades em quatro categorias principais: *Must-haves* (Deve ter), *Should-haves* (Deveria ter), *Could-haves* (Poderia ter) e *Won't-haves* (Não terá por enquanto). A sigla *MoSCoW* é derivada das iniciais dessas categorias.

- ***Must-haves***: Representam os requisitos essenciais para o sucesso do projeto. São funcionalidades sem as quais o produto ou projeto não será considerado

aceitável. A implementação desses itens é vital e não pode ser adiada.

- ***Should-haves***: Refere-se a requisitos que são importantes, mas não cruciais. A implementação desses itens é desejável e contribuirá significativamente para o valor do produto. Eles são priorizados após os *Must-haves* e, se possível, são incorporados ao escopo do projeto.
- ***Could-haves***: Incluem requisitos que são desejáveis, mas não são considerados essenciais. Sua implementação é opcional e pode ser considerada se houver recursos disponíveis após a conclusão dos itens *Must-haves* e *Should-haves*. São aspectos que agregam valor, mas não comprometem a viabilidade do projeto se não forem incluídos.
- ***Won't-haves***: São requisitos que foram deliberadamente identificados como não prioritários para a fase atual do projeto. Pode ser que esses itens sejam considerados em fases futuras ou em iterações posteriores. A decisão de não incluir esses requisitos é muitas vezes baseada em limitações de tempo, recursos ou prioridades estratégicas.

O uso da metodologia MoSCoW proporciona uma visão clara sobre as prioridades do projeto, ajuda na gestão eficaz do escopo e permite que as equipes e as partes interessadas tomem decisões informadas sobre o que deve ser entregue primeiro. Essa abordagem facilita a adaptação a mudanças nas condições do projeto e ajuda a garantir que os elementos mais críticos para o sucesso sejam abordados antes dos demais.

3.4 LINGUAGEM UML

A *UML*, ou Linguagem de Modelagem Unificada (em inglês, *Unified Modeling Language*), é uma linguagem visual padronizada amplamente utilizada na engenharia de software para modelar, documentar e projetar sistemas complexos. Segundo (MAURO NUNES, 2003), "é uma linguagem que utiliza uma notação padrão para especificar, construir, visualizar e documentar sistemas de informação orientados por objetos."

Criada na década de 1990, a *UML* oferece uma maneira eficaz de representar visualmente as diferentes perspectivas e elementos de um sistema, facilitando a compreensão e a comunicação entre os membros da equipe de desenvolvimento e outros *stakeholders*.

A principal finalidade da *UML* é fornecer uma linguagem comum que possa ser compreendida por analistas de negócios, desenvolvedores de software, gerentes de projeto e outros profissionais envolvidos no ciclo de vida do desenvolvimento de software. Ela engloba uma variedade de diagramas, cada um focado em aspectos

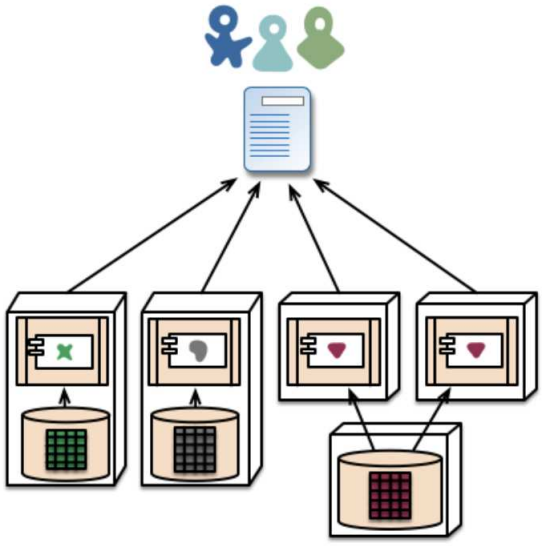
específicos do sistema, permitindo uma representação abstrata e visual de diferentes partes e processos.

3.5 ARQUITETURA DE MICROSSERVIÇOS

A arquitetura de microsserviços é uma variante da conhecida Arquitetura Orientada a Serviços *Service-Oriented Architecture (SOA))*, que, segundo (JAMES LEWIS, 2014), é uma abordagem para desenvolver um único aplicativo como um conjunto de pequenos serviços, cada um executando seu próprio processo e se comunicando a partir de mecanismos leves.

Este paradigma de design tem ganhado considerável destaque na atualidade devido à sua capacidade de promover a escalabilidade, flexibilidade e agilidade no desenvolvimento e manutenção de sistemas. Essa abordagem propõe a construção de um sistema como uma coleção de serviços independentes e autônomos, cada um responsável por uma funcionalidade específica. A Figura 2 abaixo representa de maneira gráfica como os microsserviços são organizados.

Figura 2 – Formato de organização de uma arquitetura de microsserviços.



Fonte: (JAMES LEWIS, 2014).

As principais características de uma arquitetura deste tipo, são:

- **Descentralização:** Os microsserviços operam de forma independente, cada um com seu próprio conjunto de dados e lógica de negócios. Isso reduz a complexidade do sistema global e permite a evolução e manutenção individual de cada serviço.

- **Escalabilidade Independente:** Cada serviço pode ser escalado de forma independente, respondendo às demandas de tráfego e carga de trabalho específicas. Isso otimiza o uso dos recursos disponíveis e melhora a eficiência operacional.
- **Flexibilidade Tecnológica:** Cada microsserviço pode ser desenvolvido utilizando tecnologias diferentes, desde que sejam compatíveis com a sua funcionalidade. Isso permite a escolha da melhor ferramenta para cada tarefa, levando a um sistema mais eficaz e especializado.
- **Resiliência e Tolerância a Falhas:** A arquitetura de microsserviços é projetada para ser resiliente a falhas. Se um serviço falha, isso não afeta necessariamente os outros serviços, garantindo maior disponibilidade e confiabilidade do sistema.
- **Rápida Implantação e Iteração:** Os microsserviços podem ser implantados e atualizados de forma independente, acelerando o ciclo de vida do software. Isso permite iterações rápidas e a introdução de novos recursos de maneira ágil.

A utilização desta arquitetura para um sistema depende de diversos fatores. Segundo (MENDES, 2021), há pontos positivos e negativos desta implementação.

Dentre os pontos positivos, pode-se destacar:

- **Alta escalabilidade:** A escalabilidade independente de cada microsserviço facilita a adaptação às mudanças nas demandas do sistema, resultando em uma escalabilidade mais eficaz.
- **Boa manutenção:** A modularidade dos microsserviços torna a manutenção e atualização do sistema mais fácil e segura, reduzindo o risco de impactos indesejados.
- **Evolução Acelerada:** A flexibilidade tecnológica e a rápida iteração promovem a inovação, permitindo que a empresa se mantenha competitiva em um ambiente dinâmico.

Dentre os pontos negativos, pode-se destacar:

- **Complexidade da Gestão:** A gestão de uma grande quantidade de microsserviços pode se tornar complexa, exigindo ferramentas e práticas eficientes para monitoramento, coordenação e gerenciamento.
- **Comunicação entre Serviços:** Uma comunicação excessiva entre os microsserviços pode impactar a performance do sistema, sendo necessário um cuidado especial no design das interações.

- **Consistência dos Dados:** Manter a consistência dos dados distribuídos entre os microsserviços pode ser desafiador, exigindo estratégias eficazes de gerenciamento de transações e concorrência.

3.6 MONGODB

O *MongoDB* é um sistema de gerenciamento de banco de dados *NoSQL* (não relacional) que se destaca pela sua flexibilidade e capacidade de armazenar e manipular dados de forma eficiente. Desenvolvido pela *MongoDB, Inc.*, o *MongoDB* é uma solução de código aberto que oferece uma abordagem diferente dos bancos de dados relacionais tradicionais.

Segundo (OLIVEIRA, S. S. de, 2014), "bancos de dados orientados a documentos são baseados no armazenamento de pares de chave-valor, tendo um esquema altamente flexível. Esta característica torna os bancos de dados orientados à documentos ótimas opções para dados semi-estruturados, como os utilizados em ferramentas web colaborativas."

O fato do *MongoDB* armazenar e organizar os dados em formato de documentos flexíveis, possibilitando uma estrutura dinâmica e adaptável, é o que permite uma conexão eficaz em uma estrutura de microsserviços.

Enquanto os bancos de dados não relacionais oferecem flexibilidade e escalabilidade, quando implementados em uma arquitetura de microsserviços complexa, podem intensificar os desafios inerentes à gestão de um ambiente distribuído. Por este motivo, a complexidade do sistema torna-se exponencial, tornando-se ainda mais complicado o processo de manutenção e inserção de novas funcionalidades.

3.7 POSTGRESQL

Segundo (CARVALHO, 2017), *PostgreSQL* é um sistema de gerenciamento de banco de dados relacional de código aberto conhecido por garantias estabelecidas de confiabilidade, de recursos de consulta e de uma operação previsível. Projetado para oferecer um ambiente de armazenamento eficiente e flexível, o PostgreSQL suporta a manipulação de dados em um formato de tabela relacional.

Este sistema proporciona uma variedade de recursos avançados, incluindo suporte a transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), procedimentos armazenados, gatilhos, visões e extensões personalizadas. O PostgreSQL é particularmente reconhecido por sua escalabilidade, sendo capaz de gerenciar grandes volumes de dados e fornecer um desempenho consistente mesmo em ambientes de alto tráfego.

A natureza de código aberto do PostgreSQL, aliada a uma comunidade ativa de desenvolvedores, contribui para atualizações regulares, correções de segurança

e melhorias contínuas. Sua flexibilidade e capacidade de personalização fazem dele uma escolha popular para uma variedade de aplicações, desde pequenos projetos até implementações corporativas mais complexas.

É frequentemente adotado por organizações que buscam uma solução de banco de dados confiável e de alto desempenho, beneficiando-se de sua arquitetura robusta e ampla gama de funcionalidades.

3.8 ELIXIR

Segundo (JURIC, 2015), "*Elixir* é uma linguagem de programação funcional moderna para a construção de sistemas escalonáveis, distribuídos e tolerantes a falhas em larga escala para a máquina virtual *Erlang*".

A linguagem *Elixir* segue os princípios da programação funcional, incluindo imutabilidade de dados, funções puras e alta ênfase em expressões, facilitando a escrita de código mais limpo, seguro e testável.

Uma das características mais marcantes é a sua eficácia no suporte à concorrência e ao paralelismo. Ela utiliza o modelo de atores, onde processos leves se comunicam por meio de mensagens, tornando a criação de sistemas concorrentes e distribuídos de alto desempenho uma tarefa mais simples.

Além disso, a combinação da linguagem com a máquina virtual *Erlang* oferece escalabilidade notável, sendo particularmente útil para sistemas em tempo real, como aplicativos de mensagens e sistemas de telecomunicações, que demandam alta concorrência e confiabilidade.

A linguagem *Elixir* é conhecida por sua sintaxe clara e expressiva, inspirada em *Ruby*, o que a torna fácil de ler e escrever. Ela também herda da máquina virtual *Erlang* uma ênfase em robustez e tolerância a falhas, sendo amplamente utilizada em sistemas críticos que não podem falhar, como os sistemas de telecomunicações.

Possui um ecossistema crescente de bibliotecas e *frameworks* que ajudam os desenvolvedores a criar uma variedade de aplicativos, desde sistemas distribuídos até aplicativos móveis.

Em resumo, ela acaba sendo uma escolha atraente para desenvolvedores que desejam criar sistemas concorrentes, escaláveis e robustos, especialmente em aplicativos em tempo real e sistemas distribuídos. Sua combinação de programação funcional, suporte à concorrência e sintaxe expressiva a torna uma ferramenta poderosa para enfrentar desafios técnicos complexos.

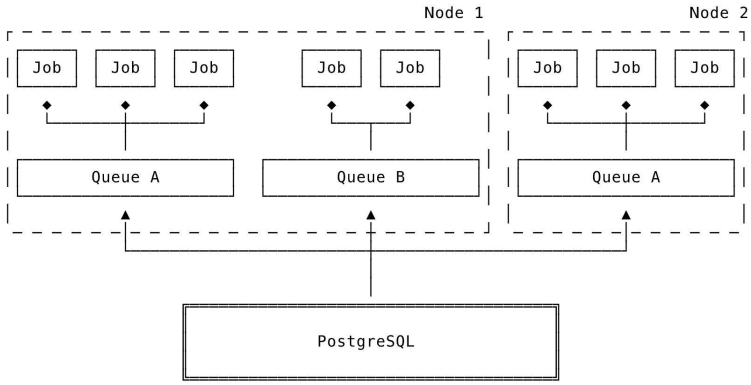
3.9 OBAN

A biblioteca *Oban*, no contexto da linguagem de programação Elixir, é uma ferramenta robusta e confiável projetada para gerenciar trabalhos em segundo plano.

Segundo (OBAN, 2015), uma de suas características distintivas é a persistência no banco de dados, o que significa que os dados sobre os trabalhos são armazenados no banco, garantindo a recuperação do estado mesmo em reinicializações do sistema. Isso contribui para a confiabilidade do sistema, especialmente em cenários de falha.

Além disso, o Oban oferece recursos avançados, como priorização e agendamento de trabalhos, permitindo que os desenvolvedores controlem a ordem de execução e programem tarefas para o futuro. O funcionamento destas filas, e como elas se conectam com o banco de dados, pode ser observado na figura 3 abaixo.

Figura 3 – Funcionamento das filas e conexão com o banco de dados.



Fonte: (OBAN, 2015).

A biblioteca também se destaca por sua capacidade de lidar com falhas de trabalhos de maneira eficaz. Em caso de erros, o Oban oferece mecanismos para reprogramar automaticamente os trabalhos, contribuindo para a tolerância a falhas.

A integração transparente com o sistema de transações do Ecto, uma biblioteca de mapeamento de objetos-relacionais para Elixir, facilita a coordenação entre trabalhos em segundo plano e operações de banco de dados. Além disso, o Oban fornece recursos de monitoramento e métricas, permitindo que os desenvolvedores avaliem o desempenho do sistema e identifiquem possíveis áreas de melhoria.

3.10 CRONJOB

Segundo (HIVELOCITY, 2015), um *cronjob*, no contexto da computação e sistemas operacionais baseados em *Unix*, refere-se a uma tarefa agendada ou programada para ser executada automaticamente em intervalos regulares, determinados pelo cronômetro do sistema (cron). O *cronjob* é uma forma de automatizar a execução de comandos ou *scripts* em momentos específicos, como diariamente, semanalmente, mensalmente ou em outros padrões predefinidos.

São amplamente utilizados para automatizar tarefas repetitivas, como backup de dados, atualizações de software, limpeza de arquivos temporários, entre outras atividades programadas.

3.11 ARQUITETURA *UMBRELLA*

Segundo (ELIXIR, 2023), a arquitetura *Umbrella* é um paradigma de desenvolvimento de software que se baseia na criação e organização de um projeto em unidades independentes e interconectadas, denominadas de aplicações *umbrellas*. Cada aplicação *umbrella* representa um módulo individual dentro de um projeto maior, mantendo sua própria lógica de negócios, funcionalidades e interfaces de usuário. Essa abordagem permite uma estrutura modular e escalável, facilitando o gerenciamento de sistemas complexos.

Este formato de arquitetura também é uma variante da conhecida Arquitetura Orientada a Serviços (*Service-Oriented Architecture (SOA)*), que, como já comentado, é uma abordagem em que diferentes responsabilidades são distribuídas em serviços independentes, que se comunicam por meio de uma rede.

A principal característica desta arquitetura é a sua capacidade de divisão do projeto em partes distintas, cada uma delas comportando funcionalidades específicas. Essas partes podem ser desenvolvidas, testadas e mantidas independentemente, promovendo eficiência e agilidade no ciclo de vida do software. Cada aplicação possui sua própria hierarquia de diretórios, modelos de dados, serviços e interfaces, criando um ambiente propício para a reutilização de código e o isolamento de responsabilidades.

O principal benefício deste formato de arquitetura está em sua modularidade, que permite a escalabilidade e flexibilidade do sistema, pois cada aplicação pode ser escalado individualmente em resposta às demandas do sistema. Além disso, possibilita o desenvolvimento paralelo, onde equipes distintas podem trabalhar simultaneamente em diferentes partes do projeto, acelerando o processo de desenvolvimento.

Outro aspecto relevante é a manutenção simplificada. Segundo (GREGORI, 2020), o isolamento de funcionalidades em aplicação independentes facilita a detecção e correção de erros, tornando a manutenção do código mais eficiente e controlada. Além disso, a arquitetura *Umbrella* promove a reutilização de código entre os serviços, contribuindo para uma prática de desenvolvimento sustentável.

4 DESENVOLVIMENTO

Neste capítulo será tratado sobre como será a arquitetura da solução para a migração de dados e a maneira como ela deverá ser implementada.

4.1 IMPACTOS NA EMPRESA COMPRADORA

Um dos maiores desafios de arquitetura de microsserviços é manter uma gestão eficiente. A dificuldade envolvida neste processo cresce de maneira exponencial conforme o tamanho da aplicação aumenta. Neste contexto, torna-se evidente que a decisão da empresa compradora de implementar uma arquitetura de microsserviços para seus sistemas comprometeu a sua eficiência, principalmente devido ao grande porte da organização. O tamanho da plataforma levou a uma quantidade de microsserviços que ultrapassou o limite de complexidade previsto, revelando uma considerável violação da regra de descentralização, um dos aspectos negativos dessa arquitetura ao lidar com a gestão de projetos extensos.

Atualmente, a empresa compradora possui mais de mil microsserviços criados. Por este motivo, a ineficiência já é uma realidade constatada. A complexidade operacional torna-se um desafio imediato, demandando uma carga de trabalho considerável para implantação, monitoramento, atualizações e escalabilidade de cada serviço de forma independente.

A comunicação entre os microsserviços, à medida que o número aumenta, apresenta um *overhead* significativo. Especialmente em interações frequentes entre diversos serviços para completar uma tarefa, esse aumento na comunicação impacta diretamente na latência e no desempenho do sistema em sua totalidade.

A coordenação entre os inúmeros microsserviços também se mostra problemática. Em operações envolvendo transações ou consistência global, garantir a integridade dos dados em transações distribuídas torna-se uma tarefa complexa que demanda esforço adicional.

Infelizmente, uma migração de arquitetura em um sistema com mais de mil micros serviços criados beira ao impossível devido a uma série de desafios significativos que surgem nesse contexto.

Primeiramente, a complexidade operacional atinge níveis quase insuperáveis. Gerenciar a migração de milhares de micros serviços de forma independente, incluindo implantação, monitoramento e atualizações, torna-se uma tarefa monumental, exigindo uma carga de trabalho operacional gigantesca.

Além disso, a comunicação entre esses inúmeros micros serviços é um problema crítico. Coordenar a transição de uma arquitetura para outra sem comprometer a funcionalidade e a eficiência dos serviços é uma tarefa altamente complexa. A necessidade de coordenação e a garantia da consistência dos dados durante a migra-

ção são obstáculos adicionais quase intransponíveis. As transações distribuídas e a manutenção da integridade dos dados em um ambiente de transição são altamente desafiadoras, podendo resultar em erros catastróficos e perda de dados.

Diante dos desafios para o desenvolvimento de uma arquitetura que realize uma migração complexa de dados, torna-se evidente que o esforço necessário para implementar um algoritmo desta magnitude no sistema da empresa compradora não justifica os benefícios almejados, que seria de facilitar o processo de migração de dados.

4.1.1 Suporte para ofertas de *leasing*

No contexto de fornecer suporte para ofertas do tipo *leasing*, uma grande complexidade técnica surge para a empresa compradora no que se refere à incorporação deste tipo de produto em sua plataforma. Essa complexidade se desvela como uma questão crítica, em grande parte devido ao fato de que toda a estrutura da plataforma foi concebida desde o princípio com um foco nas ofertas de vendas. Essas ofertas, por sua vez, englobam uma série de campos obrigatórios que desempenham funções cruciais em diversos microsserviços.

O desafio mais proeminente nesse contexto reside na constatação de que tais informações, de suma importância para ofertas de venda, não têm o mesmo grau de obrigatoriedade quando se trata de ofertas de *leasing*. Em outras palavras, adaptar o sistema de modo a proporcionar suporte efetivo para ofertas de *leasing* requer a realização de alterações substanciais em uma ampla gama de microsserviços.

Como já enfatizado anteriormente, a tarefa de introduzir modificações nesse complexo ecossistema de microsserviços se revela um processo de notável complexidade e, acima de tudo, uma atividade que demanda tempo substancial.

4.2 IMPACTOS NA *STARTUP*

Diferente do que pode ser observado na realidade da empresa compradora, há uma notável facilidade no sistema desenvolvido pela *startup*, que é oferecida pela arquitetura *Umbrella* em conjunto com a linguagem de programação Elixir.

Em primeiro lugar, é crucial mencionar que o sistema em questão é consideravelmente menor em escala em comparação com o sistema da empresa compradora. Isso se traduz em uma menor complexidade e um escopo de projeto mais gerenciável, tornando uma implementação mais eficiente.

Além disso, uma característica notável da arquitetura *Umbrella* é a sua capacidade de segmentação e isolamento de funcionalidades em subprojetos independentes. Isso proporciona uma maneira mais eficaz de introduzir novas integrações ou funcionalidades sem o risco de impactar negativamente as partes já estabelecidas do sistema.

Isso é particularmente valioso quando se considera a importância de manter a estabilidade e o desempenho de um sistema em produção, como é o caso da empresa compradora.

A semelhança com a arquitetura de microsserviços é considerável. Ambas as metodologias buscam proporcionar uma estrutura organizacional eficiente e escalável para projetos de software, facilitando sua manutenção, evolução e adaptação às necessidades do ambiente tecnológico. Contudo, há diferenças estruturais consideráveis, que facilitam a implementação de uma solução:

- **Granularidade:** A arquitetura *Umbrella* agrupa diferentes partes do sistema dentro de um único projeto. Pode incluir várias aplicações, mas todas estão contidas dentro da estrutura principal. No caso de uma estrutura de microsserviços, o sistema é dividido em componentes independentes, cada um sendo uma aplicação separada e autônoma.

- **Dependências e Acoplamento:** A arquitetura *Umbrella* pode haver maior acoplamento entre as partes, pois estão dentro do mesmo projeto e podem compartilhar código de forma mais direta. No caso de uma estrutura de microsserviços, os serviços são completamente independentes, dificultando um potencial acoplamento entre eles.

- **Escalabilidade:** Na arquitetura *Umbrella*, a escalabilidade geralmente é feita em uma escala maior, englobando todo o projeto, o que pode resultar em escalabilidade vertical. No caso de uma estrutura de microsserviços, cada serviço pode ser escalado de forma independente, permitindo a escalabilidade horizontal, o que é mais eficaz para lidar com cargas específicas.

- **Implantação:** A arquitetura *Umbrella* é implantada como um único aplicativo, facilitando a implantação e o gerenciamento. No caso de uma estrutura de microsserviços, é necessário uma estratégia de implantação mais complexa, pois cada microsserviço é implantado e gerenciado separadamente.

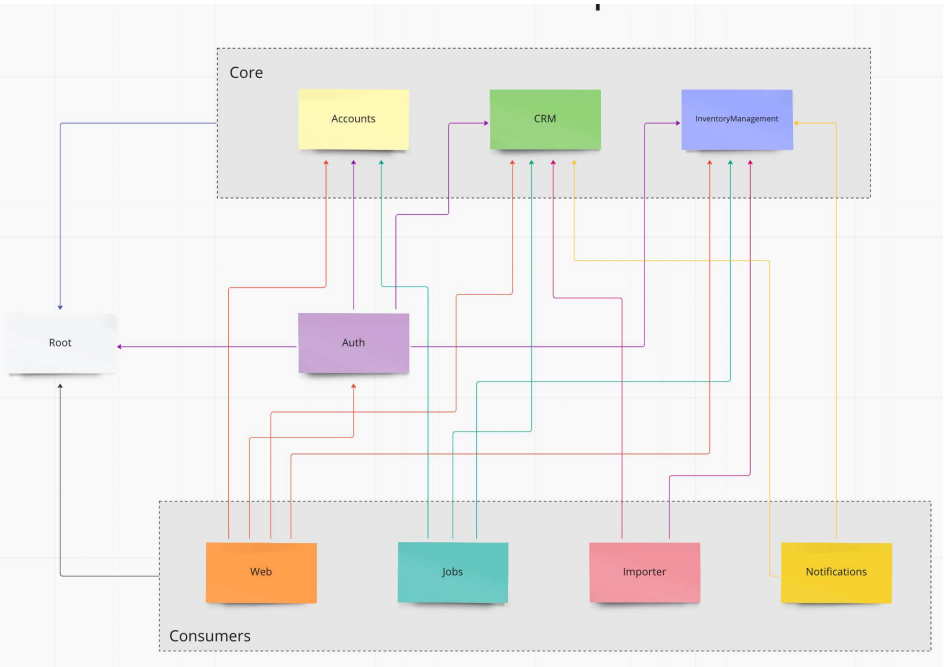
- **Manutenção:** A arquitetura *Umbrella* permite uma manutenção mais fácil, pois todos os componentes estão dentro do mesmo projeto e podem compartilhar recursos e código de forma mais direta. No caso de uma estrutura de microsserviços, a manutenção pode ser mais desafiadora devido à sua independência e à necessidade de gerenciar diferentes versões, atualizações e integrações entre serviços.

Deste modo, tendo o algoritmo centrado no lado da *startup*, o desenvolvimento de uma solução terá como resultado uma implementação rápida e eficiente.

4.3 ARQUITETURA DA *STARTUP*

A arquitetura de *Umbrella* da *startup* pode ser vista na Figura 4 a seguir.

Figura 4 – Arquitetura Umbrella da startup.



Fonte: Arquivo pessoal.

Como pode ser observado, a arquitetura possui os seguintes serviços:

- ***Root:*** Módulo central desempenhando um papel crucial ao abrigar configurações de amplo alcance que impactam todos os outros módulos. Um exemplo notável é a configuração da conexão com o banco de dados, vital para o funcionamento harmônico do sistema como um todo.
- ***Accounts:*** O módulo desempenha um papel fundamental ao cuidar do armazenamento e gerenciamento das informações elementares dos usuários.
- ***CRM:*** Módulo responsável por armazenar e proteger dados altamente sensíveis dos fornecedores. Isso inclui informações cruciais, como as assinaturas de contrato e detalhes relacionados aos orçamentos apresentados.
- ***Auth:*** Módulo responsável por processar e fazer uso das informações advindas de outros serviços dentro do sistema. Sua tarefa central envolve a análise das permissões do usuário, a verificação e validação das credenciais, garantindo que o acesso a esses dados esteja de acordo com as permissões estabelecidas para o perfil do usuário em questão..

- ***Notifications:*** Módulo responsável por desempenhar uma função essencial no âmbito da comunicação entre a plataforma e seus fornecedores e clientes. Este componente é responsável por gerenciar e orquestrar o processo de envio de mensagens informativas, alertas e atualizações relevantes para ambas as partes, estabelecendo um canal de comunicação eficaz e crucial para o bom funcionamento e interação no contexto da plataforma.

- ***Jobs:*** Módulo responsável por executar e supervisionar processos internos de forma periódica e automatizada. Sua função central abrange a coordenação e a execução regular de operações e tarefas específicas, essenciais para o funcionamento contínuo e eficiente do sistema, garantindo a integridade e o desempenho adequado das funcionalidades internas.

- ***Web:*** Módulo responsável por facilitar e viabilizar a conexão com uma variedade de serviços externos, sendo um elo fundamental para a interoperabilidade do sistema. Nesse contexto, ele possibilita a integração com diversos sistemas, incluindo, por exemplo, a aplicação *Frontend* da *startup*. Cabe ressaltar que essa aplicação possui sua própria arquitetura e funcionalidades, no entanto, é importante mencionar que detalhes específicos sobre essa arquitetura não serão abordados no escopo deste projeto, mantendo o foco na tarefa estabelecida.

- ***PubSub:*** Módulo facilitador de extrema relevância, atuando como uma ponte que possibilita uma comunicação excepcional e necessária entre dois serviços que, em princípio, não foram inicialmente projetados para se comunicar de forma direta. Essa funcionalidade se torna vital para promover a integração e a cooperação entre esses serviços que compartilham um nível hierárquico similar, contribuindo para a eficiência e o correto funcionamento do sistema como um todo.

- ***Importer:*** Módulo que possibilita a importação simultânea de um grande volume de ofertas, podendo chegar a dezenas delas, por meio de um arquivo com formato *.csv*. Essa capacidade de importação em massa por meio deste tipo de arquivo representa uma ferramenta eficaz e eficiente para o gerenciamento de um elevado número de ofertas, otimizando o processo e economizando tempo para os fornecedores.

- ***InventoryManagement:*** É o módulo de maior complexidade da plataforma e que concentra uma alta densidade das operações, sendo o local de armazenamento primário para os dados relativos a todas as ofertas e veículos disponíveis na plataforma. Sua função central abarca a gestão, organização e preservação das informações cruciais referentes a cada oferta e veículo, garantindo um repositório centralizado e estruturado.

4.4 PLANEJAMENTO ESTRATÉGICO

A fim de cumprir os objetivos propostos, e levando em consideração todos os aspectos já mencionados, é possível conceber uma solução estratégica dividida em três fases distintas. Cada uma destas fases terá a sua própria combinação de *sprints*, tendo em cada uma, tarefas com objetivos menores dentro do contexto de cada fase.

A primeira etapa se baseia no processo de mapear as informações de ambas as plataformas, para buscar uma conexão entre as informações das ofertas das duas plataformas. Também é esperado que nessa etapa sejam identificadas informações que estejam disponíveis apenas em um lado da migração. Nesse contexto, é necessário descrever o grau de importância da falta dessas potenciais informações, para definir as alterações futuras.

A segunda etapa, projetada para uma implementação a curto prazo, centraliza-se nas modificações a serem realizadas no âmbito da *startup*. O foco dessa fase inicial é capacitar a *startup* para efetuar as alterações necessárias a fim de transmitir as ofertas do tipo *leasing* para a empresa compradora. A empresa compradora, nesse estágio inicial, assume a responsabilidade de oferecer suporte somente para as informações essenciais relacionadas a essas ofertas.

A terceira etapa, por sua vez, concentra-se primordialmente no desenvolvimento de um plano estratégico para a migração de funcionalidades que estão apenas presentes no âmbito da *startup*. O objetivo é preparar ambas as plataformas para uma possível desativação do *Backend* da *startup*.

4.4.1 Primeira etapa

O mapeamento de campos durante uma migração de dados é um procedimento essencial para assegurar a transição precisa e coerente de informações entre diferentes sistemas ou bases de dados. Este processo envolve uma análise abrangente dos campos na fonte e a correspondência correspondente nos destinos, considerando divergências potenciais em estruturas, tipos de dados e formatos.

É possível separar esta fase em duas *sprints*, sendo o objetivo da primeira realizar uma pesquisa e uma análise detalhada da estrutura dos dados na origem e no destino, compreendendo aspectos como tipos de dados, tamanhos de campo e formatos de data. Identificar campos sem correspondência direta entre os sistemas é crucial nesse estágio.

A segunda *sprint* tem o objetivo de um dicionário de dados abrangente que enumere todos os campos tanto na origem quanto no destino, fornecendo informações detalhadas sobre cada campo, incluindo tipo de dado, tamanho e quaisquer restrições associadas.

A atribuição cuidadosa de tipos de dados e formatos é uma etapa crítica para

evitar perda de dados ou erros de conversão. Além disso, é importante identificar as diferenças estruturais entre os campos nos sistemas de origem e destino, pois os mesmos podem indicar a necessidade de transformações de dados.

4.4.2 Segunda etapa

Como mencionado previamente, a empresa compradora enfrenta o desafio de administrar uma extensa gama de microsserviços, o que torna a introdução de alterações em seus sistemas internos um processo intrincado e demorado. Nesse contexto, uma abordagem inicial e eficiente é concentrar a maior parte das modificações no âmbito da *startup*, reduzindo as alterações necessárias no lado da empresa compradora.

Neste contexto, é possível também separar esta fase em duas *sprints* distintas. A primeira tem o objetivo de implementar um programa que permita a migração dos dados de maneira contínua.

Enquanto isso, a segunda *sprint* tem o objetivo de analisar os resultados e identificar as possíveis falhas de mapeamento que possam ocorrer. Identificada as causas dos erros, é esperado que sejam apresentados soluções que visem o máximo de grau de sucesso de importação.

4.4.3 Terceira etapa

A terceira fase do desenvolvimento concentra-se majoritariamente na análise das funcionalidades na *startup* que não estão presentes na empresa compradora, e posteriormente, na criação de um planejamento estratégico inicial que pode ser usado em uma eventual migração de funcionalidades. Por se tratarem de duas etapas distintas, é possível separar ambas em duas *sprints* distintas.

Para esta análise, a metodologia *MosCoW* poderá ser utilizada, para elencar a prioridade das funcionalidades e montar um plano de estratégia para implementação.

Pelo fato de serem apenas consideradas funcionalidades presentes no painel de controle da plataforma, a principal argumentação que deve ser utilizada é o impacto da funcionalidade na realidade dos fornecedores. A fim de definir o impacto de uma funcionalidade, é importante também apresentar dados provenientes da *startup* para quantificar o quanto os fornecedores usam a funcionalidade em questão.

4.5 ARQUITETURA PROPOSTA

Uma solução viável para essa situação envolve a utilização de agendadores de tarefas, comumente conhecidos como *Cronjobs*. Essa estratégia essencialmente compreende tarefas programadas para serem executadas em horários predeterminados ou em intervalos regulares. No contexto da migração em questão, configura-se

um *Cronjob* para automatizar a transferência de dados da *startup* para o sistema da empresa compradora de forma regular e periódica.

O conceito subjacente a essa abordagem é permitir que, a cada hora, todas as ofertas sejam encaminhadas para a plataforma da empresa compradora. A empresa compradora, por sua vez, é responsável por criar ou atualizar essas informações em seu banco de dados correspondente. Com o intuito de minimizar o número de modificações necessárias no lado da empresa compradora, é esperado que os dados sejam remetidos no formato já compatível com o banco de dados de destino, que, como mencionado previamente, é do tipo *NoSQL*.

Para realizar uma ação desta forma, será preciso uma arquitetura que incorpore três módulos da atual arquitetura *Umbrella*: o *CRM*, o *InventoryManagement* e o *Jobs*.

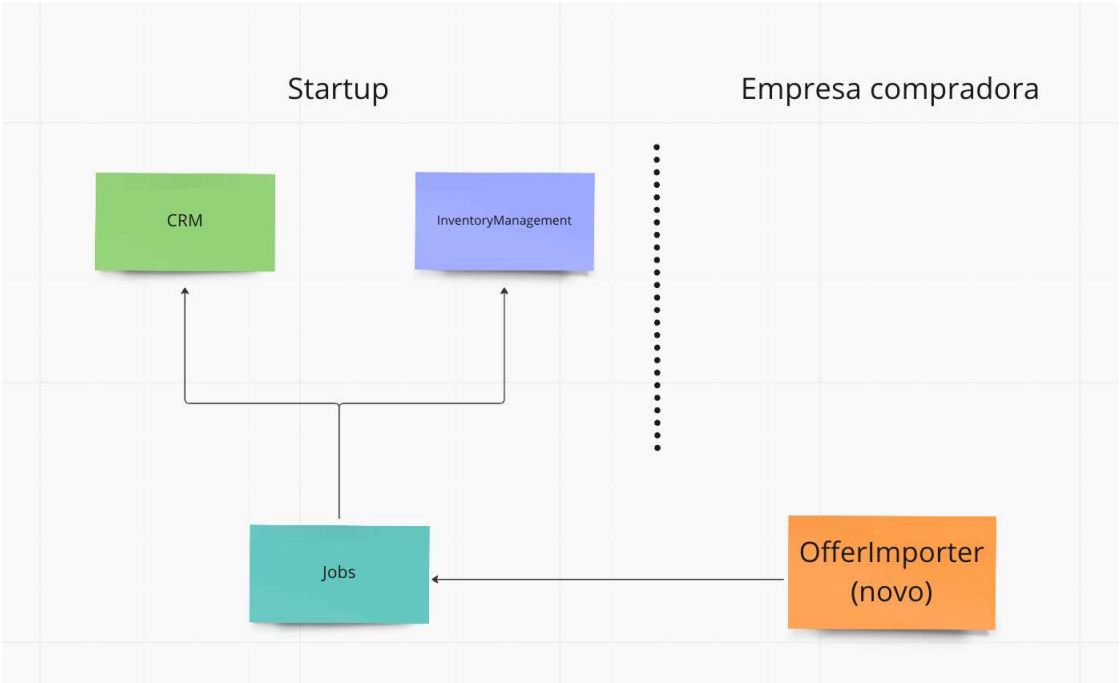
A conexão com o *CRM* é necessária para listar os fornecedores ativos da plataforma, a fim de realizar a migração das ofertas apenas destas entidades. Enquanto isso, o *InventoryManagement* será o responsável por buscar as ofertas correspondentes, transformá-las em um formato compatível com os dados da empresa receptora, e por fim, construir um arquivo *CSV* com tais informações.

A escolha da utilização de um arquivo *CSV* como método de envio dessas informações se deve ao fato de que o mesmo é capaz de suportar um volume significativo de dados a serem enviados de uma só vez. Além disso, a seleção do formato *CSV* nessa instância também visa simplificar a integração do sistema receptor no âmbito da empresa compradora, dado que a empresa mantém várias integrações externas que empregam o formato *CSV* como meio de comunicação padrão. Essa abordagem agiliza o processo de adaptação do sistema na empresa compradora, preservando a compatibilidade com os requisitos existentes.

Outro módulo importante, que deverá ser desenvolvido no âmbito da empresa receptora, é o serviço chamado *OffersImporter*. O seu objetivo é receber o arquivo *CSV*, e criar as ofertas correspondentes no banco de dados *NoSQL* presente na empresa compradora.

Um diagrama de alto nível com a arquitetura da solução proposta pode ser observado na Figura 5 a seguir.

Figura 5 – Diagrama de alto nível da arquitetura proposta.



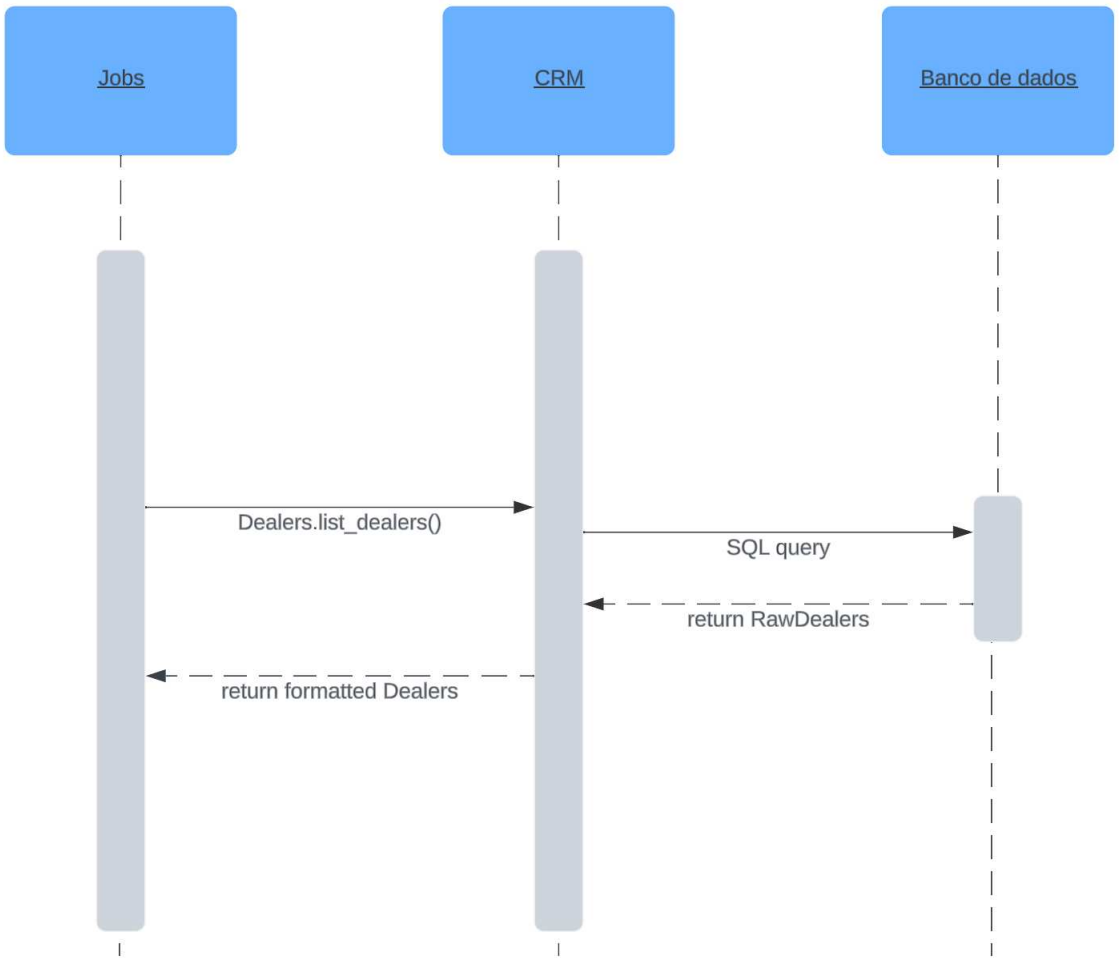
Fonte: Arquivo pessoal.

4.5.1 Arquitetura do módulo *CRM*

O módulo *CRM* será o responsável por retornar as informações dos fornecedores ativos da plataforma. Para tal, ele receberá um pedido do serviço *Jobs*, que retornará estas informações. É importante ressaltar a necessidade de uma etapa de transformar os dados provenientes do banco de dados, a fim de retornar os dados dos fornecedores de maneira estruturada e clara.

No diagrama *UML* de sequência presente na Figura 6 a seguir, é possível observar com mais detalhes as etapas deste processo.

Figura 6 – Diagrama de sequência do módulo *CRM*.



Fonte: Arquivo pessoal.

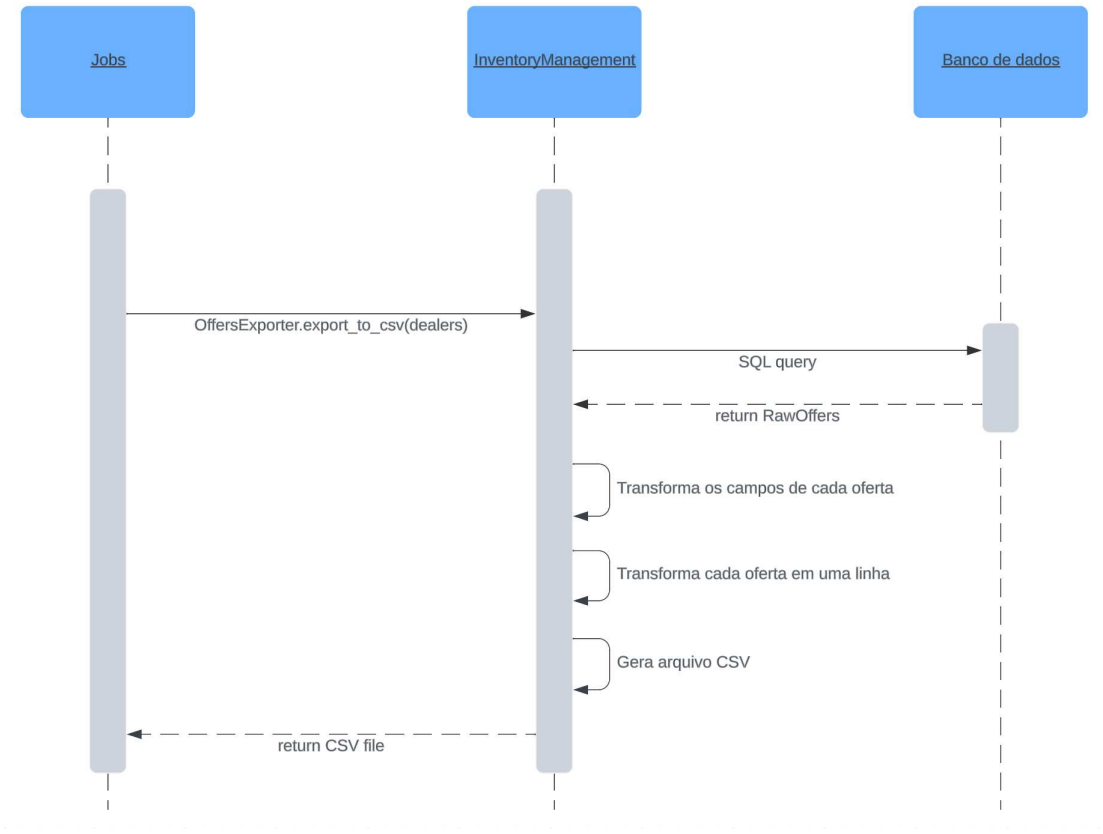
4.5.2 Arquitetura do módulo *InventoryManagement*

O módulo *CRM* será o responsável por receber as informações dos fornecedores, e retornar um arquivo *CSV* com as informações das ofertas correspondentes. Para tal, ele receberá um pedido do serviço *Jobs*, contendo uma lista de fornecedores.

Internamente, o *InventoryManagement* deverá retornar todas as ofertas correspondentes dos fornecedores recebidos e transformá-las em ofertas compatíveis com o banco de dados receptor. Após esta etapa, é preciso criar um arquivo *CSV* no qual cada linha corresponda com uma oferta transformada. Por fim, o sistema retorna estas informações para o serviço *Jobs*, que requisitou as informações inicialmente.

No diagrama *UML* de sequência presente na Figura 7 a seguir, é possível observar com mais detalhes as etapas deste processo.

Figura 7 – Diagrama de sequência do módulo *InventoryManagement*.



Fonte: Arquivo pessoal.

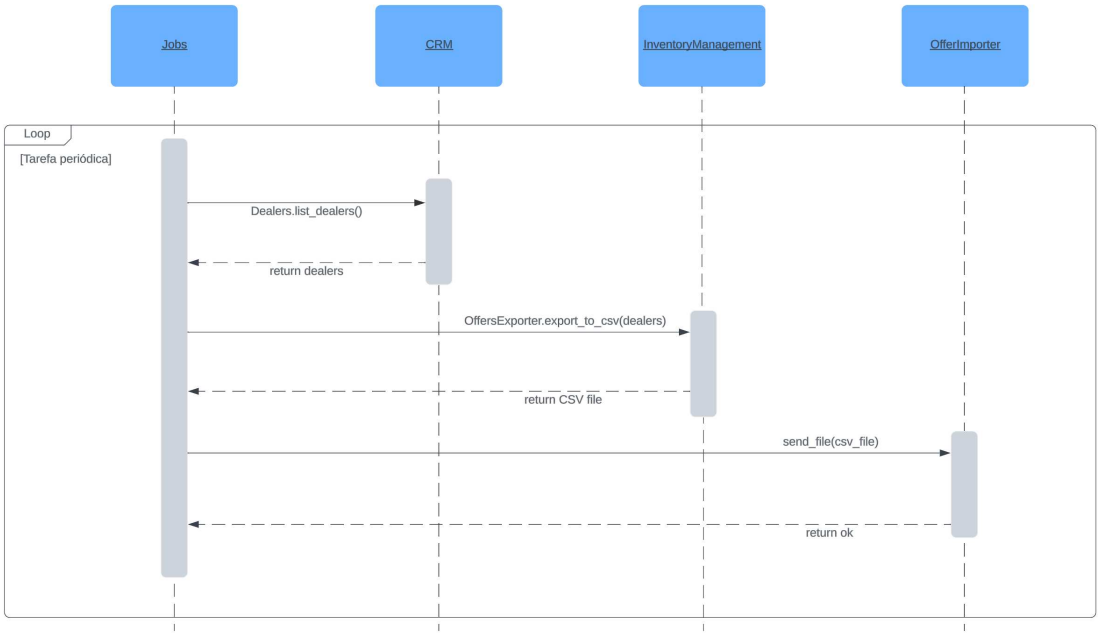
4.5.3 Arquitetura do módulo *Jobs*

Como já comentado, o módulo *Jobs* requisitará para o módulo *CRM* os dados dos fornecedores, e enviará estas informações para o módulo *InventoryManagament*, a fim de obter um arquivo *CSV* válido com os dados das ofertas. O módulo *Jobs* também será o responsável por enviar estas informações para o serviço *OfferImporter* presente no sistema da empresa receptora.

Além disso, o módulo em questão também é responsável para que este processo aconteça de maneira periódica por meio de *Cronjobs*. É importante que o mesmo forneça suporte para monitoramento e seja capaz de lidar com falhas de maneira eficiente.

No diagrama *UML* de sequência presente na Figura 8 a seguir, é possível observar com mais detalhes como o módulo em questão deve funcionar.

Figura 8 – Diagrama de sequência do módulo *Jobs*.



Fonte: Arquivo pessoal.

5 IMPLEMENTAÇÃO

Neste capítulo será descrita a implementação da solução proposta. Como já comentado, a solução se concentra em duas etapas, sendo a primeira tendo um foco maior de implementação na *startup*, e a segunda com um foco na empresa compradora. Neste capítulo, especificamente, será tratado sobre a primeira etapa.

5.1 PRIMEIRA ETAPA

A primeira etapa para a implementação envolve o mapeamento de campos das plataformas. A intenção desta etapa é unificar a definição de oferta em ambas as plataformas. Para isso, foi necessário buscar todas as informações que a empresa compradora utiliza para definição de suas ofertas. Após esta etapa, todos os campos foram mapeados para o seu respectivo campo na *startup*.

5.1.1 Dados na empresa compradora

Como já mencionado, a empresa compradora utiliza um banco de dados não relacional, e por tal motivo, foi possível observar o agrupamento de certos parâmetros em grupos específicos. Tais grupos podem ser observados abaixo:

- ***Root***: informações básicas de uma oferta.
- ***Price***: informações de valores de venda para a oferta.
- ***Attributes***: informações técnicas a respeito do veículo que a oferta é conectada.
- ***Primary Contact***: informações a respeito do fornecedor.
- ***Rates***: informações sobre os possíveis financiamentos de leasing para a respectiva oferta.
- ***Location***: informações a respeito da localização do veículo. A princípio, esta informação é correlacionada com a informação de localização do fornecedor.
- ***Fuel Emission***: informações a respeito do consumo de combustível do veículo.

Os campos para cada grupo serão descritos nos tópicos a seguir.

5.1.1.1 Root

Nome	**Tipo**	**Descrição**
title	String	Título da oferta
dealerId	String	Identificador único do fornecedor
images	Lista de strings	Imagens do anúncio da oferta
comment	Texto	Descrição da oferta
lender	String	Nome do banco para o qual a oferta é financiada
grossListPrice	Decimal	Valor bruto de venda do veículo
category	Enum	Categoria do veículo
make	String	Nome da marca/montadora do veículo
model	String	Nome do modelo do veículo
condition	Booleano	Nome do banco para o qual a oferta é financiada

Tabela 1 – Lista de campos correspondentes ao módulo *Root*.

5.1.1.2 Price

Nome	**Tipo**	**Descrição**
vat	Decimal	Valor do imposto sobre o valor agregado
type	Enum	Tipo de cálculo de juros
consumerValue	Decimal	Valor de venda do veículo para o consumidor

Tabela 2 – Lista de campos correspondentes ao módulo *Price*.

5.1.1.3 Attributes

Nome	**Tipo**	**Descrição**
usageType	Enum	Tipo de contrato de uso do veículo
mileage	Inteiro	Quantidade de quilômetros já rodados (para veículos usados)
firstRegistration	Data	Momento que o veículo foi oficializado pela montadora
vin	String	Identificador universal do veículo
hsn	String	Identificador dado à montadora para produção do veículo
tsn	String	Código do veículo com as informações de motor e outras informações
power	Inteiro	Potência do motor
gearbox	Enum	Tipo de motor
seats	Inteiro	Quantidade de assentos
doors	Inteiro	Quantidade de portas
fuel	Enum	Tipo de combustível
exteriorColor	String	Cor externa
InteriorColor	String	Cor interna
colorName	String	Nome da cor pela montadora
deliveryPeriod	Inteiro	Quantidade de dias para o carro ser produzido
cubicCapacity	Inteiro	Capacidade cúbica dos cilindros do motor

Tabela 3 – Lista de campos correspondentes ao módulo *Attributes*.

5.1.1.4 Primary Contact

Nome	**Tipo**	**Descrição**
companyName	String	Nome do fornecedor

Tabela 4 – Lista de campos correspondentes ao módulo *Primary Contact*.

5.1.1.5 Rates

Nome	**Tipo**	**Descrição**
type	Enum	Tipo de consumidor
downpayment	Decimal	Valor de depósito inicial
netLoanAmount	Decimal	Valor do montante de empréstimo
maintenanceAndWear	Decimal	Valor de manutenção e desgaste
nominalInterestRate	Decimal	Valor da taxa de juros nominal
netLoanAmount	Decimal	Valor do montante de empréstimo
annualPercentageRates	Decimal	Valor percentual da taxa de juros anual
destinationCharges	Decimal	Valor para envio
termOfContract	Inteiro	Quantidade de meses para o contrato
annualMileage	Inteiro	Quantidade de quilômetros por ano para o contrato
netRate	Decimal	Valor do contrato

Tabela 5 – Lista de campos correspondentes ao módulo *Rates*.

5.1.1.6 Location

Nome	**Tipo**	**Descrição**
city	String	Nome da cidade
street	String	Nome da rua
zip	String	Código de endereçamento postal

Tabela 6 – Lista de campos correspondentes ao módulo *Location*.

5.1.1.7 Fuel Emission

Nome	**Tipo**	**Descrição**
wltp.consumptionCombined	Decimal	Valor de consumo combinado no padrão WLTP
wltp.co2	Decimal	Valor de emissão de CO2 no padrão WLTP
wltp.electricalRange	Decimal	Quantidade de quilômetros por carga de bateria no padrão WLTP
consumptionCombined	Decimal	Valor de consumo combinado no padrão NEFZ
consumptionCombinedInner	Decimal	Valor de consumo na cidade no padrão NEFZ
consumptionCombinedOuter	Decimal	Valor de consumo fora da cidade no padrão NEFZ
combinedPowerConsumption	Decimal	Valor de consumo elétrico combinado no padrão NEFZ
co2	Decimal	Valor de emissão de CO2 no padrão NEFZ
energyEfficiencyClass	String	Classe de eficiência energética do veículo

Tabela 7 – Lista de campos correspondentes ao módulo *Fuel Emission*.

5.1.2 Dados na startup

Na *startup*, pelo fato do banco de dados ser do tipo relacional, estas informações foram separadas em uma tabela principal, e em tabelas secundárias, que possuem uma chave primária ligada à tabela principal.

As tabelas com os dados podem ser observadas abaixo.

- ***Offers***: informações básicas de uma oferta.
- ***Vehicles***: informações técnicas a respeito do veículo que a oferta é conectada.
- ***Brands***: informações da montadora do veículo.
- ***Delivery Prices***: informações a respeito dos custos para envio do veículo ao consumidor final.
- ***Dealers***: informações a respeito dos fornecedores.
- ***Colors***: informações a respeito das possíveis cores para o veículo da oferta.
- ***Fixed Leasing Rates***: informações a respeito dos possíveis financiamentos de leasing para a respectiva oferta.
- ***Special Conditions***: informações a respeito de condições especiais da oferta.
- ***Banks***: informações a respeito do banco financiador da oferta.

Os campos para cada grupo serão descritos nos tópicos a seguir.

5.1.2.1 Offers

Nome	**Tipo**	**Descrição**
id	Inteiro	Identificador único da oferta
title	String	Título da oferta
images	Lista de strings	Imagens do anúncio da oferta
description	Texto	Descrição da oferta
category	Enum	Categoria do veículo da oferta
vehicle_type	Enum	Estado do veículo da oferta
one_day_registration	Booleano	Oferta de registro de dia único
driven_km	Inteiro	Quantia de quilômetros já rodados do veículo
availability	Inteiro	Tempo para o veículo estar disponível
customer_type	Enum	Tipo de consumidor da oferta
down_payment	Decimal	Valor inicial de depósito. Padrão como zero.
loan_amount_net	Decimal	Valor do montante de empréstimo
maintenance_wear	Decimal	Valor de manutenção e desgaste
nominal_interest_rate	Decimal	Valor da taxa de juros nominal
effective_annual_interest_rate	Decimal	Valor efetivo da taxa de juros anual
annual_percentage_rates	Decimal	Valor percentual da taxa de juros anual
vat_rate	Decimal	Valor do imposto sobre o valor agregado
interest_rate_type	Enum	Tipo de cálculo de juros
current_price_net	Decimal	Valor de venda do veículo para o consumidor
wltp_combined	Decimal	Valor de consumo combinado no padrão WLTP
wltp_carbonic_emission	Decimal	Valor de emissão de CO2 no padrão WLTP
battery_range_km	Decimal	Quantidade de quilômetros por carga de bateria no padrão WLTP

fuel_consumption_combined	Decimal	Valor de consumo combinado no padrão NEFZ
fuel_consumption_city	Decimal	Valor de consumo na cidade no padrão NEFZ
fuel_consumption_highway	Decimal	Valor de consumo fora da cidade no padrão NEFZ
power_consumption_combined	Decimal	Valor de consumo elétrico combinado no padrão NEFZ
carbonic_emission	Decimal	Valor de emissão de CO2 no padrão NEFZ
energy_efficiency_class	String	Classe de eficiência energética do veículo
pollutant_class	String	Classe de poluição do veículo no padrão NEFZ
pollutant_class_detailed	String	Descrição da classe de poluição do veículo no padrão NEFZ
wltp_pollutant_class	String	Classe de poluição do veículo no padrão WLTP
wltp_pollutant_class_detailed	String	Descrição da classe de poluição do veículo no padrão WLTP
color_name_manufacturer	String	Nome da cor do veículo fornecido pela montadora
vehicle_id	Chave estrangeira	Identificador do veículo na qual a oferta está conectada
bank_id	Chave estrangeira	Identificador do banco na qual a oferta está conectada
brand_id	Chave estrangeira	Identificador da montadora do veículo na qual a oferta está conectada
exterior_base_color_id	Chave estrangeira	Identificador da cor externa do veículo
interior_base_color_id	Chave estrangeira	Identificador da cor interna do veículo
dealer_id	Chave estrangeira	Identificador do fornecedor no qual a oferta está conectada

Tabela 8 – Lista de campos correspondentes ao módulo *Offers*.

5.1.2.2 Vehicles

Nome	**Tipo**	**Descrição**
id	Inteiro	Identificador único do veículo
base_list_price_gross	Decimal	Valor bruto de venda do veículo
model_group	String	Modelo do veículo
registered_at	Data	Momento que o veículo foi oficializado pela montadora
vin	String	Identificador universal do veículo
hsn	String	Identificador dado à montadora para produção do veículo
tsn	String	Código do veículo com as informações de motor e outras informações
power_kw	Inteiro	Potência do motor
gear_type	Enum	Tipo de motor
number_of_seats	Inteiro	Quantidade de assentos
number_of_doors	Inteiro	Quantidade de portas
fuel_type	Enum	Tipo de combustível
engine_displacement_ccm	Inteiro	Capacidade cúbica dos cilindros do motor

Tabela 9 – Lista de campos correspondentes ao módulo *Offers*.

5.1.2.3 Brands

Nome	**Tipo**	**Descrição**
id	Inteiro	Identificador único da montadora
name	String	Nome da montadora

Tabela 10 – Lista de campos correspondentes ao módulo *Brands*.

5.1.2.4 Delivery Prices

Nome	**Tipo**	**Descrição**
id	Inteiro	Identificador único do método de envio
dealer_pickup_price	Decimal	Valor para ser retirado na sede do fornecedor
factory_pickup_price	Decimal	Valor para ser retirado na montadora
house_delivery_price	Decimal	Valor para ser recebido no endereço do consumidor
customer_type	Enum	Tipo de consumidor
offer_id	Chave estrangeira	Identificador da oferta na qual o método está conectado

Tabela 11 – Lista de campos correspondentes ao módulo *Delivery Prices*.

5.1.2.5 Dealers

Nome	**Tipo**	**Descrição**
id	Inteiro	Identificador único do fornecedor
name	String	Nome do fornecedor
city	String	Cidade do fornecedor
street	String	Rua do fornecedor
postal_code	String	Endereço postal do fornecedor

Tabela 12 – Lista de campos correspondentes ao módulo *Dealers*.

5.1.2.6 Colors

Nome	**Tipo**	**Descrição**
id	Inteiro	Identificador único da cor
code	String	Código da cor
name	String	Nome da cor
type	Enum	Tipo da cor (interno ou externo)

Tabela 13 – Lista de campos correspondentes ao módulo *Colors*.

5.1.2.7 Fixed Leasing Rates

Nome	**Tipo**	**Descrição**
id	Inteiro	Identificador único da taxa de leasing
term_months	Inteiro	Quantidade de meses
price_net	Decimal	Preço
mileage_km	Integer	Quantidade de quilômetros
offer_id	Chave estrangeira	Identificador da oferta

Tabela 14 – Lista de campos correspondentes ao módulo *Fixed Leasing Rates*.

5.1.2.8 Special Conditions

Nome	**Tipo**	**Descrição**
id	Inteiro	Identificador único da condição especial
name	String	Nome da condição especial
description	String	Descrição da condição especial
offer_id	Chave estrangeira	Identificador único da oferta

Tabela 15 – Lista de campos correspondentes ao módulo *Special Conditions*.

5.1.2.9 Banks

Nome	**Tipo**	**Descrição**
id	Inteiro	Identificador único do banco
name	String	Nome do banco

Tabela 16 – Lista de campos correspondentes ao módulo *Banks*.

5.1.3 Mapeamento dos dados

Com os campos definidos em ambas as empresas, é possível criar uma relação que conecte diretamente os campos de cada uma. Uma tabela com estas informações pode ser vista abaixo:

Campo na startup	**Campo na empresa compradora**
title	*title*
dealer.id	*dealerId*
images	*images*
description	*comment*
bank.name	*lender*
vehicle.base_list_price_gross	*grossListPrice*
category	*category*
brand.name	*make*
vehicle.model_group	*model*
vehicle_type	*condition*
one_day_registration	*attributes.usageType*
vehicle.driven_km	*attributes.mileage*
vehicle.registered_at	*attributes.firstRegistration*
vehicle.vin	*attributes.vin*
vehicle.hsn	*attributes.hsn*
vehicle.tsn	*attributes.tsn*
vehicle.power_kw	*attributes.power*
vehicle.gear_type	*attributes.gearbox*
vehicle.number_of_seats	*attributes.seats*
vehicle.number_of_doors	*attributes.doors*
vehicle.fuel_type	*attributes.fuel*
exterior_base_color.name	*attributes.exteriorColor*
interior_base_color.name	*attributes.interiorColor*
color_name_manufacturer	*attributes.colorName*
availability	*attributes.availability*
vehicle.engine_displacement_ccm	*attributes.cubicCapacity*
dealer.city	*location.city*
dealer.street	*location.street*
dealer.postal_code	*attributes.zip*
dealer.name	*primaryContact.companyName*
customer_type	*rates.type*

down_payment	*rates.downpayment*
loan_amount_net	*rates.netLoanAmount*
maintenance_wear	*rates.maintenanceAndWear*
nominal_interest_rate	*rates.nominalInterestRate*
effective_annual_interest$_r$ate	*rates.annualPercentualRates*
delivery_price.dealer_pickup_price	*rates.destinationCharges*
annual_percentage_rate	*rates.annualPercentageRates*
fixed_leasing_rate.term_months	*rates.termOfContract*
fixed_leasing_rate.mileage_km	*rates.annualMileage*
fixed_leasing_rate.price_net	*rates.netRate*
vat_rate	*price.vat*
interest_rate_type	*price.type*
current_price_net	*price.consumerValue*
wltp_combined	*fuelEmission.wltp.consumptionCombined*
wltp_carbonic_emission	*fuelEmission.wltp.co2*
battery_range_km	*fuelEmission.wltp.electricalRange*
fuel_consumption_combined	*fuelEmission.consumptionCombined*
fuel_consumption_city	*fuelEmission.consumptionInner*
fuel_consumption_highway	*fuelEmission.consumptionOuter*
power_consumption_combined	*fuelEmission.combinedPowerConsumption*
carbonic_emission	*fuelEmission.co2*
energy_efficiency_class	*fuelEmission.energyEfficiencyClass*

Tabela 17 – Mapeamento dos campos nas duas plataformas.

5.1.4 Diferença dos requisitos

Durante o processo de mapeamento, notou-se algumas diferenças fundamentais entre as características dos campos para as duas empresas. A primeira diferença é que a *startup* considera apenas os campos presentes na tabela *Fixed Leasing Rates* como customizáveis para cada possibilidade de contrato. A empresa compradora, por outro lado, permite a customização de todos os campos presentes no módulo "rates"para cada tipo de contrato. Na *startup*, a grande maioria destes campos são definidos a nível de oferta. Este não é um limitante para o processo de migração em si, visto que é possível replicar os dados da oferta para cada módulo de "rates". Contudo, é possível caracterizar esta questão como um limitante da *startup* para com os fornecedores.

É possível perceber também que a empresa compradora não separa as informações de custo de entrega do veículo. Enquanto na *startup* tais campos são separados nos diferentes tipos de entrega (no fornecedor, na montadora e no endereço do consumidor), a empresa compradora coloca um campo único de custo de envio. Para facilitar o futuro desenvolvimento desta funcionalidade na empresa compradora, estes campos também serão retornados nesta primeira etapa.

Os campos de classe de poluição também não são suportados pela empresa compradora. Contudo, foram realizadas algumas pesquisas e foi possível concluir que estas classes fazem parte de um antigo sistema de medição de qualidade, que já não é mais utilizado pela legislação da União Europeia. Deste modo, concluiu-se que, por hora, não há necessidade de suportar tais campos.

Por fim, é possível notar que não há suporte na empresa compradora para as condições especiais das ofertas. Seguindo a mesma lógica utilizada para os custos de envio, tais campos também serão adicionados para facilitar o futuro desenvolvimento da funcionalidade.

5.2 SEGUNDA ETAPA

A segunda etapa para a implementação envolve a implementação de um algoritmo capaz de migrar as ofertas. O algoritmo deve ser capaz de realizar a migração por meio de *CSV*s e *Cronjobs*.

5.2.1 Geração do arquivo CSV

Com o mapeamento dos campos já definidos, é possível desenvolver um código para geração do arquivo *CSV*. Primeiramente, foi criado um serviço chamado *OffersExporter* dentro do serviço *InventoryManagement* da *startup*. A primeira coisa definida no módulo é a sua descrição, as dependências que serão utilizadas posteriormente, e as colunas que estarão presentes no *CSV*. O objetivo nesta etapa é manter os campos o mais próximo possível da linguagem utilizada pela empresa compradora.

O código inicial pode ser visto a seguir.

```
defmodule InventoryManagement.Exporters.OffersExporter do
    @moduledoc """
    Exporter adapted to export Offers to an external database.
    """

    ...

    @headers ~w(
        title
        dealerId
        images
        comment
        lender
        grossListPrice
        category
        make
        model
        condition
        attributes_usageType
        attributes_mileage
        attributes_firstRegistration
        attributes_vin
        attributes_hsn
        attributes_tsn
        attributes_power
        attributes_gearbox
        attributes_seats
        attributes_doors
        attributes_fuel
        attributes_exteriorColor
        attributes_interiorColor
        attributes_exteriorColor
        attributes_colorName
        attributes_deliveryPeriod
        attributes_cubicCapacity
        location_city
        location_street
        location_zip
        primaryContact_companyName
        rates_type
        rates_downpayment
        rates_netLoanAmount
        rates_maintenanceAndWear
        rates_nominalInterestRate
        rates_annualPercentageRates
        rates_destinationCharges
        rates_annualPercentageRates
```

```
48          leasing_rates
49          price_vat
50          price_type
51          price_consumerValue
52          fuelEmission_wltp_consumptionCombined
53          fuelEmission_wltp_co2
54          fuelEmission_wltp_electricalRange
55          fuelEmission_consumptionCombined
56          fuelEmission_consumptionInner
57          fuelEmission_consumptionOuter
58          fuelEmission_combinedPowerConsumption
59          fuelEmission_co2
60          fuelEmission_energyEfficiencyClass
61          extra__factory_pickup_price
62          extra__house_delivery_price
63          extra__special_conditions
64      )a
65
66      ...
67
68  end
```

É válido destacar a escolha de unificar as informações customizáveis para cada contrato no lado da startup em uma única coluna, chamada de *leasing_rates*.

Com as colunas definidas, é possível descrever a função principal que será responsável pela migração das ofertas. Neste caso, ela será chamada de *export_to_csv*, e receberá uma lista de informações dos fornecedores. A função pode ser vista no código descrito abaixo. Importante ressaltar que o recebimento da informação dos fornecedores é necessário, visto que os serviços *InventoryManagement* e *CRM* não possuem uma ligação direta, e portanto não é possível buscar os fornecedores a partir de *InventoryManagement*. Contudo, o processo de obtenção destes fornecedores poderá ser realizado no serviço de *Jobs*, que será descrito mais adiante.

```
1  defmodule InventoryManagement.Exporters.OffersExporter do
2      @moduledoc """
3      Exporter adapted to export Offers to an external database.
4      """
5
6      ...
7
8      alias InventoryManagement.Offers
9
10     @type dealer :: %{
11       id: binary(),
12       name: String.t(),
```

```
13        street: String.t() | nil,
14        city: String.t() | nil,
15        postal_code: String.t() | nil
16      }
17
18      @spec export_to_csv(list(dealer)) :: binary
19      def export_to_csv(dealers) do
20          offers = list_offers()
21          dealers_by_id = Map.new(dealers, &{&1.id, &1})
22
23          ...
24      end
25
26      defp list_offers(filters) do
27          opts = [
28            assocs: [
29              :special_conditions,
30              :delivery_prices,
31              :fixed_leasing_rates,
32              :brand,
33              :bank,
34              :vehicle,
35              :exterior_base_color,
36              :interior_base_color,
37              :special_conditions
38            ],
39          ]
40
41          Offers.list_public_offers(opts)
42      end
43
44      ...
45
46  end
```

Nas linhas 10-18 há a definição dos parâmetros dos fornecedores que serão enviados para a função pública do módulo. Tais definições em funções públicas são consideradas uma boa prática para desenvolvimento de códigos em Elixir.

Na linha 21 há também uma reestruturação dos dados dos fornecedores. O propósito desta reestruturação é evitar uma complexidade do tipo O(n + 1) no decorrer do código. Desta forma, a informação será de fácil acesso, e será evitado uma busca complexa para cada uma das ofertas.

É possível também observar no código acima a presença de uma função privada responsável pela listagem das ofertas. O objetivo é listar as ofertas públicas da plataforma juntamente com as informações contidas nas tabelas já comentadas

anteriormente.

No código a seguir, são criadas algumas funções privadas de conversão de dados que serão utilizadas posteriormente para construir as linhas do *CSV*.

```
1  defmodule InventoryManagement.Exporters.OffersExporter do
2      @moduledoc """
3      Exporter adapted to export Offers to an external database.
4      """
5
6      ...
7
8      defp format_decimal(nil), do: nil
9      defp format_decimal(decimal), do: Decimal.to_string(decimal, ...
           :normal)
10
11     defp build_special_conditions(offer) do
12         special_conditions =
13             Enum.map(offer.special_conditions, fn special_condition ->
14                 %{name: special_condition.name, description: ...
                       special_condition.description}
15             end)
16
17         Jason.encode!(special_conditions)
18     end
19
20     defp build_leasing_rates(fixed_leasing_rates) do
21         fixed_leasing_rates
22         |> Enum.sort_by(fn leasing_rate -> leasing_rate.term_months end)
23         |> Enum.map_join("/", fn rate ->
24             price = format_decimal(rate.price_net)
25
26             "#{rate.term_months}-#{rate.mileage_km}=#{price}"
27         end)
28     end
29
30     ...
31
32 end
```

Nas linhas 8-9 é possível observar a presença de uma funcionalidade interessante do Elixir que é chamada de *Pattern Matching*. Quando a informação enviada para a função é um valor nulo, ela automaticamente retorna um valor nulo. Quando a informação recebida é um decimal, a mesma é enviada para uma função que a converte para texto.

Nas linhas 11-18, há a função responsável por converter as condições especi-

ais da oferta para ser inserida no *CSV*. Como esta é uma funcionalidade que não foi implementada ainda na empresa compradora, a solução encontrada para lidar com as múltiplas condições especiais que uma oferta pode ter, foi transformar as informações em uma informação do tipo *JSON*, que contenha a informação do nome e da descrição da condição especial. Na linha 17, este *JSON* é convertido para texto, para ser corretamente inserido em uma coluna do *CSV*.

Nas linhas 20-26, há a função responsável por converter as possibilidades de *leasing* em uma forma que o sistema da empresa compradora compreenda facilmente. Para isso, as possibilidades de contrato são ordenadas a partir da informação dos *term_months*, e convertidas para um código de texto. O formato do código de texto escolhido para esta conversão é baseado em um padrão utilizado internamente na empresa compradora, e também por fornecedores. O texto final possui um formato semelhante a este:

$$12\text{-}10000\text{=}123.45 \tag{1}$$

Sendo, neste caso, 12 a quantidade de meses no qual o contrato é estabelecido, 10000 a quantidade de quilômetros que poderão ser rodados, e 123.45 o valor desta possibilidade de contrato.

Após a definição destas funções, é criada uma função que converte cada oferta da plataforma em uma linha do *CSV*. Este código pode ser observado abaixo.

```
1  defmodule InventoryManagement.Exporters.OffersExporter do
2      @moduledoc """
3      Exporter adapted to export Offers to an external database.
4      """
5
6      ...
7
8      @spec export_to_csv(list(dealer)) :: binary
9      def export_to_csv(dealers) do
10         offers = list_offers()
11         dealers_by_id = Map.new(dealers, &{&1.id, &1})
12
13         offers
14         |> Stream.map(&build_data_for_row(&1, dealers_by_id))
15         ...
16     end
17
18     defp build_data_for_row(%Offer{} = offer, dealers_by_id) do
19         dealer = Map.fetch!(dealers_by_id, offer.dealer_id)
20
21         %{
```

```
22              title: offer.title,
23              dealerId: dealer.id,
24              images: offer.images,
25              comment: offer.description,
26              lender: offer.bank.name,
27              grossListPrice: ...
                    format_decimal(offer.vehicle.base_list_price_gross),
28              category: offer.category,
29              make: offer.brand.name,
30              model: offer.vehicle.model_group,
31              condition: offer.vehicle_type,
32              attributes_usageType: offer.one_day_registration,
33              attributes_mileage: offer.vehicle.driven_km,
34              attributes_firstRegistration: offer.vehicle.registered_at,
35              attributes_vin: offer.vehicle.vin,
36              attributes_hsn: offer.vehicle.hsn,
37              attributes_tsn: offer.vehicle.tsn,
38              attributes_power: offer.vehicle.power_kw,
39              attributes_gearbox: offer.vehicle.gear_type,
40              attributes_seats: offer.vehicle.number_of_seats,
41              attributes_doors: offer.vehicle.number_of_doors,
42              attributes_fuel: offer.vehicle.fuel_type,
43              attributes_exteriorColor: offer.exterior_base_color.name,
44              attributes_interiorColor offer.interior_base_color.name,
45              attributes_colorName: offer.color_name_manufacturer,
46              attributes_deliveryPeriod: offer.availability,
47              attributes_cubicCapacity: ...
                    format_decimal(offer.vehicle.engine_displacement_ccm),
48              location_city: offer.dealer.city,
49              location_street: offer.dealer.street,
50              location_zip: offer.dealer.zip_code,
51              primaryContact_companyName: offer.dealer.name,
52              rates_type: offer.customer_type,
53              rates_downpayment: format_decimal(offer.down_payment,
54              rates_netLoanAmount: format_decimal(offer.loan_amount_net),
55              rates_maintenanceAndWear: ...
                    format_decimal(offer.maintenance_wear),
56              rates_nominalInterestRate: ...
                    format_decimal(offer.nominal_interest_rate),
57              rates_annualPercentageRates: ...
                    format_decimal(offer.annual_percentage_rate),
58              rates_destinationCharges: ...
                    format_decimal(offer.delivery_price.dealer_pickup_price),
59              rates_annualPercentageRates: ...
                    format_decimal(offer.annual_percentage_rate),
60              leasing_rates: ...
                    build_leasing_rates(offer.fixed_leasing_rates),
```

```
61              price_vat: offer.vat_rate,
62              price_type: offer.interest_rate_type,
63              price_consumerValue: ...
                    format_decimal(offer.current_price_net),
64              fuelEmission_wltp_consumptionCombined: ...
                    format_decimal(offer.wltp_combined),
65              fuelEmission_wltp_co2: ...
                    format_decimal(offer.wltp_carbonic_emission),
66              fuelEmission_wltp_electricalRange: ...
                    format_decimal(offer.battery_range_km),
67              fuelEmission_consumptionCombined: ...
                    format_decimal(offer.fuel_consumption_combined),
68              fuelEmission_consumptionInner: ...
                    format_decimal(offer.fuel_consumption_city),
69              fuelEmission_consumptionOuter: ...
                    format_decimal(offer.fuel_consumption_highway),
70              fuelEmission_combinedPowerConsumption: ...
                    format_decimal(offer.power_consumption_combined),
71              fuelEmission_co2: format_decimal(offer.carbonic_emission),
72              fuelEmission_energyEfficiencyClass: ...
                    offer.energy_efficiency_class,
73              extra__factory_pickup_price: ...
                    format_decimal(offer.delivery_price.factory_pickup_price),
74              extra__house_delivery_price: ...
                    format_decimal(offer.delivery_price.house_pickup_price),
75              extra__special_conditions: ...
                    format_special_conditions(offer.special_conditions)
76          }
77      end
78
79      ...
80
81  end
```

É importante mencionar a utilização do método *Stream* na linha 14 para a construção de cada linha do *CSV*. Uma *Stream* é basicamente uma lista de operações envolvendo listas. Ela agrupa todas as listas e as operações só serão realizadas no final do processo. A partir deste agrupamento, o sistema se torna mais eficaz. Isso é importante, visto que um *CSV* pode ter milhares de linhas para serem processadas.

Por fim, há o código responsável por converter estas informações em um *CSV* válido. O código utilizado para tal pode ser observado a seguir.

```
1  defmodule InventoryManagement.Exporters.OffersExporter do
2      @moduledoc """
3      Exporter adapted to export Offers to an external database.
4      """
5
6      alias NimbleCSV.RFC4180, as: CSV
7
8      ...
9
10     @spec export_to_csv(list(dealer)) :: binary
11     def export_to_csv(dealers) do
12         offers = list_offers()
13         dealers_by_id = Map.new(dealers, &{&1.id, &1})
14
15         offers
16         |> Stream.map(&build_data_for_row(&1, dealers_by_id))
17         |> Stream.map(&build_csv_row/1)
18         |> Enum.to_list()
19         |> List.insert_at(0, @headers)
20         |> CSV.dump_to_iodata()
21         |> IO.iodata_to_binary()
22     end
23
24     defp build_csv_row(row_data) do
25         Enum.map(@headers, fn header -> Map.fetch!(row_data, ...
               String.to_existing_atom(header)) end)
26     end
27
28     ...
29 end
```

Na linha 22, é possível observar a função *build_csv_row*, que é utilizada para transformar cada elemento definido anteriormente na função *build_data_for_row* em uma lista que possui a mesma ordenação das colunas definidas inicialmente em *headers*.

Após esta conversão, a *Stream* é transformada em uma lista, e consequentemente todas as operações são executas. Após, uma linha é adicionada para sinalizar o nome das colunas. Por fim, o resultado é enviado para a biblioteca *NimbleCSV*, que converte as informações para um arquivo *CSV* válido.

O processo de conversão envolve a transformação da lista em uma lista de bytes. Após esta conversão, o mesmo é formatado em um número binário válido para ser utilizado externamente.

Portanto, o código desenvolvido é capaz de receber uma lista de fornecedores e retornar todas as ofertas públicas da plataforma em um arquivo *CSV* válido.

5.2.2 Conexão com a empresa compradora

Para a realização da conexão com a empresa compradora, será criado um novo módulo dentro do serviço *Jobs* que será responsável pela conexão com o sistema da empresa compradora.

```
1  defmodule Jobs.Exporters.ExternalIntegrationAdapter do
2      @moduledoc """
3      Module to export a binary file to an external integration
4      """
5
6      @type send_file(binary()) :: :ok | :error
7      def send_file(binary_file) do
8          client()
9          |> Tesla.post("/offer_importer", %{binary_file: binary_file})
10         |> handle_response()
11     end
12
13     defp build_client do
14         middleware = [
15             {Tesla.Middleware.BaseUrl, get_env(:base_url)},
16             {Tesla.Middleware.Headers, [{"API-Key", ...
                   get_env(:api_key)}]},
17             Tesla.Middleware.JSON
18         ]
19
20         adapter = {Tesla.Adapter.Hackney, []}
21         Tesla.client(middleware, adapter)
22     end
23
24     defp handle_response({:ok, %Tesla.Env{body: "ok", status: ...
           200}}), do: :ok
25
26     defp handle_response({_ok_or_error, data}) do
27         Sentry.capture_message("#{__MODULE__}.send_file", extra: ...
               %{error: inspect(data)})
28         :error
29     end
30
31     defp get_env(key) do
32         :jobs
33         |> Application.fetch_env!(__MODULE__)
34         |> Keyword.fetch!(key)
35     end
36 end
```

O código acima realiza uma conexão do tipo *POST* com o servidor da empresa

compradora e envia o arquivo do *CSV* em formato binário. Por questões de segurança, o servidor da empresa compradora possui uma autenticação utilizando um código de acesso. Este código de acesso é configurado na linha 16 utilizando o *header API-Key*.

Caso esta função retorne algum erro, o mesmo é enviado para o sistema de identificação de erros da plataforma, chamado de *Sentry*.

Importante ressaltar a utilização de variáveis do sistema para configurar a URL de destino e também da chave de autenticação. A implementação destas variáveis será comentado no decorrer deste documento.

5.2.3 Criação do Cronjob

Com o código desenvolvido para geração de um *CSV* e do sistema para conexão com a empresa compradora, é possível agora criar um programa capaz de realizar esta tarefa de maneira periódica, e enviar este CSV a cada intervalo de tempo para o seu destino.

Para tal ação, é desenvolvido um módulo no serviço *Jobs* que é responsável por listar todos os fornecedores da plataforma, e enviá-los para o módulo *OffersExporter* recém criado no serviço *InventoryManagement*.

Para o gerenciamento deste *Cronjob*, será utilizado uma biblioteca do Elixir chamada de *Oban*. Como já comentado, ela possui uma considerável robustez para lidar com erros, permitindo a realização de novas tentativas automaticamente. Além disso, a biblioteca é capaz de guardar o histórico de suas operações no próprio banco de dados.

O módulo com tais operações pode ser observado abaixo.

```
1  defmodule Jobs.SyncJobs.OffersExporterJob do
2    @moduledoc """
3    Job to sync the offers with an external integration.
4    """
5
6    use Oban.Worker,
7      queue: :offers_exporter,
8      max_attempts: 3
9
10   alias CRM.Dealers
11   alias InventoryManagement.Exporters.OffersExporter
12   alias Jobs.Exporters.ExternalIntegrationAdapter
13
14   @impl Oban.Worker
15   def perform(%Oban.Job{}) do
16     Dealers.list_dealers()
17     |> OffersExporter.export_to_csv()
```

```
18      |> ExternalIntegrationAdapter.send_file()
19    end
20  end
```

A configuração interna do *Oban* é realizada no módulo interno do Elixir chamado de *Application*. Para fins de segurança, as configurações do *Oban* também serão definidas por variáveis do sistema.

5.2.4 Configuração do sistema

A configuração do sistema é realizada por meio de variáveis do sistema. Isto aumenta a segurança do projeto, visto que estas variáveis são configuradas internamente no servidor, e apenas a pessoa responsável pela manutenção do servidor possui acesso a estas informações. Além disso, por ser configurado internamento no servidor, é possível alterar estas informações de maneira fácil, sem alteração direta no código.

Para a implementação destas variáveis, é utilizado os módulos internos de configuração que cada serviço *Umbrella* que a aplicação possui.

Portanto, para a configuração da conexão com a empresa compradora, é implementado o seguinte código no módulo de configuração do *Jobs*:

```
1  config :jobs, Jobs.Exporters.ExternalIntegrationExporter,
2      base_url: System.get_env("EXTERNAL_INTEGRATION_BASE_URL"),
3      api_key: System.get_env("EXTERNAL_INTEGRATION_API_KEY")
```

Para a configuração do módulo de periodicidade, é implementado o seguinte código no módulo de configuração do *Jobs*:

```
1  config :jobs, Oban,
2      repo: Root.Repo,
3      queues: [offers_exporter: 1],
4      plugins: [
5      {Oban.Plugins.Pruner, max_age: 86_400},
6      {Oban.Plugins.Cron,
7          crontab: [
8              {System.get_env("OFFERS_EXPORTER_CRONJOB", "*/30 * * * ...
                   *"), Jobs.SyncJobs.OffersExporterJob}
9          ],
10         timezone: "Europe/Berlin"},
11     Oban.Plugins.Lifeline
12     ]
```

É possível observar acima que o código implementado possui a configuração padrão de realizar o envio das ofertas a cada 30 minutos. Por ser definido por uma va-

riável do sistema, é possível ainda alterar este intervalo alterando o código do *Cronjob*, caso necessário.

5.3 TERCEIRA ETAPA

O objetivo da terceira etapa é desenvolver um plano estratégico inicial para a migração das funcionalidades da plataforma, a fim de permitir um futuro desligamento de todo o painel de controle presente na *startup*.

5.3.1 Mapeamento das funcionalidades

A funcionalidades que estão presentes apenas no lado da *startup* foram mapeadas, com o propósito de dimensionar o tamanho desta migração. Após o mapeamento, as funcionalidades foram separadas seguindo a metodologia de priorização *MoSCoW*.

Os detalhes de cada funcionalidade pode ser observado nas seções a seguir.

5.3.1.1 Suporte a ofertas *only-leasing*

Atualmente, a plataforma da empresa compradora apenas possui suporte para ofertas que, além de serem ofertadas em um contrato do tipo *leasing*, precisam ser ofertas em um contrato de venda. Isso é um bloqueante principalmente para fornecedores que trabalham apenas com esse tipo de contrato, além de limitar fornecedores de criar ofertas mais específicas.

Desta forma, é possível classificar o grau desta funcionalidade como *Must-have* (Deve ter).

5.3.1.2 Suporte a múltiplos serviços de entregas

Atualmente, a plataforma da empresa compradora apenas possui suporte para um tipo de entrega do veículo para o consumidor final. Isso é considerado um limitador para fornecedores que fornecem suporte para diferentes tipos de entrega, como entrega diretamente na porta do consumidor, ou de retirada na própria montadora.

Contudo, esta não é uma funcionalidade que bloqueia a inserção de novas ofertas. Deste modo, é possível classificá-la como *Could-have* (Poderia ter).

5.3.1.3 Suporte a condições especiais

A plataforma da *startup* possui suporte para a implementação de condições especiais, que normalmente estão ligadas a programas público e privados que visam investir em setores específicos. Um exemplo recente foi a implementação de um bônus do governo alemão para o uso de veículos elétricos.

Atualmente, cerca de 10% das ofertas ativas da plataforma possuem suporte a pelo menos uma condição especial.

Infelizmente, a plataforma da empresa compradora não possui suporte para uma definição dinâmica destas condições, sendo necessário mover um time para a implementação de cada uma delas. Isso se tornou um problema, por exemplo, para fornecer suporte ao bônus do governo alemão que foi mencionado anteriormente.

Contudo, não há indícios de que a falta desta funcionalidade poderá causar o bloqueio na inserção de novas ofertas. Deste modo, é possível classificá-la como *Should-have* (Deveria ter).

5.3.1.4 Suporte à análise de orçamento

A plataforma da *startup* possui suporte para ferramentas de análise de orçamento para fornecedores. O objetivo da funcionalidade é permitir com que os fornecedores tenham controle direto sobre quantos clientes eles ainda podem abrir um contrato. Atualmente, cerca de 15% dos fornecedores utilizam esta ferramenta para controle e planejamento interno.

Por ser considerado um diferencial da *startup*, e visto por uma quantidade razoável de fornecedores como uma ferramente importante para controle, é possível classificar esta funcionalidade como *Should-have* (Deveria ter).

5.3.1.5 Suporte à exportação de clientes

A plataforma da *startup* possui suporte para exportar dados referentes aos clientes dos fornecedores. Os arquivos gerados geralmente são arquivos do tipo *CSV*, que posteriormente podem ser usados para análises em ferramentas de *Business Intelligence* ou alguma plataforma externa específica para tal fim.

Contudo, esta ferramenta não é muito utilizada, visto que estas informações podem ser também obtidas a partir de outros serviços externos. Desta forma, é possível classificar esta funcionalidade como *Could-have* (Poderia ter).

5.3.2 Definição da Estratégia

Com as funcionalidades definidas, é possível estabelecer uma linha do tempo com a ordem de migração. O argumento para separar a prioridade de funcionalidades que estavam com mesma categoria da metodologia *MoSCoW*, foi o impacto causado na realidade dos consumidores. Desta forma, o suporte para múltiplos serviços de entregas tem uma prioridade maior que a exportação de dados, por exemplo.

A Tabela 18 a seguir apresenta os detalhes de uma possível ordem de implementação.

Ordem de implementação	**Funcionalidade**
1°	Suporte a ofertas *only-leasing*
2°	Suporte à análise de orçamento
3°	Suporte a condições especiais
4°	Suporte a múltiplos serviços de entregas
5°	Suporte à exportação de clientes

Tabela 18 – Lista de prioridade para as funcionalidades.

Importante ressaltar a inexistência de funcionalidades do tipo *Won't-haves* (não terá por enquanto). O motivo é que a intenção é migrar todas as funcionalidades existentes para a empresa compradora, e portanto, todas as funcionalidades possuem, pelo menos, a categoria *Could-haves* (poderia ter).

É importante ressaltar que a migração de cada uma destas ofertas necessitam de planejamentos específicos, e que provavelmente terão tarefas menores a serem realizadas. Neste contexto, a metodologia *Scrum* pode ser utilizada para um planejamento prático, tendo pelo menos uma *sprint* para cada funcionalidade.

Além disso, é possível que as prioridades se alterem ao decorrer do desenvolvimento. Com o uso da metodologia *Scrum*, é possível alterar a prioridade das tarefas de maneira eficiente.

6 ANÁLISE DE RESULTADOS

Neste capítulo serão analisados os resultados obtidos durante o desenvolvimento do projeto.

6.1 ANÁLISE DO GRAU DE TRANSFERÊNCIA DE OFERTAS

Com as etapas do projeto finalizadas, é possível observar o grau de sucesso de transferência das ofertas para um sistema a outro. Para realizar tal análise, foram coletadas informações de todas as ofertas enviadas para a empresa compradora até a data da medição. Com estas informações é possível definir quais delas foram criadas com sucesso.

Além disso, com os dados das ofertas que falharam durante o processo, foi possível identificar a principal causa dos erros.

É possível observar o grau de sucesso e a data que a medição foi realizada na Tabela 19 abaixo.

Data da medição	**Total de ofertas**	**Grau de sucesso**	**Principal causa dos erros**
04/05/2023	9840	93.7%	Falta de campos e mapeamento incorreto
07/06/2023	13160	93.1%	Diferentes regras de validação
29/06/2023	14541	95.5%	Falta de informações a respeito do veículo
20/07/2023	12975	95.7%	Falta de informações a respeito do veículo
31/08/2023	12255	95.0%	Falta de informações a respeito do veículo
05/10/2023	11491	95.7%	Falta de informações a respeito do veículo

Tabela 19 – Resultados obtidos por meio da transferência de ofertas.

Analisando a tabela em questão, é possível observar uma diferença grande nas regras de validação para a publicação de ofertas em ambas as plataformas, principalmente no que se refere às informações de consumo dos veículos. O momento também coincidiu com a implementação de uma nova regra na União Europeia, chamada de *WLTP*, que basicamente unifica a análise de consumo para carros elétricos, híbridos e a combustível fóssil.

Esta nova regra para caracterizar o consumo de veículos obrigou a *startup* a alterar as regras de validação, para seguir este padrão. Esta alteração foi a responsável pelo aumento do grau de sucesso de 93.1% para 95.5%, do dia 07 de junho ao dia 29 de junho. Contudo, esta alteração apenas teve efeito para quando as ofertas eram atualizadas pelos fornecedores. Isso significa que ofertas que estavam estáveis na plataforma, ainda estão com estas informações faltando.

Mas é possível observar na tabela abaixo que os erros causados pela falta destas informações diminuiu consideravelmente da primeira medição até a última, realizada no dia 05 de outubro.

Data da medição	**Erros causados por diferentes regras de validação**
04/05/2023	48.7%
07/06/2023	25.57%
29/06/2023	26.68%
20/07/2023	20.27%
31/08/2023	15.51%
05/10/2023	14.43%

Tabela 20 – Proporção de erros causados por diferentes regras de validação.

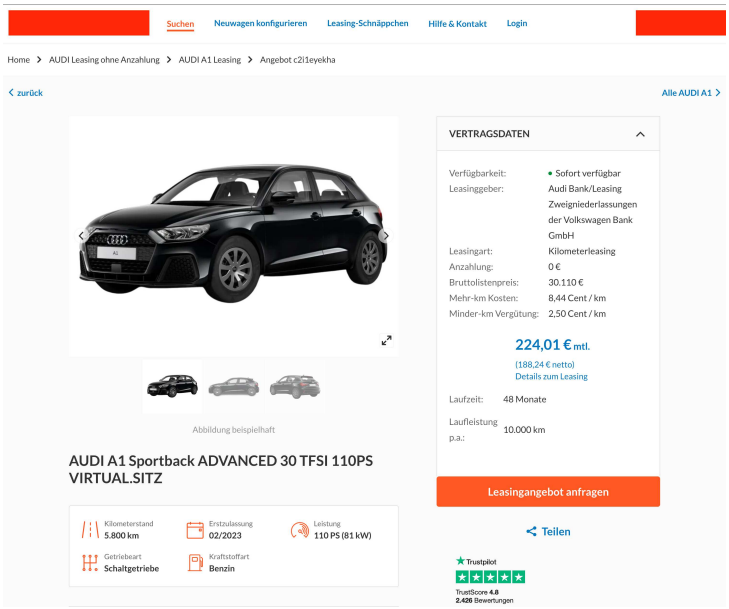
Atualmente, os problemas estão centrados na falta de informações dos veículos na plataforma. Por possuírem fontes de dados distintas, alguns veículos presentes na startup não estão disponíveis na empresa compradora, e vice-versa. Do montante total de erros, cerca de 61.53% dos atuais erros se devem a esta falta de informações.

6.2 OFERTAS EM AMBAS AS PLATAFORMAS

Atualmente, quase 96% das ofertas enviadas são criadas com sucesso na plataforma da empresa compradora. Um exemplo de uma oferta que está disponível em ambas as plataformas pode ser observado nas Figuras 9 e 10 a seguir.

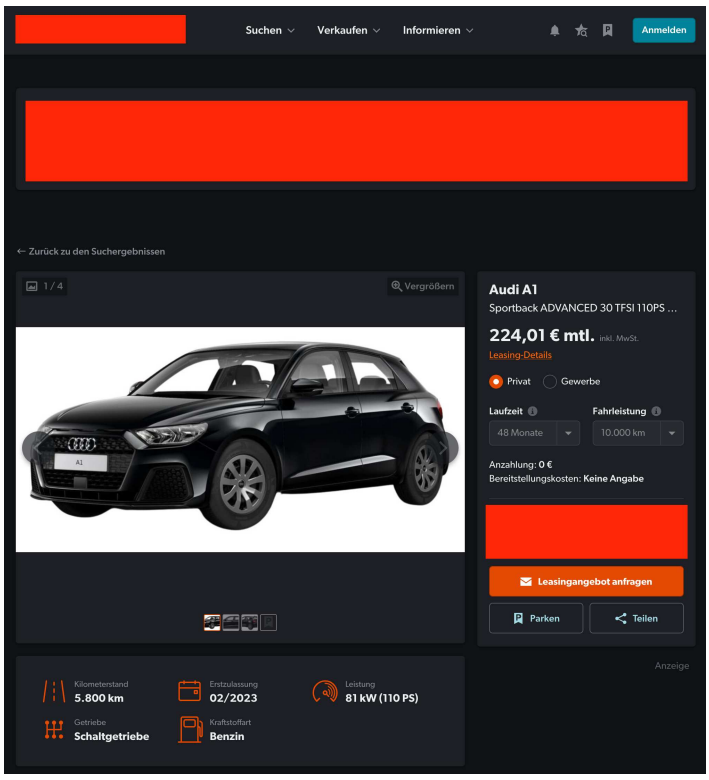
Importante ressaltar que logos e outras referências às empresas foram censurados das imagens.

Figura 9 – Exemplo de oferta na startup.



Fonte: Arquivo pessoal.

Figura 10 – Exemplo de oferta na empresa compradora.



Fonte: Arquivo pessoal.

6.3 PROBLEMAS DA ARQUITETURA

Durante a análise detalhada do processo, identificou-se que a estratégia baseada em *Cronjobs*, embora eficaz em muitos aspectos, trouxe consigo uma notável lentidão na execução do procedimento de replicação das ofertas. Essa circunstância emergiu como um desafio significativo em relação à perspectiva futura da empresa, que tem como meta migrar a recepção de dados diretamente do *Frontend* da *startup* para o banco de dados da empresa compradora.

A complexidade crítica reside no fato de que, devido à natureza periódica desse sistema, as modificações nas ofertas do lado da *startup* não são instantaneamente refletidas no banco de dados da empresa compradora. Recentemente, essa problemática se manifestou de maneira evidente quando uma oferta foi inadvertidamente criada por um fornecedor, e, mesmo após esforços de exclusão, persistiu ativa na plataforma, ocasionando prejuízos substanciais.

Essa situação ressalta a necessidade de soluções que otimizem a sincronização entre as plataformas envolvidas, visando mitigar possíveis impactos negativos decorrentes de descompasso nos dados e garantir a integridade e eficiência operacional do sistema.

6.4 PLANEJAMENTOS FUTUROS

A análise do grau de transferência de ofertas está acontecendo de forma recorrente, e a expectativa é que o grau de assertividade aumente com o passar dos próximos meses. É esperado, inclusive, que este valor aumente consideravelmente após o começo da terceira etapa do desenvolvimento, que envolverá a migração de funcionalidades.

No contexto deste projeto, é esperada a migração das funcionalidades resultantes da terceira etapa, tendo um trabalho mais centralizado no lado da empresa compradora. Há um grande desafio em jogo na plataforma compradora que é o completo suporte a ofertas que são apenas do tipo *leasing*. Para tal suporte, serão necessárias alterações em diversos micro-serviços da plataforma.

Uma das metas para 2024 é mover todas as funcionalidades necessárias para o funcionamento total da plataforma da empresa compradora. O objetivo é, até final do ano, desativar toda a plataforma de controle de ofertas da *startup*, permitindo aos fornecedores o uso prático de apenas uma das plataformas.

7 CONCLUSÃO

Em síntese, o projeto alcançou êxito ao enfrentar o desafio complexo de integrar duas entidades automotivas distintas, uma *startup* de *leasing* e uma empresa de comércio de veículos já consolidada na Alemanha.

O sucesso da utilização da metodologia *Scrum* como forma de planejamento interno permitiu que a mesma fosse finalizada de maneira relativamente rápida. O tempo total de duração do projeto foi de sete meses.

A migração de mais de 10 mil ofertas, implementada por meio de *Cronjobs* e envio de *CSV*, proporcionou uma solução eficaz, embora alguns desafios de desempenho tenham sido identificados, exigindo considerações futuras.

Até o presente momento, cerca de 95% das ofertas enviadas foram criadas com sucesso no sistema da empresa compradora. Contudo, há ainda a demanda para aumentar esse grau de assertividade. Desta forma, é necessário um trabalho contínuo até o final da migração para que os 5% restantes sejam migradas corretamente.

As análises detalhadas das arquiteturas e a sincronização contínua durante o processo de migração possibilitaram a coexistência das plataformas, mantendo funcionalidades exclusivas da *startup* pós-migração.

Infelizmente, algumas limitações técnicas, principalmente relacionadas ao desempenho, foram observadas. Tais questões podem se tornar um problema caso os fornecedores necessitarem que as alterações nas ofertas sejam atualizadas em ambas as plataformas de maneira instantânea.

Para a migração das funcionalidades, foi desenvolvido um pequeno plano estratégico que pode servir como uma introdução ao tópico. Esta fase inicial é importante, visto que o próximo objetivo estabelecido é a remoção completa do *Backend* da *startup*.

Por fim, é possível concluir que o projeto não apenas resolve desafios imediatos, mas estabelece uma base sólida para a evolução contínua da integração entre essas entidades automotivas.

REFERÊNCIAS

BUNDESVERBAND DEUTSCHER LEASING-UNTERNEHMEN. **We are the voice of the leasing sector**. Berlim, 2020.

CARVALHO, Vinícius. **PostgreSQL**: Banco de dados para aplicações web modernas. [*S.l.*]: Casa do Código, 2017.

ELIXIR. **Dependencies and umbrella projects**: Umbrella projects. [*S.l.*: *s.n.*], 2023. Disponível em: https://hexdocs.pm/elixir/1.16/dependencies-and-umbrella-projects.html. Acesso em: 19 nov. 2023.

GREGORI, Chris. **Getting our feet wet with Elixir Umbrella Applications**: What is an Elixir Umbrella application? [*S.l.*: *s.n.*], 2020. Disponível em: https://medium.com/multiverse-tech/getting-our-feet-wet-with-elixir-umbrella-applications-8ba4b1f7b7dd. Acesso em: 19 nov. 2023.

HIVELOCITY. **What is Cron Job?** [*S.l.*: *s.n.*], 2015. Disponível em: https://www.hivelocity.net/kb/what-is-cron-job/. Acesso em: 19 nov. 2023.

JAMES LEWIS, Marting Fowler. **Microservices**: A definition of this new architectural term. [*S.l.*: *s.n.*], mar. 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 12 nov. 2023.

JURIC, Saša. **Elixir in Action**. [*S.l.*], 2015.

LEASEUROPE. **Leaseurope Index Q1 2023**. Bruxelas, 2023.

MAURO NUNES, Henrique O'Neill. **Fundamentos de UML**. [*S.l.*]: Editora de Informática, 2003.

MENDES, Iasmin Santos. Arquitetura Monolítica vs Microsserviços: Uma análise comparativa. Universidade de Brasília, Brasília, 2021. Disponível em: https://bdm.unb.br/bitstream/10483/30715/1/2021_IasminSantosMendes_tcc.pdf. Acesso em: 12 nov. 2023.

OBAN. **Oban**: Robust job processing for Elixir. [*S.l.*: *s.n.*], 2015. Disponível em: https://getoban.pro/. Acesso em: 19 nov. 2023.

OLIVEIRA, Ronielton Rezende. **A Técnica de Priorização MoSCoW**. [*S.l.*]: Management Plaza Internacional, 2014. Disponível em: https://www.ronielton.eti.br/publicacoes/artigoprince2moscow2014_mpbr.pdf. Acesso em: 12 nov. 2023.

OLIVEIRA, Samuel Silva de. Bancos de Dados Não-relacionais: Um novo paradigma para armazenamento de dados em sistemas de ensino colaborativo. Macapá, 2014. Disponível em: https://www2.unifap.br/oliveira/files/2016/02/35-124-1-PB.pdf. Acesso em: 5 nov. 2023.

RAFAEL DIAS RIBEIRO, Horácio da Cunha e Sousa Ribeiro. Métodos Ágeis em Gerenciamento de Projetos. **SPIN Educação Profissional**, Rio de Janeiro, 2015. Disponível em: https://www.faeterj-rio.edu.br/downloads/bbv/0059.pdf. Acesso em: 12 nov. 2023.