MATEUS LOPES DA COSTA TUDÉIA

DESENVOLVEDOR *BACKEND* NA EMPRESA *IOASYS*

LAVRAS – MG
2022

MATEUS LOPES DA COSTA TUDÉIA

DESENVOLVEDOR *BACKEND* NA EMPRESA *IOASYS*

Relatório de Estágio Supervisionado apresentado à Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação, para obtenção do título de Bacharel.

Profº. Dilson Lucas Pereira
Orientador

LAVRAS - MG
2022

MATEUS LOPES DA COSTA TUDÉIA

RELATÓRIO DE ESTÁGIO: DESENVOLVEDOR BACKEND NA EMPRESA IOASYS

Relatório de Estágio Supervisionado apresentado à Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação, para obtenção do título de Bacharel.

Prof. Dilson Lucas Pereira	ICET/UFLA
Prof. Renata Teles Moreira	ICET/UFLA
Prof Maurício Ronny de Almeida Souza	ICET/UFLA

Prof. Dilson Lucas Pereira
Orientador

LAVRAS - MG
2022

AGRADECIMENTOS

Primeiro inicio agradecendo a Deus pelo amor e proteção, além da força e coragem que me deu para conseguir passar por todas as dificuldades durante esse processo, que não foram poucas. Agradeço a minha família pelo apoio incondicional, ele foi fundamental para que eu continuasse firme, especialmente minha mãe, Maria da Paz Lopes Tudéia, e meu pai, Silvano da Costa Tudéia, e minha tia, Maria de Jesus Tudéia. Agradeço também aos muitos amigos que fiz durante esse período, e que agora fazem parte da minha história, especialmente cito o Dian di Grazia, Rafael da Silva Ribeiro, Eduardo Carvalho Leite, Paulo Dehon Cardoso Cedro e Júlia Lima, saibam que a gratitão pela nossa amizade e pelos momentos que passamos juntos eu levarei sempre no meu coração. Ainda gostaria de agradecer a Ana Caroline Ventris Godoy pela parceria de tivemos durante todo esse processo e por ter estado comigo nos momentos que mais precisei de apoio e companhia. Por fim, agradeço ao muitos momentos que tive de aprendizagem, que muito me ensinou sobre vida e me fez crescer e evoluir muito mais que profissionalmente.

RESUMO

Este relatório tem como objetivo relatar a experiência do estagiário na realização do estágio, não apenas na parte de codificação, mas toda a área que a Engenharia de *Software* abrange, mais especificamente gerência de projetos, arquitetura, qualidade de *software*, interação humano-computador, aprendizados essenciais para a realização do estágio. O estágio descrito nesse relatório foi realizado na empresa *ioasys*, e teve início em 20 de novembro de 2020 e término no dia 01 de maio de 2021.

Palavras-chave:. Engenharia de *Software*; Arquitetura; Gerência de projetos.

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

API	Application Programming Interface
BSI	Bacharelado em Sistemas de Informação
CQRS	Segregação de Responsabilidade de Comandos e Consultas
DDD	Domain Driven Design
IBGE	Instituto Brasileiro de Geografia e Estatística
LESS	Large-Scale Scrum
PBR	Product Backlog
PO	Product Owner
UFLA	Universidade Federal de Lavras
VLI	VLI Multimodal S.A.

SUMÁRIO

1 INTRODUÇÃO

Falar do foco profissional, minimizar descriçao do estágio

Usualmente o discente quando inicia seu processo de aprendizado em um curso de graduação, principalmente os do Bacharelado em Sistemas de Informação (BSI), o faz com no mínimo dois objetivos, ou iniciar uma carreira académica, com o foco em trabalhar como pesquisador ou um docente, ou uma carreira profissional no mercado de trabalho. E para ingressar no mercado de trabalho e ser um profissional mais completo e competitivo é importante que o discente tenha uma experiência na área que deseja atuar.

O Componente Curricular Estágio Supervisionado/TCC no Curso de Bacharelado em Sistemas de Informação da UFLA pode ser realizado em três modalidades, sendo uma delas o Estágio em uma Organização. Este trabalho caracteriza-se como um Relatório de Estágio em uma Organização, o qual entende-se por “qualquer atividade desenvolvida por um discente em um ambiente de trabalho que visa à preparação para o trabalho produtivo na sua área de formação” (UFLA, 2020). Com o objetivo de adquirir uma excelente experiência profissional a fim de se destacar no mercado de trabalho o estágio realizado pelo autor se concretizou atuando como Desenvolvedor de *Software*, entre novembro de 2020 e abril de 2021, na empresa *Innovation Oasys* Desenvolvimento de Sistemas Ltda, ou simplesmente *ioasys*. O recorte apresentado possui o objetivo de sintetizar o ambiente em que o trabalho foi executado, bem como as vertentes do ensino do curso de Sistemas de Informação que se conectam com as atividades práticas desenvolvidas.

Além deste capítulo, este documento está organizado como segue. O Capítulo 2 apresenta uma visão da organização, que o trabalho foi realizado, de seu processo de desenvolvimento de software e de seus produtos. O Capítulo 3 aborda os conceitos e as respectivas áreas de atuação das atividades executadas. O Capítulo 4 descreve as tecnologias utilizadas. O Capítulo 5 apresenta as considerações finais acerca das atividades desempenhadas durante o estágio.

2 SOBRE A ORGANIZAÇÃO

Este capítulo tem como objetivo descrever a organização em que o estágio foi realizado, mencionando uma breve contextualização de sua história, momento atual, processos organizacionais, bem como os respectivos produtos relacionados à área de atuação do autor, a área de engenharia de software.

2.1 Ioasys

A *ioasys* (*Innovation Oasys* Desenvolvimento de Sistemas) é uma empresa fundada em 2012, de transformação digital, onde seus clientes são auxiliados a entregarem mais valor a seus usuários através de soluções digitais atuais e alinhadas com as melhores diretrizes do mercado. Atualmente a empresa já conta com quase 400 funcionários. Recentemente, a *ioasys* foi adquirida pela Alpargatas S.A, uma indústria brasileira de capital aberto, do ramo de calçados e lonas, dona também de marcas como Havaianas e *Rothy's.* A sede da empresa encontra-se em Belo Horizonte/MG, tendo escritórios nas cidades de Lavras/MG e São Paulo/SP.

A empresa tem como especialidade trazer o melhor do universo digital para seus clientes, seguindo os princípios de metodologia ágil e abordagem centrada ao usuário, modificando a realidade das pessoas com tecnologia e inovação de ponta a ponta. Sua cultura organizacional de lema "De pessoas, para pessoas" traz uma abordagem organizacional horizontal[1] e focada nos colaboradores, tendo a diversidade como um pilar. O próprio lema cultural já traduz a importância de um ambiente heterogêneo, repleto de vivências, experiências e impressões da sociedade e suas necessidades[2].

A organização operacional é dividida em três setores, sendo eles: Inovação, Design e Tecnologia. Neste relatório iremos dar ênfase nos trabalhos desempenhados pelo autor que estão relacionados ao setor de Tecnologia da empresa, será enfatizado mais especificamente na área de desenvolvimento de software utilizando o framework .Net e .Net Core.

---

[1] A organização horizontal, orientada a processos de negócios, torna a operação mais flexível, centrada nos propósitos da organização e com maior proximidade do consumidor final. (OSTROFF, 1999)

[2] Todos os valores dispostos nesta seção foram fornecidos diretamente pela empresa mediante consulta prévia à realização deste trabalho.

2.2 Clientes

A ioasys possui diversos clientes nas mais variadas áreas, como Burger King, Suvinil, Havaianas, BASF, CVC e outros. Este relatório foca nos projetos do cliente VLI Multimodal S.A., visto que foi o experienciado pelo autor.

A VLI é uma empresa de logística do Brasil que controla algumas concessionárias de transporte ferroviário de cargas, uma empresa oferece aos seus clientes e ao mercado brasileiro serviços logísticos que integram ferrovias, terminais e portos, por meio de operações transparentes e eficientes.

A VLI atua em todos os estados da região Sudeste, além dos estados de Goiás, Bahia, Sergipe, Maranhão, Tocantins e Distrito Federal. As atividades da empresa estão estruturadas em cinco corredores de logística integrada: Norte, Nordeste, Sudeste, Fluminense e Paulista. Com mais de 10 mil quilómetros de ferrovias, esses corredores estão estrategicamente localizados nas principais regiões produtoras de *commodities* agrícolas e minerais e de produtos industrializados e siderúrgicos.

Dado esse contexto e a vasta atuação da empresa na prestação de serviços logísticos, para esses segmentos produtores, surgiu-se a necessidade de realizar a gestão das vendas da empresa de maneira digital e interativa. A partir desta necessidade a *ioasys* foi contratada com o objetivo propor uma solução. Para isso, por tanto, iniciou-se o desenvolvimento do projeto Plano de Vendas VLI. Também chamado de PV Digital (Plano de Vendas Digital), o projeto nasceu para ser uma ferramenta de apoio simples e funcional aos gestores nas atividades de prospecção e negociação de vendas junto aos seus clientes.

O objetivo do projeto é realizar a inclusão, manipulação e gestão das vendas de transporte, inicialmente, ferroviário e portuário. A aplicação conta com diversos relatórios que permite a VLI realizar as tomadas de decisões da sua área comercial. O projeto tem duas frentes: uma aplicação web e um *mobile,* disponibilizado na tecnologia IOS.

A equipe responsável pelo PV Digital tem a alocação de cinco desenvolvedores, dois *backend*, dois *frontend web* e um *frontend* app e *web*, tem também a alocação compartilhada entre os projetos VLI de um arquiteto de *software* e um designer, além do apoio contínuo do cliente como um líder digital e um analista de negócios.

3 CONCEITOS E TECNOLOGIAS UTILIZADAS

Este Capítulo tem como objetivo contextualizar os leitores dos principais conceitos utilizados no decorrer das atividades desenvolvidas pelo autor, que também são amplamente utilizados no mercado de trabalho de desenvolvimento de *software*. As primeiras subseções deste capítulo abordam os conceitos do anteriormente citada, metodologia *Scrum*, utilizada para manter o processo de desenvolvimento ágil e facilmente adaptável às mudanças e as subseções seguintes irão dar breve explicação sobre as diversas tecnologias utilizadas pelo autor no desenvolvimento do projeto. O *Scrum* é implementado no desenvolvimento de projetos na *ioasys*, porém o *framework* é aplicado adaptando-se da melhor forma para suprir a necessidade do contexto de cada time.

3.1 Scrum

O termo "ágil" em vigor provém do chamado Manifesto Ágil (2001), e se propõe a disponibilizar valores e princípios para que times atuem em seus respectivos contextos.

O Manifesto Ágil foi criado por um time composto de diversos autores provenientes de diferentes escolas de pensamento voltados para o desenvolvimento de software. Embora tenha sido criado há vinte anos da data de escrita deste trabalho, os valores e os princípios propostos perduram até os dias atuais e seguem sendo fundamentais para diferentes metodologias de trabalho usadas em empresas de tecnologia. "Indivíduos e interações mais que processos e ferramentas", "Software em funcionamento mais que documentação abrangente", "Colaboração com o cliente mais que negociação de contratos" e "Responder a mudanças mais que seguir um plano" são os valores principais propostos pelo Manifesto Ágil, os quais são mais bem detalhados em outros doze pilares.

O *Scrum* é uma metodologia ágil de gerenciamento de projetos que ajuda pessoas, times e organizações a gerar valor com soluções adaptadas para problemas complexos e é composto por artefatos, eventos e pelo time *Scrum*. Cada artefato contém um compromisso para garantir que ele forneça informações que aumentem a transparência e o foco contra o qual o progresso pode ser medido: Para o *Product Backlog*, é a Meta do produto. Para o *Sprint Backlog*, é a Meta da *Sprint*. Para o Incremento, é a Definição de Pronto (SCHWABER; SUTHERLAND, 2020).

Para Schwaber e Sutherland (2020), o *Product Backlog* é uma lista ordenada e

emergente do que é necessário para melhorar o produto e é a única fonte de trabalho realizado pelo time *Scrum*. Seu compromisso é a Meta do Produto, que descreve um estado futuro do produto que pode servir como um alvo para o time *Scrum* planejar.

O *Sprint Backlog* é composto pela Meta da *Sprint* (porque), o conjunto de itens do *Product Backlog* selecionados para a *Sprint* (o que), bem como um plano de ação para entregar o Incremento (como). E por fim, um incremento é um trampolim concreto em direção à Meta do Produto. Cada incremento é adicionado a todos os incrementos anteriores e completamente verificados, garantindo que todos os incrementos funcionem juntos (SCHWABER; SUTHERLAND, 2020).

Segundo Schwaber e Sutherland (2020), o time *Scrum* consiste em um *Scrum Master*, um *Product Owner* e *Developers*. O *Scrum Master* é responsável por estabelecer o *Scrum* conforme definido no Guia do *Scrum*, ele faz isso ajudando todos a entenderem a teoria e a prática do *Scrum*, tanto no time *Scrum* quanto na organização. O *Product Owner* é responsável por maximizar o valor do produto resultante do trabalho do time *Scrum*. Por fim, os *Developers* são as pessoas do time *Scrum* que estão comprometidas em criar qualquer aspecto de um incremento utilizável a cada *Sprint*.

Os eventos são usados no *Scrum* para criar regularidade e minimizar a necessidade de reuniões não definidas no *Scrum*. Cada evento no *Scrum* é uma oportunidade formal para inspecionar e adaptar os artefatos do *Scrum*. Esses eventos são projetados especificamente para permitir a transparência necessária (SCHWABER; SUTHERLAND, 2020).

A *Sprint* é um evento de duração fixa de no máximo um mês para criar consistência de trabalho. Cada *Sprint* pode ser considerado um projeto curto em que algo de valor é gerado ao fim. A *Sprint Planning* inicia a *Sprint* ao definir o trabalho a ser realizado na Sprint. Este plano resultante é criado pelo trabalho colaborativo de todo o time *Scrum* (SCHWABER; SUTHERLAND, 2020).

A *Daily Scrum* é um evento de no máximo 15 minutos para os *Developers* time *Scrum* com o propósito de inspecionar o progresso em direção a Meta da *Sprint* e adaptar o *Sprint Backlog* conforme necessário, ajustando o próximo trabalho planejado. O propósito da *Sprint Review* é inspecionar o resultado da *Sprint* e determinar as adaptações futuras. O time *Scrum* apresenta os resultados de seu trabalho para os principais *stakeholders* e o progresso em direção a Meta do Produto é discutido. Em relação à *Sprint Retrospective*, seu propósito é planejar maneiras de aumentar a qualidade e a eficácia (SCHWABER; SUTHERLAND,

2020).

Como dito no início do capitulo, a *ioasys* utilizado o *scrum* como a principal ferramenta para gestão de projetos, mas o framework foi adaptado a realidade da equipe e essa adaptação pode ser simplificado pelas seguintes etapas: Requisitos, Desenvolvimento, Testes e Suporte.

Na primeira etapa de Requisitos, o *Product Owner*(PO), figura do *Scrum* responsável por direcionar o projeto de acordo com a necessidade do cliente e das demais partes interessadas (os *stakeholders*), faz a especificação de todos os requisitos do projeto, ou seja, são documentado todas as funcionalidades e especificações que as partes interessadas desejam para a solução final e geralmente é organizado em uma lista de atividades que precisam ser feitas para a conclusão do projeto, denominada *backlog* do produto.

Em seguida, na etapa de desenvolvimento, é feita uma priorização das atividades do *Backlog* nas quais as mais prioritárias são inseridas na *Sprint*, conceito que pode ser entendido como um período de tempo utilizado para concluir as atividades propostas. Ao término de uma *Sprint* é feita uma reunião, conhecida como *Sprint Review*, com o objetivo de revisar o que foi realizado. Se ainda existirem tarefas no *Backlog*, esse ciclo se repete, iniciando novamente com uma *Sprint Planning*, reunião na qual é feita uma priorização das atividades do *Backlog* do produto para entrar na Sprint.

Paralelamente à etapa de desenvolvimento, é executada a etapa de testes, conduzida por um profissional especializado em qualidade de *software*. Os testes das funcionalidades são realizados à medida que os desenvolvedores vão finalizando o desenvolvimento. Em geral, ao final da *Sprint* a funcionalidade produzida já está testada pelo profissional de qualidade de *software* e aprovada pelo PO ou pelo cliente.

Por fim, a etapa de Suporte se dá ao final do desenvolvimento da solução, onde, dependendo do que for acordado, a equipe pode ficar disponível para qualquer eventualidade que aconteça com o *software*, como descobrimento de *bugs* ou comportamentos inesperados. Geralmente, ao final das atividades do projeto podem surgir novas demandas, como novas funcionalidades ou até mesmo uma nova necessidade, que precisa ser iniciado um novo projeto, assim, começando novamente o ciclo de desenvolvimento, com a definição dos requisitos por parte do PO.

3.2 Git

Segundo Valente (2020), Git é um sistema de código aberto para controle de versões, comumente utilizado por desenvolvedores de software, porém o sistema faz o versionamento de qualquer tipo de arquivo. Um histórico de alterações dos arquivos é criado, sendo possível saber quais alterações foram realizadas e por quem, tendo total liberdade para retornar às versões anteriores.

O local onde são armazenados os arquivos é chamado de repositório, onde tem-se diferentes *branches*, ramificações do mesmo código em pontos diferentes de desenvolvimento, ou seja, pode-se trabalhar com diferentes versões do mesmo código (VALENTE, 2020). Geralmente são criadas duas *branches* para os ambientes de uma aplicação, uma para desenvolvimento e uma para produção. Cada nova funcionalidade é feita em uma *branch* separada das principais e no fim de seu desenvolvimento e testes é realizado o *merge*, operação que combina diferentes ramificações em apenas uma.

O código do projeto ao qual o autor trabalhou fica armazenado em repositórios utilizando o Git, em que, através destes repositórios, os desenvolvedores realizam o controle de versão dos códigos. Dentre as mais diversas ferramentas existentes para tal tarefa, a plataforma *Azure* DevOps foi a utilizada pela equipe do projeto e deste modo foi a que o autor fez o uso para o controle de versão.

Diversas boas práticas para o controle de versão são seguidas pelo autor, por exemplo, manter três branches separadas para os ambientes de produção, geralmente chamada de "*master*", homologação, frequentemente denominada "*homolog*" e desenvolvimento, comumente chamada de "*develop*". Quando os desenvolvedores iniciam uma nova atividade devem criar uma nova *branch* originada da *develop*. Essa *branch* pode ter, por convenção, seu nome iniciado por "*feature*" ou "*feat*", seguido de uma barra e o nome da funcionalidade que será desenvolvida.

No controle de versão o fluxo de desenvolvimento de um incremento de *software* segue o seguinte processo: Após finalizado o desenvolvimento e os testes da nova atividade referente ao incremento, o desenvolvedor gera uma *release*[3] para o ambiente de

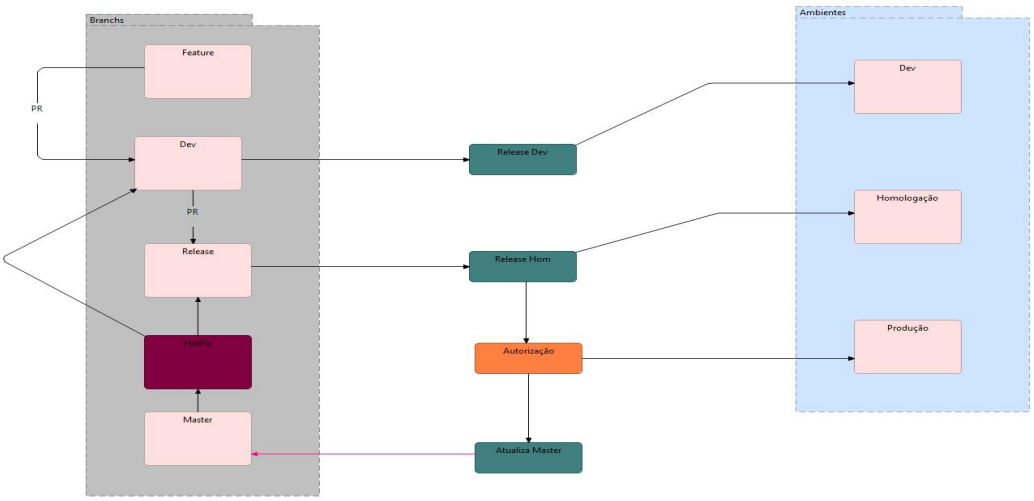
[3] Um Release é a entrega de um sistema funcional que atende objetivos predefinidos. Ela pode ser interna ou externa, ou seja, ela pode ser gerada dentro de um ambiente controlado pela equipe de desenvolvimento ou pode ser publicada no ambiente produtivo, local onde os usuários finais poderão ter acesso ao que foi desenvolvido

desenvolvimento (atualização da branch *develop* com a nova *feature*) para as devidas validações dos critérios de aceite. Em seguida qualquer desenvolvedor do time gera uma nova *release*, mas agora para o ambiente de homologação (é feita a atualização da *branch homolog* a partir da *develop*), para um segundo estágio de testes. Por fim, é gerado a última *release* para o ambiente de produção (atualização da *branch master*).

Além do incremento de *software*, apelidado de *feature*, é utilizado também o conceito de *hotfix*, o *hotfix* tratasse da correção de um problema encontrado no ambiente de produção, podendo ser um *bug* ou um comportamento indesejado não previsto durante o desenvolvimento de uma *feature.* Para o fluxo do *hotfix* é gerado uma *branch* a partir da *master* e assim que finalizado o desenvolvimento é aberto um *pull request*, primeiro para o ambiente de homologação, após a validação do time de qualidade é feito uma release para o ambiente de produção, em seguida, o ambiente de desenvolvimento é atualizado também.

Na Figura 1 pode-se ver a ilustração do fluxo de geração de *release* nos ambientes de desenvolvimento, homologação e produção.

Figura 1: *Git flow* para geração de *release*



Fonte: *ioasys*

3.3 WebApi

API (*Application Programming Interface*) é uma forma de comunicação entre sistemas, no qual permitem a integração entre dois ou mais sistemas, em que um deles, a API, fornece informações e serviços que podem ser utilizados por outros, sem a necessidade dos sistemas que a consomem precisarem conhecer detalhes de implementação do software. Uma

WebApi é uma API que pode ser acessada pela web através do protocolo HTTP, protocolo base de qualquer troca de dados na *Web*, e um protocolo cliente-servidor, ou seja, as requisições são iniciadas pelo destinatário, geralmente um navegador da *Web*.

As *WebAPIs* podem ser utilizadas para fornecer dados e funcionalidades para diversas aplicações móveis, web ou desktop. Por exemplo, em um aplicativo móvel para gerenciamento de notas dos estudantes, seguindo as boas práticas do mercado, o aplicativo deve ser apenas um consumidor de uma fonte de dados, utilizando uma *WebAPI* para manipular os dados necessários, como buscar uma listagem de alunos ou cadastrar uma nova nota.

3.4 .Net Core

O .Net é uma plataforma de desenvolvimento de código aberto criada e mantida pela *Microsoft*, com suporte à criação de diversos tipos de aplicações, por exemplo para dispositivos *mobile Windows Phone* ou APIs e Web APIs para o *backend*. É possível escrever aplicações .Net utilizando as linguagens C#, F e *Visual Basic* e, independente da linguagem escolhida, a aplicação é executada de forma nativa no sistema operacional compatível desejado.

Atualmente existem algumas versões do .Net, em que cada implementação permite executar aplicações .Net em diferentes ambientes, como *Linux, macOS, Windows*, iOS, *Android* e outros. A versão original da implementação .Net, o .Net *Framework* dá suporte a execução de *websites*, serviços, aplicações *desktop* entre outras no sistema operacional *Windows*. O .NET Core, lançado em 2016, é uma implementação chamada de *cross-platform* pois oferece suporte às diferentes plataformas *Windows, Linux, e macOS*, permitindo a criação de *websites*, serviços e aplicativos de console.

No decorrer do estágio em desenvolvimento .Net do autor, foi utilizada a versão .Net Core 3.1, porem foi atualizado a versão do .Net para a versão 6.0, devido a *Microsoft* parar de dar suporte para as versões anteriores a 6.0.

3.5 CQRS (Command Query Responsibility Segregation)

O CQRS (segregação de responsabilidade de comandos e consultas), consiste em um padrão arquitetural (pattern). Apresentado pela primeira vez por Greg Young no artigo

“CQRS Documents by Greg Young”, publicado em 2010, este padrão é descrito como “um padrão muito simples que permite muitas oportunidades de arquitetura que, de outra forma, não existiriam”.

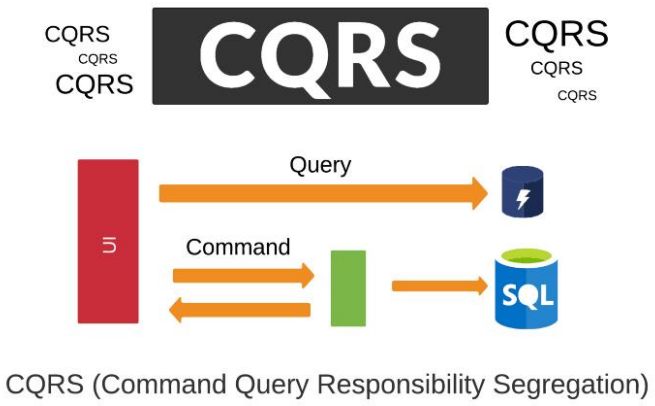
De acordo com Ferreira (2016), a origem do CQRS é baseada no CQS (*Command-Query separation*), criado por Bertrand Meyer durante o desenvolvimento da linguagem de programação Eiffel. A ideia principal do CQS era, como cita Nascimento (2019), que os métodos de uma aplicação podem ser comandos ou consultas, mas nunca ambos.

Nesse sentido, para Young (2010):

> CQRS é simplesmente a criação de dois objetos em que anteriormente havia apenas um. A separação ocorre com base no fato de os métodos serem um comando ou uma consulta (YOUNG, 2010, p.15).

Quando se diz respeito ao uso do CQRS, Martin Fowlertem tem bastante relevância. Em seus estudos, Fowler (2011), comenta que a essência do CQRS é a possibilidade de utilizar, inclusive, modelos diferentes para atualizar e ler informações. Com base nas afirmações dos autores, fica claro que com o CQRS divide-se comandos (command) que executam inclusões, alterações e exclusões no modelo de dados e consultas (query), que somente fazem a leitura das informações. De forma simples, pode-se entender o modelo de segregação com demonstrado na Figura 2 .

Figura 2 – CQRS



Fonte: GOOGLE imagens

Fowler (2011), ressalta que, por modelos separados, entende-se que serão modelos de objetos diferentes, executando em processos lógicos diferentes, possivelmente em hardware separados. Para trabalhar com diferentes modelos para escrita e leitura se faz necessário entender e dividir corretamente as responsabilidades.

Para Young (2011):

> Essa separação reforça a noção de que o lado do comando e o lado da consulta têm necessidades muito diferentes. As propriedades arquitetônicas associadas aos casos de uso de cada lado tendem a ser bem diferentes. É necessário entender que no lado da consulta tem-se somente métodos para obter dados, enquanto na parte de comando, são todos os outros eventos que acabam de alguma forma alterando informações no banco de dados (YOUNG, 2011, p.17).

Para entender de forma prática tal divisão, pode-se verificar o exemplo descrito na Figura 3. Na arquitetura padrão será criado somente um serviço que faça inclusões, alterações, exclusões e leituras.

Figura 3 - Exemplo de uso sem CQRS.

Fonte: Young (2010).

Com a aplicação do pattern, tem-se a divisão em dois serviços, como demonstrado na Figura 4.

Figura 4 - Exemplo utilizando CQRS

Fonte: Young (2010).

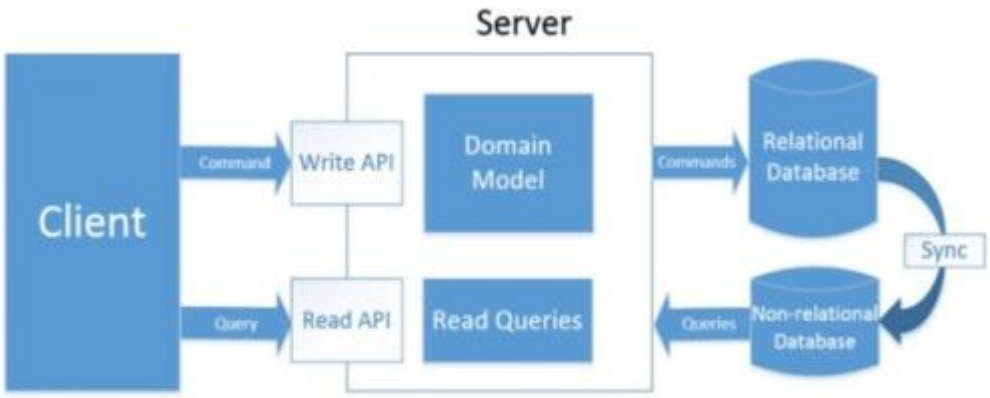
3.5.1 Diferentes tipos de implementação

Pires (2016) ressalta que não existe uma única maneira de implementar o CQRS na sua aplicação, que dependendo da necessidade, pode ser feito de uma forma simples ou muito complexa. Ainda no pensamento de Pires (2016):

> A implementação pode ser feita utilizando modelos diferentes, porém, utilizando um único banco de dados ou, a infraestrutura pode ser distinta, tendo por exemplo, um banco relacional para os comandos e um banco NoSQL para consultas. (Pires, 2016).

A Figura 5 ilustra esta separação do modelo de dados em bancos de dados distintos.

Figura 5 - CQRS com diferentes bancos de dados.



Fonte: Passos (2018).

No entanto, separar em bancos diferentes aumenta a complexidade do projeto, trazendo outra necessidade: o sincronismo de informações.

3.5.2 Sincronismo de informações

O sincronismo de dados entre os bancos pode ser executado de diferentes formas. A aplicação pode implementar atualizações automáticas, ou seja, sempre que uma alteração for realizada no banco de escrita, um processo seja executado para atualizar o banco de leitura. Outros modelos que podem ser implementados são atualizações periódicas, ou seja, diariamente ou em alguns períodos pré-definidos, serão executados processos para atualizar as informações no banco de leitura.

Pires (2016) explica que a estratégia mais utilizada é a de atualização eventual. Tal estratégia parte do princípio que todo dado exibido já pode estar desatualizado. A atualização eventual, consiste em disparar um processo assíncrono para atualização de dados assim que uma alteração no banco de escrita seja realizada.

Tal necessidade apresenta um novo *pattern* que pode ser utilizado, o *Event Sourcing*, pois com ele, tem-se a informação completa da última alteração realizada no banco de dados.

3.5.3 Benefícios do CQRS

Como todo padrão arquitetural criado, o uso do CQRS é recomendado em situações específicas onde são encontrados benefícios na sua utilização. Nesta subseção serão apresentados algumas das vantagens de se utilizar CQRS para desenvolvimento de aplicações. De acordo com Microsoft (2019), um dos benefícios é a possibilidade de realizar o dimensionamento independente dos bancos, ou seja, trazendo a possibilidade de escalar da forma desejada a infraestrutura.

A respeito dessa possibilidade, Ferreira (2016) afirma que, tipicamente, o número de consultas num sistema é muito superior ao número de comandos realizados. Conclui-se, então, que a infraestrutura disponível para o banco de leitura deve ser mais robusta do que a disponível para o banco de escrita.

Fowler (2010) ressalta que o desempenho do banco também pode ser afetado com tal separação, “pode-se aplicar diferentes estratégias de otimização para os dois lados, sendo um exemplo disso, o uso de diferentes técnicas de acesso ao banco de dados”. Tais estratégias permitem que bancos de leitura sejam otimizados para melhor performance durante a leitura e vice-versa.

3.5.4 Desvantagens do CQRS

Assim como vantagens, todos os padrões arquiteturais têm desvantagens. Segundo Fowler (2010), o CQRS não é indicado para sistemas que necessitam ter informações atualizadas do mesmo modo que são lidas, item este, que pode ser resolvido implementando sincronismo entre os bancos de forma automática.

Já para Microsoft (2019):

> A complexidade é outra desvantagem do uso de CQRS. Apesar do padrão ser simples, ele acaba resultando em um design de aplicativo mais complexo. Com base nestas informações, pode-se entender que em projetos que contenham regras de negócio simples, o padrão arquitetural não é o mais indicado para o uso. (MICROSOFT, 2019).

3.5.5 Aplicação de CQRS em microsserviços

Os benefícios da utilização de CQRS para desenvolver microsserviços foram um fator chave para a motivação do *pattern*. Pensando cada vez mais em desenvolver aplicações nativas para nuvem e que possuem premissas de resiliência, escalabilidade, desempenho e disponibilidade, o CQRS acabou ganhando espaço no desenvolvimento de microsserviços.

Para Fowler (2011):

> A utilização de CQRS (Command Query Responsibility Segregation) e a possibilidade de segregar leitura e escrita de dados persistidos, estas operações podem ser escaladas de forma independente, o que acaba sendo um grande ganho na tentativa de criar aplicativos com alto nível de escalabilidade (FOWLER, 2011).

Seguindo a linha do desenvolvimento de microserviços, com CQRS pode-se modularizar o banco de dados, possibilitando a granularização mais fina dos componentes de uma aplicação.

O referido padrão arquitetural foi a base para a construção da arquitetura da aplicação *backend* do projeto Plano de Vendas Digital.

3.6 DDD (*Domain Driven Design*)

A criação de softwares não é algo simples, uma vez que se deve adquirir um bom modelo de domínio para monitorar a complexidade do software que será desenvolvido. Tal modelo deve abranger elementos relevantes significativos para a construção do domínio, de forma que os desenvolvedores venham a conseguir um aproveitamento suficiente (SANTOS, 2015).

Na incessante busca de assegurar a agilidade nas técnicas para o desenvolvimento de um software surgiram várias técnicas e métodos criados para as inúmeras fases ligadas ao desenvolvimento do projeto. Entre elas, apresentando uma abordagem voltada para o domínio do problema nomeada *Domain Driven Design* (DDD) (MATTOS; DOLL; ALMEIDA, 2010).

Agregado na Engenharia de Software, essa abordagem de desenvolvimento de sistemas engloba técnicas e métodos que salientam a associação com o cliente, onde o sucesso e/ou a aceitabilidade de um software estão intimamente ligados ao relacionamento das metodologias utilizadas pelo DDD (DA SILVA; FILHO; DA SILVA, 2018).

O DDD é executado como uma maneira de guiar métodos de desenvolvimento de modo ágil (EVANS, 2003), oferecendo uma alternativa de representar o mundo autêntico na arquitetura, focando e reconhecendo o domínio central, que são por sua vez características relevantes para se obter uma qualidade superior da arquitetura de software (HIPPCHEN *et al*., 2018).

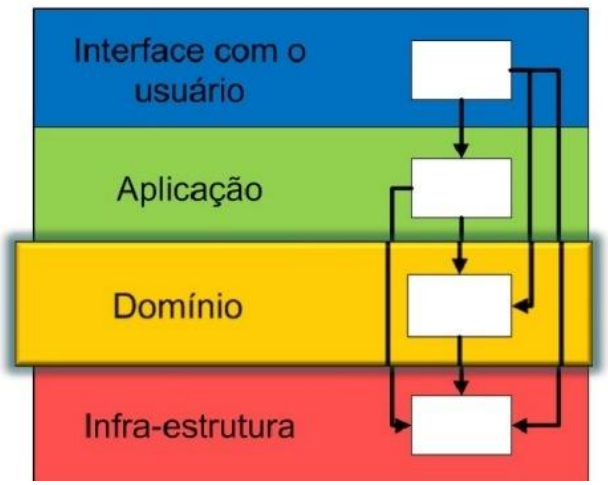
Mattos; Doll; Almeida (2010), ratificam que:

> A primeira aparição do termo Domain Driven Desing foi em 2004, no livro de Eric Evans, com título "Domain Driven Desing: atacando as complexidades no coração do software", onde o autor conceitua DDD como uma forma de pensar e um agrupamento de princípios que irão cooperar na criação de sistemas potentes pelos desenvolvedores (MATTOS; DOLL; ALMEIDA, 2010).

Uma das distinções do DDD em comparação com outras filosofias de projeto é o panorama dado a modelagem, onde a modelagem de domínio e a implementação de aplicação não são consideradas como atividades diferentes, mas sim que se apresenta em outros métodos, sendo um modelo apresentado em códigos (MACEDO, 2009). Alguns domínios compreendem o mundo físico, outros podem ser atingíveis, a carga de informações pode ser grande, porém, alguns modelos são considerados dispositivos redutores dessa sobrecarga de informações (SANTOS, 2015).

Na figura 6, pode-se observar as quatro camadas essenciais seguindo os princípios do DDD, sendo elas (CUKIER, 2010; MATTOS; DOLL; ALMEIDA, 2010):

Figura 6: Arquitetura em camadas.



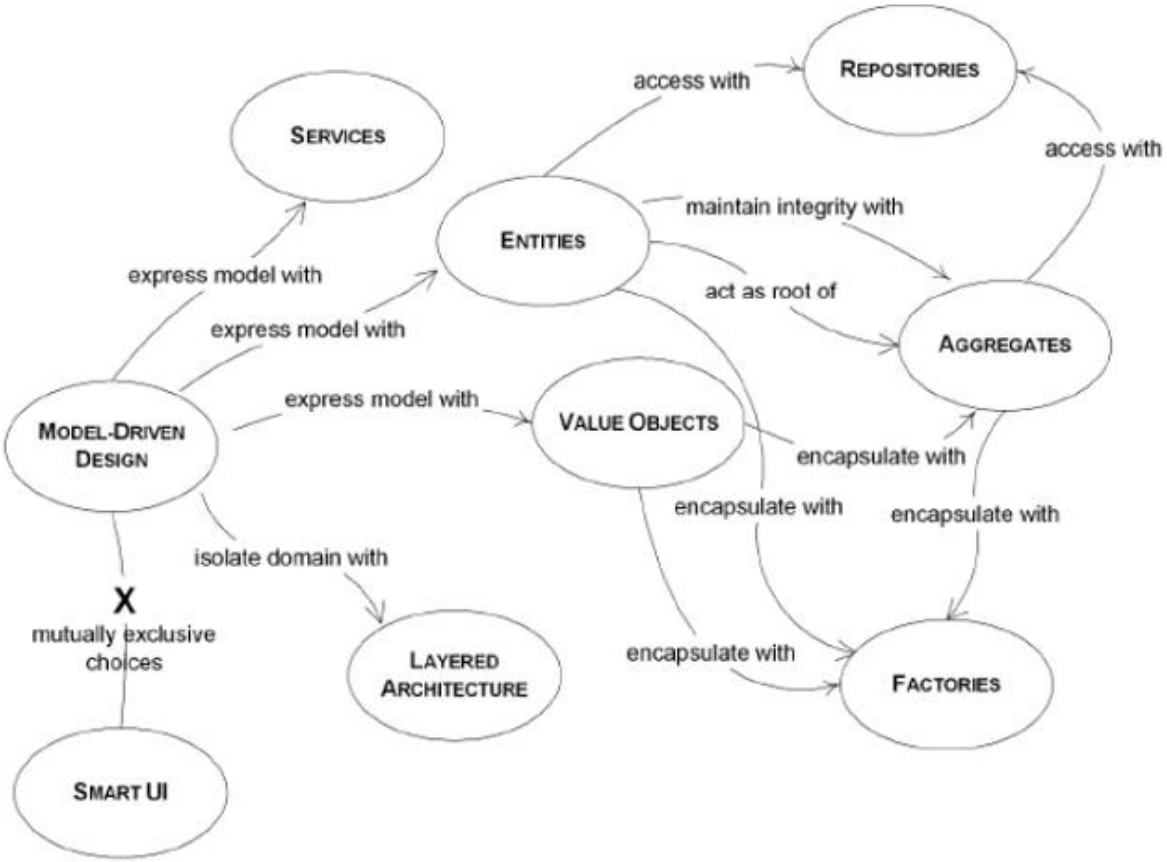
Fonte: GOOGLE Imagens.

- Camada de interface de Usuário: Camada que exibe informações do sistema e executa comandos do usuário.
- Camada de aplicação: Não possui a logicidade do negócio, limitando apenas por ser encarregada por vincular a Interface de Usuário com as camadas inferiores.
- Camada de domínio: onde todo o centro do DDD se encontra, deve se manter o mais isolado possível de todas as outras camadas
- Camada de Infraestrutura: oferece recursos técnicos que darão suporte às camadas superiores, atuando como uma biblioteca e concedendo informações entre elas.

Cada uma dessas camadas deve desempenhar apenas a função a qual se confere, uma vez que que o isolamento da camada domínio é uma premissa para o DDD, sendo esta camada o centro do DDD. Para que o software consiga servir otimamente a um domínio específico, é indispensável que previamente que se estipule uma Linguagem Ubíqua, estabelecendo as definições que integram as conversas cotidianas entre os especialistas e os desenvolvedores, sendo compreendido por todos, sem ambiguidades. Evans (2003), define a Linguagem Ubíqua como sendo uma linguagem comum com termos claramente definidos, que integram o domínio do negócio e que são aplicados por todos os envolvidos no processo do desenvolvimento do sistema. Para que a Linguagem Ubíqua tenha positividade no desenvolvimento do *software*, ela deve representar os desejos dos especialistas e desenvolvedores do domínio, sendo assim, o seu uso gera elaboração do domínio do modelo, minimizando a necessidade de análise e modelos de projetos e possibilitando a concentração da equipe de projeto nas tarefas primordiais durante o desenvolvimento de uma funcionalidade estabelecida (SANTOS, 2015).

Na figura 7, podem-se observar os padrões em DDD para criar a camada de domínio, também conhecidos como blocos de construção.

Figura 7: Mapa de navegação da linguagem do Design Dirigido por Modelos.



Fonte: SANTOS, (2015).

- Entidades: Classes de objetos que carecem de uma identidade, a definição desse objeto deve ser simples, estabelecida pela sua identidade e não seus atributos. Enquadra configurações especiais de modelagem e de design (CUKIER, 2010; MATTOS; DOLL; ALMEIDA, 2010; SANTOS, 2015). A identidade não é estabelecida pela união dos atributos ou pelo estado do objeto, mas sim por uma ininterrupção temporal, sendo que identidades mal acordadas podem gerar uma incongruência de dados (MACEDO, 2009).
- Objetos de valor: Ao contrário do item anterior, os objetos de valor não possuem uma identidade ou descrevem alguma característica, sendo que na maioria das vezes são imutáveis e possuem um curto ciclo de vida (MATTOS; DOLL; ALMEIDA, 2010; SANTOS, 2015).
- Serviços: São as classes que contém lógica de negócio, porém não pertencem a uma Entidade ou a um Objeto de valor (COSTA; HILD, s/d), devem apresentar algum

significado para o domínio (MACEDO, 2009).

- Módulos: No decorrer da evolução do modelo, cresce o número de conceitos com o qual o desenvolvedor irá lidar, assim como o número de classes e suas associações, gerando assim a inevitabilidade de agrupar as classes em módulos, que nada mais são do que blocos lógicos onde o conjunto de classes da aplicação é particionado (MACEDO, 2009).
- Agregados: Este bloco exerce a manutenção da integridade do modelo (CUKIER, 2010); podem ser definidos como conjuntos de Entidades ou Objetos de Valores que são agrupados em uma classe única (MATTOS; DOLL; ALMEIDA, 2010).
- Fábricas: São empregadas para agrupar a informação necessária para a elaboração de objetos complexos ou agregados, gerando uma interface que reproduz os desejos do cliente e uma visão subjetiva do objeto a ser construído (MATTOS; DOLL; ALMEIDA, 2010; SANTOS, 2015).
- Repositórios: Têm como objetivo agrupar toda a lógica necessária para atingir a referência de um objeto, onde estes não necessitam lidar com a infraestrutura para apanhar informações essenciais de outros objetos de domínio (COSTA; HILD, s/d). São classes responsáveis por controlar o ciclo de vida dos outros objetos (Entidades, Objetos de valor, Agregados); concentram operações de criação, alteração e remoção de objetos (CUKIER, 2010).

Um conceito importante para o DDD é o Bounded Context que são delimitadores ou agrupamentos de domínios de um determinado negócio, isso significa que as entidades desse contexto devem ser construídas expressando as intenções do contexto para os quais a entidade venha a pertencer. A maior complexidade vista no desenvolvimento de um sistema é compreender o domínio do problema a ser solucionado e suas regras. Partindo do estudo do domínio, cria-se o modelo, que pode ser definido como uma estrutura de conhecimento facilitado, onde acomoda conhecimentos válidos e evidencia o problema que será introduzido.

O modelo de domínio não atua somente como um diagrama simples, mas sim dá conceito que o diagrama objetiva transferir (COSTA; HILD, s/d). Uma vez que o desenvolvimento permanece evidenciado no modelo de domínio, a etapa principal deve ser o isolamento dos conceitos que correspondem o entendimento desse domínio, sendo o modo preferível para atingir tal isolamento a utilização de uma arquitetura multicamadas (MACEDO, 2009). A viabilidade de isolar responsabilidades distintas em camadas diferenciadas é uma das vantagens de se usar tal arquitetura.

O projeto se beneficia especialmente do isolamento da camada de domínio, possibilitando ao desenvolvedor se centralizar na definição do domínio e em seu relacionamento. O isolamento dessa camada sustenta a totalidade do projeto (MACEDO, 2009). A diferença prevalecente entre o DDD e as demais abordagens de desenvolvimento é o mérito referido a modelagem de domínio, uma vez que esta modelagem e a implementação são tratadas como atividades semelhantes que não progridem mutuamente (COSTA; HILD, s/d).

4 ATIVIDADES DESENVOLVIDAS

Neste Capítulo serão apresentadas as atividades desenvolvidas e as oportunidades experienciadas pelo autor desde o início de sua carreira. O Capítulo segue com a descrição do período de estágio e do período efetivo do autor como desenvolvedor .Net nas seções 4.4 e 4.5.

4.1 Onboarding

No dia 20 de novembro de 2020 o autor deu início ao processo de estágio e na ioasys as primeiras atividades do colaborador é chamado de *Onboarding,* que se refere ao mecanismo adotado pela empresa, pelo qual novos funcionários adquirem o conhecimento, as habilidades e os comportamentos necessários para se tornarem membros da mesma, entender melhor os processos, cultura e ferramentas utilizadas em comum por todos os colaboradores.

As atividades referentes a esse processo são reuniões com lideranças responsável por explicar cada setor da empresa, divisões de trabalho, cultura organizacional e a apresentação da ferramenta *TeamWork*, que é utilizada para a comunicação interna da empresa (*chat* que permite contato com todos os colaboradores), utilizada também como ferramenta para realização de apontamentos de horas trabalhadas, esses apontamentos são fundamentais para que seja contabilizadas as horas dedicadas a cada cliente e consequentemente sejam feitas as devidas cobranças.

A última atividade do processo de *Onboarding* é a apresentação do time que o estagiário vai fazer parte e qual produto irá atuar. Nesse momento são apresentados a estrutura do time, quem ficará responsável pela sua mentoria e quem serão as lideranças mais próximas que poderiam ser acionadas em caso de problemas ou dúvidas.

4.2 Atividades durante o projeto PV Digital

Finalizado o processo de *Onboarding* o autor foi direcionado para a equipe do projeto PV Digital. Inicialmente o autor ficou apenas como suporte à equipe principal dando assistência no desenvolvimento de teste de unidade para funcionalidades já em ambiente produtivo. Porém, devido a saída de um integrante e a boa aderência do autor com o fluxo de

trabalho e a equipe, ele passou a fazer parte do time principal ainda no segundo mês do estágio, desta forma, passou a trabalhar como desenvolvedor backend. A aplicação *backend* foi desenvolvida na plataforma .Net Core, conforme mencionado anteriormente no item 2.1 e detalhado no item 3.4. Criada em uma estrutura de serviços web a aplicação utiliza o framework *ASP.NET Core*, SQL *Server* para o banco de dados e o ORM utilizado foi o *Entity Framework Core*. A função deste serviço *web,* portanto*,* é fornecer APIs para serem integrados com a aplicação *frontend*.

O PV Digital tinha a finalidade inicialmente de importar planilhas, gerando relatórios por meio destes dados. Devido à alta demanda de utilização do sistema, a estrutura código precisou passar por um processo de refatoração. Esse processo foi realizado por um arquiteto, em que a nova estrutura implementada pelo mesmo foi baseada no padrão arquitetural CQRS, no qual o item 3.5.1 descreve detalhadamente sobre o referido padrão, além disso, a construção dos domínios da aplicação foi baseada na abordagem DDD, no qual o item 3.5.2 descreve detalhadamente sobre a abordagem. O autor iniciou o desenvolvimento no projeto após a realização deste processo.

Após a construção da nova estrutura de código, surgiram novas demandas na qual o autor passou a desenvolvê-las. Dentre essas demandas o autor foi responsável por correções de *bug,* novas *features* e integrações com sistemas externos.

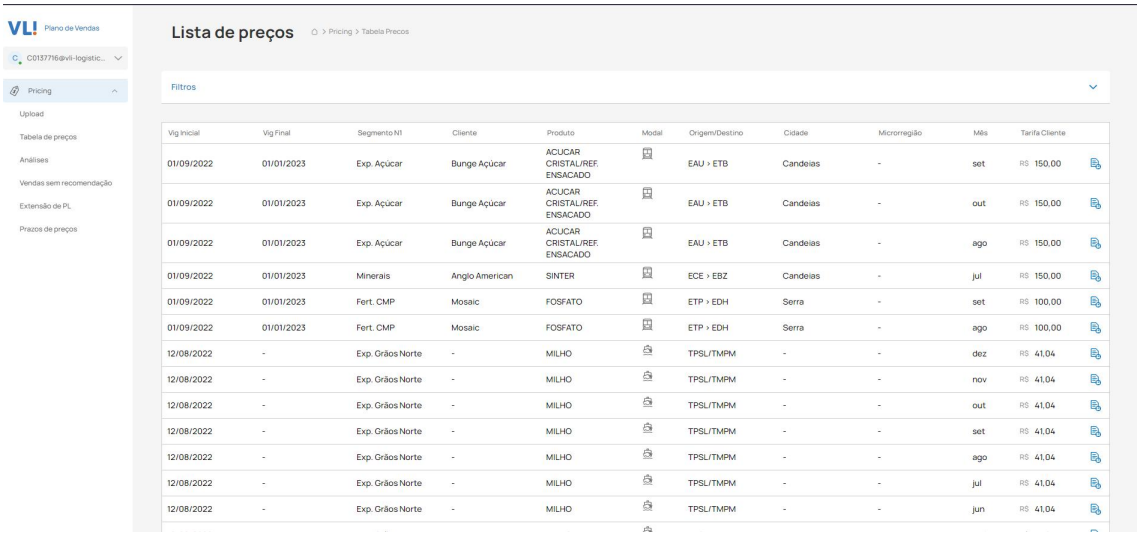
- *BUGs*: Dentre os vários *bugs* solucionados pelo autor, cabe destacar, o mais relevante, que foi uma correção de dados incorretos gerados na produção, causados por uma publicação indevida. A solução foi corrigir o código publicado e fazer um *update* no banco do ambiente para a correção dos dados gerados.
- *FEATURE*: Foram implementadas inúmeras melhorias para os módulos já existentes, além da criação de novos módulos, onde o autor teve participação integral na criação dos módulos: Recomendação de Preço; Demanda; Instrumentos Jurídico.
- INTEGRAÇÕES: O autor também participou do desenvolvimento de integrações com alguns sistemas externos: *Netlex*; *Data Warehouse*; IBGE.

Vale ressaltar que dentre as demandas desenvolvidas pelo autor, as de maior relevância, foram:

- MÓDULO RECOMENDAÇÃO DE PREÇO: O modelo de recomendação foi dividido em duas features: *Upload* planilha (recebe um arquivo contendo os preços sugeridos para

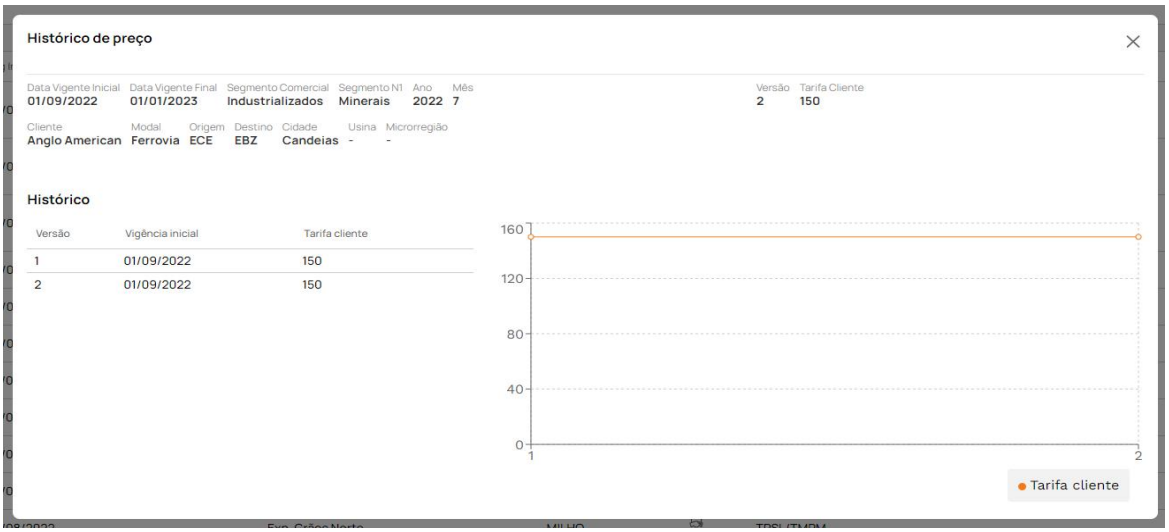
venda dos produtos, armazena no banco de dados e, durante a realização de uma venda o usuário recebe sua referida indicação de preço); Tabela de preços (além da listagem de todas as recomendações sugeridas, o usuário tem acesso ao histórico de recomendações de preços). Na Figura 8 é possível visualizar a pagina principal do módulo Recomendação de Preço. Quanto que na Figura 9 é possível visualizar a modal de histórico de recomendação de preços de um produto específico.

Figura 8: Tela de listagem de preços recomendados do PV Digital.



Fonte: VLI Multimodal

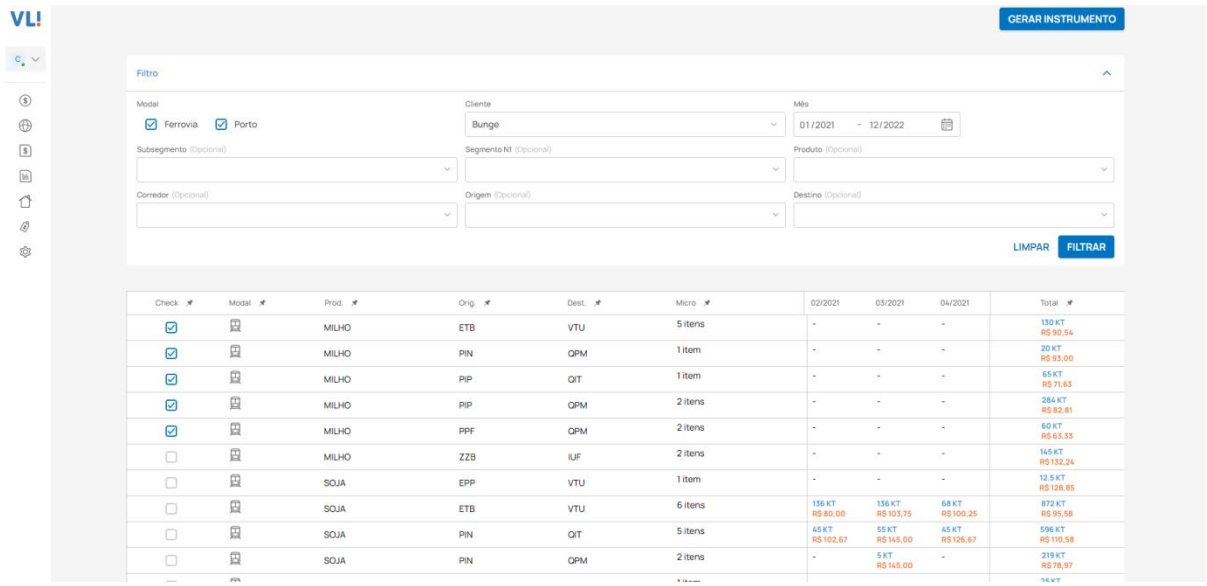
Figura 9: Modal para visualização de histórico de preço recomendado



Fonte: VLI Multimodal.

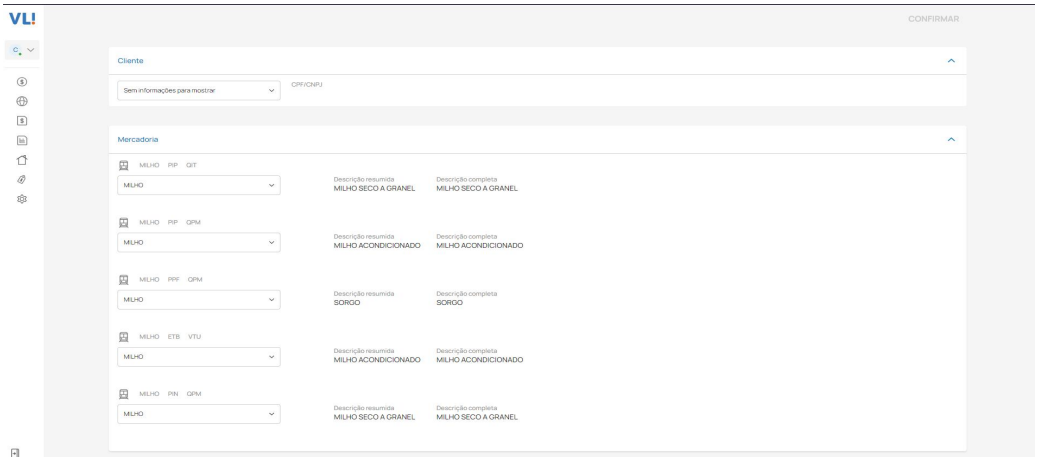
- MÓDULO INSTRUMENTO JURÍDICO: Neste módulo o usuário seleciona as vendas que foram realizadas em um determinado período, para um cliente específico, após um processo de categorização destas vendas a aplicação envia esses dados através de uma integração com o sistema Netlex, gerando o documento jurídico para fins fiscais. Na Figura 10 e exibido a tela de de filtro de venda no módulo Instrumento Jurídico, enquanto que da Figura 11 é possível visualizar a tela de classificação de dados das vedas que serão utilizadas para geração do instrumento jurídico.

Figura 10: Tela principal do módulo Instrumento Jurídico



Fonte: VLI Multimodal

Figura 11: Tela de classificação dos dados da venda para geração do instrumento jurídico.



Fonte: VLI Multimodal

- INTEGRAÇÃO NETLEX: Para o desenvolvimento desta integração foi realizado reuniões de alinhamento entre equipes de desenvolvimento, criação de documentação para integração e o desenvolvimento do código em si. Para demonstração da documentação utilizada entre as equipes dos dois sistemas é possível visualizar o contrato de integração da Figura 12

Figura 12: Imagem da primeira página da documentação para integração com a aplicação *Netlex.*

Fonte: ioasys

4.3 Atividades gerais

Para além das atividades descritas nos itens acima, que são descrições do foi desenvolvido pelo autor, nesta seção serão mostrados os tipos de atividades que foram e continuam sendo executadas pelo autor de forma recorrente, desde o início de sua atuação. Essas atividades são realizadas para seguir um bom fluxo de trabalho, são elas: revisão do código, que é feita através das plataformas Azure DevOps[4]; *one-on-one,* reuniões de feedback com o time de desenvolvimento, realizadas pelos gestores.

A revisão do código é uma tarefa continuamente realizada pelo autor sempre ao final de cada atividade desenvolvida. É criado por parte do desenvolvedor o *pull* ou *merge request*, documento de fácil visualização que reúne as alterações desenvolvidas naquela atividade comparada à *branch* em que se foi baseada. Esse documento é enviado aos outros desenvolvedores de mesma tecnologia para a revisão de boas práticas e correções de erros que possam ter passado despercebidos (VALENTE, 2020).

Na *ioasys*, o processo de revisão de código começa através de um grupo via chat online que reúne os colaboradores de determinada tecnologia para o envio dos *pull requests*. Deve ser enviado um *link* que redireciona o usuário para a plataforma de controle de versão utilizada para o desenvolvimento de determinado projeto, em que nos mostra as diferenças de versões do código alterado. Na análise dessas diferenças de código, devemos analisar se as boas práticas de desenvolvimento estão sendo seguidas e se não temos nenhum *bug* aparente no sistema.

Para um *pull request* estar apto para o *merge*, junção da *branch* desenvolvida com a *branch* original, é necessário que ao menos dois desenvolvedores realizem o *code review* do *pull request* e deixem o seu *approve*, ato que aumenta a qualidade e confiabilidade do código.

Com relação a reunião de *feedback* feita pelos gestores, a *one-on-one, n*a *ioasys*, essas reuniões são realizadas mensalmente e são compostas pelo gestor responsável e pelo colaborador, no qual é adotado um ambiente informal e é fornecido o *feedback* sobre a atuação do colaborador pelo gestor e o *feedback* do colaborador a respeito tanto da empresa e dos processos executados quanto das relações pessoais de convivência do time.

[4] O Azure DevOps fornece serviços de desenvolvedor para permitir que as equipes planejem o trabalho, colaborem no desenvolvimento de código e criem e implantem aplicativos. O Azure DevOps dá suporte a uma cultura colaborativa e um conjunto de processos que reúnem desenvolvedores, gerentes de projetos e colaboradores para desenvolver softwares. (MICROSOFT, 2022)

4.4 Efetivação como desenvolvedor .Net Júnior

Prestes a completar o quarto mês de estágio, no final de março de 2021 foi comunicado ao autor a sua efetivação para desenvolvedor .Net júnior, recompensa de um bom desempenho e boa comunicação sempre realizando um bom trabalho. Portanto, de abril de 2021 até final de janeiro de 2022, o autor continuou atuando no projeto PV Digital, prestando manutenção do código previamente desenvolvido e implementando novas funcionalidades.

4.5 Desenvolvedor .Net Pleno

Após o autor atuar aproximadamente por um ano e quatro meses como desenvolvedor .Net Junior foi proposto a ele, juntamente com a oportunidade de trabalhar em um novo projeto, a sua promoção para desenvolvedor .Net Pleno. Como .Net Pleno o autor hoje é responsável pelo desenvolvimento do projeto do Portal B2B Alpargatas, da empresa Alpargatas.

5 CONSIDERAÇÕES FINAIS

Pode-se afirmar que os objetivos do autor foram alcançados no estágio, uma vez que foi possível aplicar conhecimentos adquiridos no curso de graduação e aprender novas tecnologias, metodologias e frameworks. A aquisição de experiência de mercado e de trabalho em equipe, através da criação de soluções para clientes reais, também configuraram grande contribuição para aperfeiçoamento profissional e pessoal. Desta-se, portanto, os aprendizados de maior importância:

- Trabalhar utilizando o Scrum em uma equipe com papéis bem definidos;
- Entender o passo a passo necessário para criar novas funcionalidades de uma aplicação;
- Criação de *features* para atender as especificidades dos clientes, como: Módulo de Demanda; Módulo de Recomendação de Preços; Integrações entre sistemas externos, como *Netlex*, *DataWarehouse* e IBGE;
- Construção de estrutura para realização de testes de unidade;
- Aprender a utilizar tecnologias consideradas de ponta no mercado e como cada uma delas pode me ajudar;
- Trabalhar com prazos, entregas e responsabilidades;
- Entender a importância de atualizar-se com as tecnologias mais utilizadas no mercado.

O maior desafio enfrentado foi adaptação à dinâmica de trabalho já existente na equipe e o entendimento por parte do autor sobre o papel na equipe. Para solucionar isso, foi possível contar com a ajuda de um mentor, que se dedicou à integração gradual do autor na rotina do time. O autor pode perceber, também, a área de engenharia de software com a qual mais se identifica, sendo está a de Arquitetura de Software, podendo, assim, direcionar seus estudos e buscar experiências profissionais que ampliem cada vez mais seu aprendizado.

O curso proporcionou as ferramentas necessárias para a realização do estágio, não apenas na parte de codificação, mas toda a área que a Engenharia de Software abrange, mais especificamente gerência de projetos, arquitetura, qualidade de software, interação humano-computador, aprendizados essenciais para a realização do estágio. É possível afirmar que este estágio foi de imensa importância para meu desenvolvimento profissional, ajudando a desenvolver uma visão crítica do mercado e ampliando a visão das áreas nas quais almejo investir. Dessa forma, conclui-se que, através dessa experiência, o autor pode considerar-se um profissional muito mais apto a contribuir com qualquer empresa.

REFERÊNCIAS

COSTA, A.; HILD, T. **Estudo da modelagem de software Domain-Driven Design aplicado na fatoração de um sistema informatizado para auxılio de Agentes Comunitários de Saúde**. s/d.

CUKIER, D. **DDD – Introdução a Domain Driven Design**. Daniel Cukier. Agile And Art. Jul. 2010.

DA SILVA, T. A.; FILHO, C. de C. C.; DA SILVA, C. T. M. **Investigação ontológica da obra de arte digital: linguagem ubíqua, modelo de domínio e programação voltada para as artes visuais**. V Simpósio Internacional de Inovação em Mídias Interativas – SIIMI. p. 1-388–416. UFG. Goiânia. 2018.

EVANS, E. **Domain Driven Design: Tackling Complexity in the Heart of Software**. Ed. Addilson-Wesley, 2003.

FERREIRA, G. **O que esconde o CQRS**. Revista Programar. 2016.

FOWLER, M. **CQRS**. 2011.

GIT. **Git**, c2021. Git Branch. Disponível em: https://git-scm.com/docs/git-branch. Acesso em: 15 de jul. de 2022.

GIT. **Git**, c2021. Git Merge. Disponível em: https://git-scm.com/docs/git-merge. Acesso em: 15 de jul. de 2022.

GIT. **Git**, c2021. Git. Disponível em: https://git-scm.com/. Acesso em: 15 de jul. de 2022.

HIPPCHEN, B.; GIESSLER, P.; STEINEGGER, R. H.; SCHNEIDER,M; ABECK, S. **Designing Microservice-Based Applications by Using a Domain-Driven Design Approach**. International Journal on Advances in Software , v. 10, n. 3 e 4, pág. 432- 445, 2018.

LESS FRAMEWORK. **Product Backlog Refinement** 2022. Disponível em: https://less.works/less/framework/product-backlog-refinement. Acesso em: 28 ago. 2022.

MACEDO, O. **Diretrizes para desenvolvimento de linhas de produtos de software com base em Domain-Driven Design e métodos ágeis**. 2009. Tese de Doutorado. Universidade de São Paulo. 2009.

MATTOS, K. M.; DOLL, L. M.; ALMEIDA, J. **DOMAIN-DRIVEN DESIGN E TESTDRIVEN DEVELOPMENT**. 5º Encontro de Engenharia e Tecnologia dos Campos Gerais. 2010.

MICROSOFT. **Azure DevOps**, c2022. Azure DevOps. Disponível em https://docs.microsoft.com/pt-br/azure/devops/user-guide/what-is-azure-

devops?toc=%2Fazure%2Fdevops%2Fget-started%2Ftoc.json&bc=%2Fazure%2Fdevops%2Fget-started%2Fbreadcrumb%2Ftoc.json&view=azure-devops. Acesso em: 15 de jul. de 2022

MICROSOFT. **DotNet Microsoft**, c2021. What is .NET Framework? Disponível em: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework. Acesso em: 15 de jul. de 2022.

MICROSOFT. **DotNet Microsoft**, c2021. What is .NET? Disponível em: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet. Acesso em: 15 de jul. de 2022.

MICROSOFT. Padrão CQRS: Segregação de responsabilidade de consulta e comando. Microsoft.

MOZILLA. **MDN Web Docs**, 2021. API. Disponível em: https://developer.mozilla.org/en-US/docs/Glossary/API. Acesso em: 15 de jul. de 2022.

MOZILLA. **MDN Web Docs**, 2021. HTTP. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP. Acesso em: 15 de jul. de 2022.

NASCIMENTO, W. CQS — Command Query Separation. 2019.

OSTROFF, **Frank**. The Horizontal Organization: what the organization of the future actually looks like and how it delivers value to customers. Oxford University Press, 1999.

PASSOS, C. CQRS — Command Query Responsability Segregation. Medium. 2018.
PIRES, E. CQRS o que é e onde aplicar. 2016.

SANTOS, E. Integração da abordagem Domain-Driven Design e de técnica Behaviour-Driven Development no desenvolvimento de aplicações web. 2015.

SCHWABER, Ken; SUTHERLAND Jeff. **O Guia do Scrum**. Disponível em: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-PortugueseBR-2.0.pdf. Acesso em: 28 de ago. de 2021.

SCRUMGUIDES. **Scrum Guides**, c2020. Scrum Guides. Disponível em: https://scrumguides.org/index.html. Acesso em: 15 de jul. de 2022.

VALENTE, Marco Tulio. **Engenharia de Software Moderna**. Disponível em: https://engsoftmoderna.info/. Acesso em: 15 de jul. de 2022.

VLI LOGÍSTICA. **VLI Logística**, 2022. Quem Somos. Disponível em: https://www.vli-logistica.com.br/quem-somos/ Acesso em? 20 de ago 2022.

YOUNG, G. CQRS Documents by Greg Young. 2010.