
**GIOVANE AGUIAR DE ALMEIDA**

# **RELATÓRIO DE ESTÁGIO**:

DESENVOLVIMENTO FULLSTACK NA DELTA GLOBAL

**LAVRAS – MG**

**2022**

**GIOVANE AGUIAR DE ALMEIDA**

# **RELATÓRIO DE ESTÁGIO**:
# DESENVOLVIMENTO FULLSTACK NA DELTA GLOBAL

Relatório apresentado à Universidade Federal de Lavras, como parte das exigências da Graduação para obtenção do título de Bacharel em Sistemas de Informação.

Prof. DSc. Dilson Lucas Pereira
Orientador

**LAVRAS – MG**

**2022**

*Dedico este trabalho à minha família, que me apoiou em todo o decorrer do curso, com palavras de ânimo e coragem, que me auxiliaram e me mantiveram no caminho correto.*

## AGRADECIMENTOS

Agradeço, em primeiro lugar, a Deus, que me deu a força e a saúde necessárias, que me possibilitaram essa conquista. Aos meus amigos e familiares, pelo apoio com palavras de ânimo e todo tipo de auxílio durante a graduação, que tanto contribuiu para o meu aprendizado. Gostaria de agradecer também à Universidade Federal de Lavras, essencial no processo da minha formação profissional e que me apresentou tantas oportunidades de aprendizado e amadurecimento. Agradeço ao professor DSc. Dilson Lucas Pereira, por ter se disponibilizado a ser meu orientador, sempre muito solícito e prestativo, me auxiliando e esclarecendo todas as minhas dúvidas.

# RESUMO

Com a constante evolução do mercado automobilístico e a expansão das frotas veiculares de empresas especializadas em transporte e logística, naturalmente, a demanda por serviços de monitoramento e assistência veicular também cresceram. O desenvolvimento desse cenário gerou o ambiente propício para que as organizações criassem suas soluções e tentassem conquistar o mercado. Dessa maneira, a Delta Global desenvolveu seus próprios serviços para atenderem a esse mercado crescente. O objetivo deste relatório é descrever as principais atividades do estágiário na empresa Delta Global SA, atuando como desenvolvedor de software fullstack, trabalhando na modelagem dos bancos de dados, nos desenvolvimentos de API's e na implementação de interfaces para o usuário, com tecnologias como Javascript, ReactJS, PHP, CodeIgniter e MySQL.

**Palavras-chave:** Software. Arquitetura. Fullstack. Front-end. Back-end. Javascript. ReactJs. PHP. CodeIgniter. MySQL.

# ABSTRACT

With the constant evolution of the automobile market and the expansion of vehicular fleets of companies especialized in logistics and transport, naturally, the need for assistance and monitoring services also increased. This scenario's development created a propicious environment so that companies could develop their solutions and attempt to conquer the market. Thus, Delta Global developed their own services to attend to this growning markert. The purpose of this report is to describe the main activities as an intern at Delta Global SA, acting as a fullstack software developer, working on designing databases, API development and implementing user interfaces, using technologies like Javascript, ReactJs, PHP, CodeIgniter, Nodejs and MySQL.



**Keywords:** Software. Architecture. Fullstack. Front-end. Back-end. Javascript. ReactJs. PHP. CodeIgniter. MySQL.

## LISTA DE FIGURAS

**SUMÁRIO**

# 1 INTRODUÇÃO

A Delta Global S.A é uma empresa que fornece serviços para o mercado de seguros na América Latina, por meio da tecnologia. A sede da empresa está localizada em Porto Alegre, no Rio Grande do Sul, porém a Delta possui sua própria fábrica de software localizada em Lavras, Minas Gerais, onde a maioria dos desenvolvedores atuam.

O serviço carro chefe da empresa é o Delta Assist, uma plataforma de assistência veicular que funciona vinte e quatro horas por dia, sete dias por semana. Essa plataforma conta com aproximadamente seis mil prestadores de serviço de assistência localizados em todo o Brasil, permitindo com que a Delta consiga atender a grande parte do território nacional de maneira ágil. O Delta Assist possui operadores sempre a postos para atender a possíveis sinistros que venham a ocorrer com os clientes do serviço, abrindo chamados e enviando prestadores para o local onde esse cliente se encontra. Outro projeto da empresa é o Historicar. Por meio dele, clientes podem realizar consultas às bases de dados do projeto para checarem o histórico do veículo, se possuí alguma pendência, entre outras possibilidades.

O estágio na Delta teve início em 11/01/2022 e foi motivado por boas recomendações sobre a empresa e pela oportunidade de aprendizado e crescimento profissional. A carga horária do estágio é de trinta horas semanais, onde o estagiário atuou como desenvolvedor fullstack, ou seja, trabalhando tanto no desenvolvimento do projeto na parte do backend implementando as regras de negócio e trabalhando com bancos de dados, quanto do frontend ao implementar interfaces e exibir as informações no navegador do usuário.

Este trabalho consiste em reportar as principais atividades desenvolvidas no projeto Delta Fleet, um sistema de gerenciamento de frotas veiculares que oferece suporte a rastreamento em tempo real, controle de jornadas, relatórios de abastecimentos, e cálculos de telemetria. Por meio do sistema, um cliente que contratou os serviços do Delta Fleet pode acompanhar o posicionamento e os percursos que seus veículos estão fazendo, não só na data do acesso, mas também de dias anteriores. Além disso, é possível configurar alertas de diversos tipos para um conjunto de veículos, esses alertas podem ser de geo-cercas, velocidade e nível de bateria.

Além deste capítulo introdutório, este documento está organizado da seguinte maneira.

- O capítulo dois especifica as principais tecnologias e ferramentas utilizadas durante as atividades do estagiário.
- O capítulo três descreve as atividades realizadas durante o estágio.

- O capítulo cinco descreve as considerações finais sobre o estágio.

# 2 TECNOLOGIAS UTILIZADAS

Este capítulo apresenta as tecnologias utilizadas diariamente, durante o desenvolvimento do projeto.

## 2.1 GIT

Em desenvolvimento de software o versionamento de código é um conceito bastante utilizado. Esse versionamento registra toda a evolução do projeto, cada alteração sobre cada arquivo. Dessa maneira, é possível saber quem fez o que, quando e onde (DIAS, 2016). Além disso, é possível também criar vários ramos de desenvolvimento em paralelo de forma que vários desenvolvedores conseguem atuar em diferentes funcionalidades sem alterar o ramo principal e estável do software. Esse controle da linha do tempo e dos ramos do projeto são fundamentais para a segurança do mesmo, facilitando a correção de eventuais problemas .

Atualmente existem algumas opções de ferramentas de versionamento de código disponíveis, como por exemplo, o CVS, Subversion e TFS e, a utilizada na Delta é o GIT. Essa é uma ferramenta gratuita e de código aberto, utilizada em projetos pequenos e grandes, que possibilita a quem a utiliza versionar o código que está sendo desenvolvido. Dessa forma, é possível criar pontos de verificação, `branches` ou ramos para separar o código em desenvolvimento do código estável, além de dar mais segurança aos desenvolvedores caso algum erro seja cometido.

O GIT é uma das primeiras ferramentas necessárias para a instalação do projeto. É por meio dele que o desenvolvedor clona o repositório que mantém o código fonte da aplicação. No contexto da Delta o fluxo de trabalho com essa ferramenta ocorre da seguinte maneira: ao iniciar o dia é buscada do repositório a última versão do código atualizado da `branch` de desenvolvimento, por meio do comando `git pull origin nome_da_branch`.

Nesse processo de atualização, eventualmente ocorrem conflitos de código. Conflitos acontecem quando dois desenvolvedores trabalham simultaneamente em uma mesma parte do projeto, realizando alterações nas mesmas linhas do código. Como o código possui duas versões diferentes, o sistema de versionamento cria um conflito que precisará ser resolvido. Caso ocorra algum conflito de código nessa etapa de atualização é necessário fazer uma revisão das alterações e corrigí-lo. Após a atualização do projeto local com o repositório remoto, para desenvolver uma nova funcionalidade ou corrigir algum problema no código, uma nova `branch` é criada utilizando o comando `git checkout -b nome_da_branch`. Esse comando faz dois

em um, cria uma `branch` com o nome escolhido e já aponta o projeto para essa nova `branch` criada.

Durante o desenvolvimento da funcionalidade ou correção, para criar os pontos de verificação do código são utilizados os comando `git add`, para adicionar as mudanças realizadas na área de `staging` e, logo em seguida, `git commit -m`. A opção `-m` é utilizada para inserir uma mensagem geralmente utilizada para descrever o que o `commit` está alterando no código. Após finalizado o desenvolvimento e atualizado o repositório remoto com o comando `git push`, é realizado o `pull request`. Este é uma solicitação criada para que o desenvolvimento ou correções realizadas sejam mescladas à versão estável do código. Para isso é necessário adicionar um revisor à solicitação para que o código seja lido e revisado, reduzindo a chance da atualização quebrar o código já funcional.

## 2.2 BITBUCKET

O Bitbucket é uma plataforma online e gratuita para a hospedagem de repositórios de projetos, existem outras como Github e o Gitlab. Essas ferramentas são utilizadas amplamente por empresas que desejam versionar seus projetos, utilizando as diversas ferramentas disponíveis no sistema, como por exemplo, a criação de `pull requests`, `issues`, entre outras.

O Bitbucket, além de disponibilizar ferramentas que facilitam a revisão de código por meio dos pull requests, também disponibiliza `Pipelines` para desenvolver, testar e disponibilizar seu projeto. Por esses e outros motivos, como a integração com outras plataformas como o Trello, o Bitbucket foi o sistema escolhido pela Delta Global para hospedar e manter o projeto por meio de suas ferramentas disponíveis.

## 2.3 MYSQL

O programa MySQL é um sistema de gerenciamento de banco de dados relacional que utiliza a linguagem de consulta estruturada SQL como interface de acesso e extração de informações do banco de dados em uso. O MySQL é um dos sistemas de gerenciamento de bancos de dados mais populares e usados no mundo. É rápido, multitarefa e multiusuário (MANZANO, 2011). Utilizando a arquitetura do tipo cliente-servidor, é possível realizar as seguintes operações:

- Consulta de dados (`Data Query`): Solicitar informações específicas de uma base de dados.
- Manipulação de dados (Data Manipulation): É possível inserir, alterar, ordenar e excluir dados, além de diversas outras operações.
- Identidade de dados (`Data Identity`): Definir tipos de dados. É possível definir relações entre tabelas.
- Controle de acesso dos dados (`Data Accesss Control`): Por meio do sistema gerenciador de bancos de dados é possível definir quem tem acesso às informações armazenadas, protegendo a autenticidade dos dados.

A Delta usa o MySQL como seu sistema gerenciador de bancos de dados padrão no dia a dia, utilizando a ferramenta para a criação de novas tabelas, procedimentos para facilitar a busca de dados específicos, persistência de dados, entre outros.

## 2.4 APPLICATION PROGRAMMING INTERFACE (API)

Uma API possibilita que empresas exponham os dados e funcionalidades de suas aplicações para desenvolvedores externos, parceiros de negócio e departamentos internos de suas organizações (EDUCATION, 2020). O objetivo principal de uma API é disponibilizar recursos de uma aplicação, abstraindo os detalhes da implementação e restringindo o acesso a tais recurosos por meio de regras específicas (VENTURA, 2015).

É muito comum essas interfaces de programação implementarem as operações conhecidas como CRUD (*Create*, *Read*, *Update*, *Delete*). A operação *Create* é utilizada quando se deseja criar um novo recurso na aplicação, de tal maneira que depois possamos acessá-lo. A operação *Read* é utilizada para buscar os recursos salvos dessa aplicação, geralmente por meio de seleções em bancos de dados. A operação *Update* é utilizada quando se deseja alterar ou modificar algo recurso da aplicação e, a *Delete* quando se deseja apagar algum recurso.

## 2.5 PHP

A linguagem PHP possibilita a criação de programas confiáveis e de alta complexidade, capazes de satisfazer a qualquer tipo de tarefa envolvida no desenvolvimento de um sistema (SARAIVA MAURíCIO DE O.; BARRETO, 2018). Pode ser utilizada juntamente com

o HTML para a criação de sistemas web e sites dinâmicos. Uma das grandes vantagens desta linguagem é a rápida curva de aprendizado, o que facilita a familiarização com a ferramenta, especialmente para desenvolvedores iniciantes.

Ser uma linguagem bem difundida no contexto de desenvolvimento, estar disponível desde 1995, e possuir uma grande comunidade de desenvolvedores são atributos que fazem do PHP uma boa ferramenta para utilização. Dessa maneira, é possível encontrar online uma ampla gama de bibliotecas já existentes, criadas com o intuito de agilizar o desenvolvimento de aplicações, além de ser fácil solucionar quaisquer problemas que se apresentem durante o desenvolvimento.

Na Delta, o PHP é utilizado como a linguagem principal para a criação e manutenção das API's (`Application Programming Interface`). Dessa forma, grande parte das rotas, também conhecidas como URL's (`Uniform Resource Locator`), que acessam os recursos dos sistemas estão escritas nessa linguagem que trata das regras de negócio e, aliada ao MySQL, realiza operações de consulta, criação, alteração e deleção de dados.

## 2.6 ARQUITETURA MVC (MODEL VIEW CONTROLLER)

MVC é o acrônimo para `Model View Controller` (Modelo Visão Controlador), uma importante e conhecida estratégia para implementação de sistemas computacionais baseada na separação de responsabilidades (RIBEIRO, 2013). O MVC é dividido em:

- `Model`:representam conhecimento. É uma representação de uma abstração na forma de dados computacionais (normalmente bytes).
- `View`: é uma representação visual do modelo. Várias views podem representar o mesmo `model`, cada uma priorizando determinado conjunto de atributos, por exemplo. A `view` recupera informações do model realizando "perguntas"a ele. A `view` pode, inclusive, atualizar o `model`, mandando as mensagens apropriadas. E a `view` precisa conhecer a semântica de cada informação devolvida pelo `view`.
- `Controller`: é uma ligação entre o usuário e o sistema. Ele arranja as `view` na tela. Ele provê formas do usuário se manifestar, através de apresentação de menus ou outros elementos. Ele recebe a interação do usuário, traduz ela e passa a diante (para o `model` ou `view`).

## 2.7 CODEIGNITER

O CodeIgniter é um framework versátil e leve possibilitando a construção de aplicações e sistemas sob o paradigma da orientação a objetos e seguindo o padrão arquitetônico MVC

(BASTOS, 2011). Além dele, existem diversas outras disponíveis no mercado, como o Laravel, Symfony, Zend e CakePHP. Por padrão esse framework traz consigo um conjunto de bibliotecas já instaladas e arquivos que contém modelos de configuração, o que suaviza a curva de tempo necessária para colocar o projeto em funcionamento e possibilita ao desenvolvedor manter o foco no desenvolvimento do mesmo. Essa ferramenta proporciona vantagens para quem a utiliza, como por exemplo:

- Ser de fácil aprendizado e acesso;
- Seu uso é adequado tanto para aplicações pequenas quanto para grandes sistemas;
- Trabalha com o padrão de arquitetura MVC;
- Tem uma documentação bem elaborada.

Na Delta, o projeto Delta Fleet, assim como outros, são construídos utilizando o CodeIgniter.

## 2.8 APACHE HTTP SERVER

O Apache é um servidor web de código aberto amplamente utilizado para a hospedagem de sistemas web, responsável por 46% de todos os sites hospedados. Por meio dele é possível que donos de sites e sistemas web disponibilizem seus conteúdos para os usuários que acessam tais sistemas. É um dos servidores mais confiáveis e um dos que têm mais tempo de mercado também, com seu lançamento em 1995 (L., 2021).

Com o Apache é possível hospedar uma aplicação, funcionando como um intermediário entre os clientes que desejam acessar o conteúdo e a máquina onde os arquivos estão salvos. Quando um cliente pede acesso a um conteúdo, a função do Apache é buscar esse conteúdo na máquina e devolvê-lo para tal cliente. Por ser multiplataforma, esse software pode ser utilizado tanto em sistemas operacionais Unix quanto em Windows.

O Apache é utilizado pelo estagiário para disponibilizar um servidor web em sua máquina local, possibilitando que o mesmo possa desenvolver as demandas e executar o projeto localmente.

## 2.9 NGINX

O Nginx é um servidor web usado para vários propósitos, que vem sendo bastante utilizado. É a escolha de muitos desenvolvedores e empresas por possuir uma arquitetura simples,

porém escalável, de fácil configuração e que utiliza pouco recurso computacional (KHOLODKOV, 2015). Além disso, esse servidor web disponibiliza várias funcionalidades úteis como configuração de cache e muitas outras. Por esses motivos, a Delta utiliza o Nginx como servidor web para servir suas aplicações.

## 2.10 HTML (HYPERTEXT MARKUP LANGUAGE)

HTML (Linguagem de Marcação de HiperTexto) é o bloco de construção mais básico da web. Define o significado e a estrutura do conteúdo da web. Outras tecnologias, além do HTML, geralmente são usadas para descrever a aparência/apresentação (CSS) ou a funcionalidade/comportamento (JavaScript) de uma página da web (DOCS, 2021b).

O termo "Hipertexto" faz referência aos endereços que são encontrados em um documento HTML que identificam outras páginas na web. Esses endereços são uma parte fundamental nas páginas web já que é por meio deles que é feita a navegação. A "Marcação" são elementos utilizados para anotar textos, vídeos, imagens e até mesmo outras páginas. Podemos citar por exemplo os elementos `<div>`, `<p>`,`<img>`, `<video>`, `<audio>` e vários outros essenciais para a modelagem de um documento HTML.

A Delta utiliza os documentos HTML para construir suas páginas web para a visualização dos usuários.

## 2.11 CSS (CASCANDIG STYLE SHEET)

CSS (Cascading Style Sheets ou Folhas de Estilo em Cascata) é uma linguagem de estilo usada para descrever a apresentação de um documento escrito em HTML ou em XML. O CSS descreve como elementos são mostrados na tela, no papel, na fala ou em outras mídias (DOCS, 2021a).

Ao final da composição de todos esses elementos, terá-se uma página web pronta para ser exibida para um usuário, porém essa página não será muito agradável de se ver, nem de se utilizar já que não possuirá nenhuma estilização que facilite a identificação de elementos, nem terá programados os eventos de interação do usuário com a página. O CSS melhorará a interface do usuário, alterando a forma, cor e outros aspectos dos elementos HTML, de maneira que fiquem mais fáceis de serem identificados.

Portanto, por meio do CSS é possível moldar a interface da maneira que se julgar mais adequada.

A Delta utiliza o CSS para estilizar os elementos HTML, utilizados em suas interfaces.

## 2.12 JAVASCRIPT

O Javascript é uma linguagem de programação leve e interpretada, orientada a objetos (DOCS, 2021c). Nasceu, inicialmente, para responder às interações dos usuários em sistemas web. Apesar disso, com o passar dos anos o Javascript evoluiu e passou a ser utilizado em bastantes ambientes diferentes, tais como o Node.js e o React Native para aplicações mobile. Sendo uma das linguagens mais utilizadas no mundo, possui um arcabouço de bibliotecas e ferramentas disponíveis para o uso, que facilitam e agilizam muito o desenvolvimento de aplicações.

Na Delta o javascript é utilizado nos projetos para programar os eventos que irão responder às interações dos usuários em todos os projetos.

## 2.13 REACTJS

O React é uma biblioteca Javascript para construção de interfaces de usuário. Por meio dele é possível criar interfaces interativas de maneira fácil (REACT, 2022). O React é baseado em componentes que gerenciam o próprio estado, combinados para montar uma interface complexa. Utiliza o JSX (`Javascript XML`) para facilitar a manipulação dos objetos na interface, assim como a exibição de dados.

Além disso, essa biblioteca pode ser utilizada tanto no servidor, utilizando Node, quanto em dispositivos móveis por meio do React Native. Existem outras opções no mercado utilizadas para a criação de interfaces visuais, como o Angularjs e o Vuejs. Por ter essa característica de funcionar em diversos ambientes, possuir uma grande comunidade e ser uma tecnologia atual, o React foi escolhido para o projeto Delta Fleet para a criação de interfaces de usuário tanto no navegador quanto nos dispositivos móveis.

## 2.14 NODE.JS

O Node.js é um ambiente de servidor gratuito e de código aberto, que possibilita a criação de aplicações no lado servidor (`server-side`) com algumas vantagens em relação aos

seus principais concorrentes (como PHP ou Java, por exemplo), no que se refere a leveza, flexibilidade, suporte e produtividade (OLIVEIRA CLáUDIO LUíS, 2021) . Foi desenvolvido para suportar aplicações web escaláveis. Ele funciona por meio de um `Event Loop`, que o permite executar operações não-bloqueantes, mesmo o Javascript sendo uma linguagem `single thread`.

O Node é amplamente utilizado por empresas na construção de suas aplicações que funcionam no lado do servidor, criando rotas de acesso a recursos, também conhecidas como URL's (`Uniform Resource Locator`). O Node também pode ser utilizado para outras situações, como `Web Scraping` ou mineração de dados, processamentos no lado do servidor, entre outros. Esse ambiente de execução oferece nativamente um conjunto de bibliotecas utilitárias que facilitam o desenvolvimento de aplicações, tais como funções para criação de servidores HTTP, UDP, TCP, entre outros, funções para manipulação de arquivos e navegação nos diretórios da máquina, funções para trabalhar com o sistema operacional da máquina, processos e muito mais.

## 2.15 AXIOS

Axios é um cliente HTTP baseado-em-promessas para o node.js e para o navegador. É isomórfico, ou seja, pode rodar no navegador e no node.js com a mesma base de código. No lado do servidor usa o código nativo do node.js - o modulo http, enquanto no lado do cliente (navegador) usa XMLHttpRequests (AXIOS, 2020). Existem outras, como por exemplo, a Fetch API, nativa do Javascript porém que só funciona nos navegadores. Por ser de código aberto, funcionar tanto no lado do servidor quanto do navegador, e possuir uma boa documentação, a biblioteca Axios foi escolhida para ser o cliente HTTP no projeto DeltaFleet.

## 2.16 RABBITMQ

O RabbitMQ é um corretor de mensagens que implementa o Protocolo de Enfileiramento de Mensagens Avançado (AMQP). Este protocolo é utilizado para padronizar sistemas de troca de mensagens e suas interações, como publicar e consumir mensagens das filas (DOSSOT, 2014). O RabbitMQ é um dos vários serviços de mensageria disponíveis no mercado. Por ser leve, multiplataforma e ser fácil de configurar, ele é um dos serviços mais utilizados para tratamento de filas no mundo do desenvolvimento de software (DOSSOT, 2014).

Serviços de mensageria são servidores de mensagens. Eles são responsáveis por enfileirar mensagens e armazená-las em disco, caso esteja configurado para tal, de maneira que quando algum cliente queira consumir tal mensagem, ela esteja disponível. Pode-se fazer uma analogia com uma caixa de correio, as mensagens enfileiradas são as cartas que serão depositadas (inseridas na fila), elas permanecerão lá até que alguém remova (consuma a mensagem da fila) a carta da caixa. Esse tipo de serviço melhora a escalabilidade da aplicação, distribuindo a carga dos processamentos de dados pela qual a mesma é responsável.

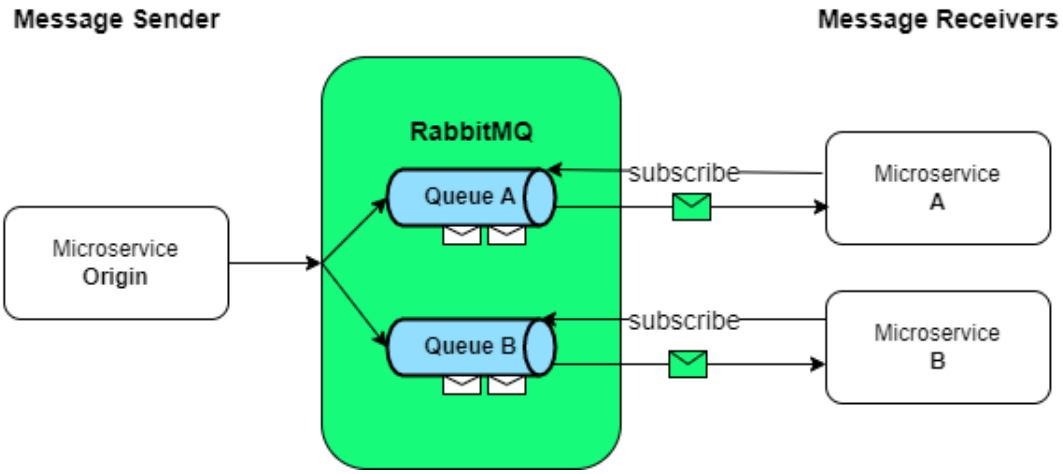
## 2.17 MICROSSERVIÇOS

Segundo a IBM, microsserviço é um estilo arquitetural, no qual grandes aplicações de software são compostas por um ou mais serviços menores. Microsserviços podem ser disponibilizados de maneira independente uns dos outros e possuem baixo acoplamento (REDBOOKS, 2015). Esse modelo arquitetônico é bastante utilizado quando se quer produzir aplicações de software com alta disponibilidade, alta performance e escalabilidade.

A Figura 2.1 exibe, de maneira simplificada, o funcionamento do RabbitMQ. A região hachurada em verde claro representa o `broker`, camada responsável por orquestrar todas as filas de mensagens. o `Microservice Origin` representa o `publisher`, a aplicação que publica as mensagens nas filas para que os microsserviços `A` e `B`, os `consumers`, possam consumir tais mensagens.

Na Delta utiliza algumas características desse modelo, que serão abordadadas na seção 3.5.

Figura 2.1 – Funcionamento básico do RabbitMQ




Fonte: elaborada pelo autor, baseada em
https://docs.microsoft.com/pt-br/dotnet/architecture/microservices/multi-container-microservice-net-applications/rabbitmq-event-bus-development-test-environment

# 3 ATIVIDADES REALIZADAS

Como estagiário em desenvolvimento de `fullstack`, as atribuições se iniciam na modelagem do banco de dados até a integração dos dados no `frontend`. Assim que uma demanda é designada ao estagiário, inicialmente, o mesmo deve conversar com os analistas do projeto para garantir que todas as funcionalidades demandadas são possíveis de serem construídas.

Na Delta, uma funcionalidade geralmente surge a pedido de clientes ou de uma nova ideia desenvolvida internamente. Com os requisitos definidos e discutidos com os líderes técnicos, a demanda é atribuída ao programador que será reponsável por desenvolvê-la. Depois de implementada, a demanda será disponibilizada no ambiente de homologação, para que os analistas possam testá-las e buscar eventuais problemas de implementação. A após ser testada, é então disponibilizada no ambiente de produção para que os clientes possam utilizá-la.

As subseções seguintes discutem tarefas específicas realizadas pelo estagiário.

## 3.1 MODELAGEM DO BANCO DE DADOS

Depois de feito o alinhamento da demanda o próximo passo é pensar na modelagem do banco de dados MySQL. Nesse momento é necessário estudar as relações entre as entidades, para definir se será necessária a criação de uma nova relação, ou se basta adicionar colunas às já existentes. Caso seja necessário modificar a arquitetura existente no banco de dados, é necessário consultar o líder técnico do projeto a fim de validar as alterações a serem feitas. Caso tais alterações sejam aprovadas, eu mesmo as realizo, geralmente acompanhado de um desenvolvedor mais experiente ou do próprio líder técnico. Também é necessário definir se a demanda exigirá a criação de procedimentos no banco, com o intuito de melhorar a performance das consultas ou alterações que ocorrerão durante o uso da nova funcionalidade. Esse tipo de procedimento no banco, no contexto da Delta, é utilizado principalmente na geração de relatórios no sistema, sendo pouco utilizado em telas de cadastro e edição de funcionalidades. O cliente SQL que eu utilizo para trabalhar no banco de dados é o DBeaver, uma ferramenta completa e que suporta diversos tipos de bases de dados SQL.

## 3.2 O DESENVOLVIMENTO DA API (APPLICATION PROGRAMMING INTERFACE)

Finalizada a modelagem do banco de dados é hora de planejar a criação da API. O projeto Delta Fleet utiliza como tecnologia `backend` a ferramenta Codeigniter, dessa maneira

a arquitetura segue o padrão de projeto `Model View Controller`. Antes de começar o desenvolvimento da funcionalidade, é necessário nos certificarmos de que o Apache está ligado corretamente, de maneira a servir o projeto localmente em nossa máquina. Além disso, é necessário também iniciar a aplicação ReactJs com o comando `npm start`. O ponto de acesso da aplicação são as Controllers, arquivos responsáveis por definir as rotas da API e invocar as *Models* para que essas tratem das regras de negócio. Caso a nova funcionalidade exija uma tela de CRUD (*Create*, *Read*, *Update*, *Delete*) será necessária a implementação dessas operações na `Model`, a camada responsável por encapsular toda a lógica e regras de negócio envolvidas.

### 3.2.1 IMPLEMENTAÇÃO DAS MODELS

A `Model` também é responsável por invocar a camada do Codeigniter responsável pela interação com o banco de dados, o `Query Builder`. Na Figura 3.1 podemos observar como é simples montar uma consulta com o `Query Builder` usando apenas a linguagem PHP, já que o mesmo encapsula a interação com o driver nativo do banco de dados. O código da Figura seleciona todos as colunas da relação `customer`. É possível filtrar apenas por um `customer` específico, utilizando o parâmetro `$customer_id`

Figura 3.1 – Exemplo de uma Model utilizando o Query Builder

```
public function get_customers($customer_id = NULL)
{
    $this->db->select('*');
    $this->db->from('customer');

    if ($customer_id) {
        $this->db->where('customer_id', $customer_id);
    }

    $customers = $this->db->get();

    if (count($customers)) {
        return $customers->result();
    }

    return [];
}
```

Fonte: elaborada pelo autor

Após realizadas todas as validações de tipagem dos dados recebidos, verificar se alguma informação está faltando e garantir que tudo está de acordo com as regras de negócio definidas,

o Query Builder irá realizar a comunicação com a base de dados, inserindo novos registros, buscando, atualizando ou deletando os já existentes. Depois de realizar todos os processamentos, consultas e alterações e salvar os dados no banco, a `Model` devolverá uma resposta à `Controller` que devolverá a resposta ao cliente que requisitou a informação.

### 3.2.2 IMPLEMENTAÇÃO DAS ROTAS (CONTROLLERS)

As funções das `Controllers` são limitadas a expor o endereço de acesso a um recurso, validar se o cliente está logado corretamente, invocar as `Models` e devolver a resposta do processamento ao cliente que realizou a requisição.

No Codeigniter, antes de definir uma rota com os métodos HTTP do tipo GET, POST, UPDATE e DELETE para acessar e modificar os recursos de uma entidade, é necessário declarar o método OPTIONS. Esse método é utilizado para retornar metadados da API explicando a quem o chama quais são os métodos disponíveis para o recurso que está sendo acessado.

Figura 3.2 – Exemplo de um método HTTP do tipo OPTIONS

```
public function index_options()
{
    $this->response(NULL, 200);
}
```

Fonte: elaborada pelo autor

O próximo passo é implementar a rota. Para isso, de forma análoga ao exemplo da Figura 3.2, a rota é definida pelo nome da entidade que se deseja acessar seguida do verbo HTTP mais adequado para o tipo de operação que se deseja realizar. A função `index_options`, na Figura 3.2, simplesmente retorna uma resposta ao cliente que realizou a requisição, passando uma mensagem `NULL`, no primeiro parâmetro e o código de status da resposta, no segundo parâmetro.

Utilizando como exemplo a entidade Abastecimento e seguindo o padrão de desenvolvimento da Delta, ao buscar dados de clientes o nome da rota poderia ser definido como `index_get`. Iniciando a `Controller` geralmente a primeira coisa a se fazer é validar se a requisição está sendo feita por um usuário autenticado previamente no sistema. Para isso, é necessário buscarmos no cabeçalho da requisição HTTP, o cabeçalho `Authorization`. Esse é responsável por trazer o `token` que identifica o usuário autenticado na aplicação, e possui o formato `Bearer`. Caso este cabeçalho esteja vazio, ou o `token` enviado seja inválido, devemos

negar o acesso ao cliente que está buscando essas informações pois suas credenciais não estão validadas pela aplicação.

Validado o login do usuário, devemos invocar a `Model` passando a ela a responsabilidade de tratar e validar os dados enviados na requisição. Geralmente o trecho de código responsável por invocar a `Model` é encapsulado em uma estrutura `try catch`, onde é possível escolher a forma de tratamento de quaisquer eventuais erros. Após o retorno da `Model`, a informação é devolvida ao cliente por meio da resposta HTTP.

Figura 3.3 – Exemplo de rota para buscar abastecimentos

```
public function index_get($filtros = NULL)
{
    $usuario_id = $this->valida_e_retorna_usuario_logado($this);

    try {
        parse_str(urldecode($filtros), $filtros);
        $abastecimentos =
            $this->abastecimentoModel->get_abastecimentos(
                $usuario_id,
                $filtros
            );

        $this->response([
            "erro" => false,
            "abastecimentos" => $abastecimentos
        ], 200);
    } catch (Exception $e) {

        $this->response([
            "erro" => true,
            "mensagem" => $e->getMessage()
        ], $e->getCode());
    }
}
```

Fonte: elaborada pelo autor

Na Figura 3.3 a função `valida_e_retorna_usuario_logado` é responsável por validar o `token` do usuário e verificar se ele realmente tem acesso ao sistema como citado anteriormente. A estrutura `try catch` contém a invocação da `Model abastecimentoModel` responsável por tratar das regras de negócio, validar dados das requisições e devolver as informações aos `Controller`. Caso algum erro ocorra no processo de execução, o código irá parar e entrar no fluxo do `catch` onde uma mensagem de erro é retornada ao usuário para informá-lo do ocor-

rido. Caso contrário, a `Controller` devolverá ao usuário uma resposta com código de status 200, além dos abastecimentos buscados.

## 3.3 IMPLEMENTAÇÃO DE SERVIÇOS NO FRONTEND

Com a API criada e retornando os dados corretamente, é possível iniciar o desenvolvimento do `frontend`. A Delta possui uma equipe especializada em `frontend` na sede da empresa em Porto Alegre e, esses desenvolvedores são responsáveis por desenhar e implementar o esquema da tela, deixando-a completamente estática para que os desenvolvedores de Lavras implementem as funcionalidades e dêem vida a elas.

A primeira atividade a ser realizada é criar um serviço. Serviços são responsáveis por realizar chamadas a API por meio da biblioteca Axios, para buscar ou tratar dados de uma entidade específica no banco de dados. Por exemplo, ao criar uma função para buscar os dados dos abastecimentos realizados, tal função será criada no serviço de Abastecimentos. Os serviços são funções que fazem requisições HTTP ao `backend` e retornam o resultado da API para os componentes do React. A Figura 3.4 exemplifica um serviço que realiza uma requisição do tipo GET à API.

O bloco `try catch` contém o código responsável pela requisição HTTP. Tal requisição é realizada por meio da biblioteca `axios`. A função recebe um objeto como parâmetro, que contém propriedades importantes para que a chamada à API seja realizada corretamente.

A primeira delas é a `method`, ela define qual o verbo HTTP será utilizado na requisição. Essa propriedade é muito importante, já que é possível codificar rotas com o mesmo nome que se diferenciam apenas pelo verbo HTTP. Definir o método errôneamente irá resultar em uma resposta diferente, podendo, inclusive, parar a execução da aplicação.

A segunda propriedade é a `url`. Por meio desta define-se o caminho para o recursos que se deseja acessar. Sendo assim, é fundamental para o resultado correto da chamada.

A terceira propriedade é a `header`. Esta é responsável por guardar os metadados da requisição. Por meio dela é possível enviarmos um cabeçalho `Authorization` que conterá informações cruciais para a identificação e autententicação dos usuários.

Figura 3.4 – Exemplo de serviço para buscar abastecimentos

```
export const get_abastecimento = async (filtros = 0) => {
  let response = {};

  const queryParams =
    apiHelper.urlencode(new URLSearchParams(filtros).toString());

  try {
    response = (await axios({
      method: "GET",
      url: `https://api.exemplo.com.br/api/abastecimento/${queryParams}`,
      header: apiHelper.setHeaderBearerToken()
    })).data;

  } catch (e) {
    response = e.response.data;
  }

  return response;
}
```

Fonte: elaborada pelo autor

## 3.4 RENDERIZAÇÃO DOS DADOS NOS COMPONENTES REACTJS

Com a resposta da camada de serviço, é possível passar para a próxima etapa, exibir os dados da API no navegador do usuário, utilizando a biblioteca ReactJs. Para isso, é necessário entender os três pilares do React: Componentes, Estados e Propriedades. Componentes permitem que mais código seja reaproveitado e que seja possível segmentar uma interface em várias partes. Para o React, um componente é qualquer função Javascript que recebe como parâmetro um argumento, que são as propriedades, e retorna um JSX, acrônimo para Javascript XML, padrão utilizado para facilitar a renderização de componentes na interface. Na Figura 3.5 podemos observar o exemplo mais básico de um componente React, que retorna apenas a palavra “Abastecimentos”, que será exibida no navegador do usuário.

Figura 3.5 – Exemplo de componente React

```
import React from "react";

const Abastecimento = () => {
    return (
        <h1>Abastecimento</h1>
    );
}
```

Fonte: elaborada pelo autor

O Estado é um conjunto de variáveis que pertencem ao escopo de um componente React. Essas variáveis são utilizadas para armazenar informações tanto inseridas pelo usuário por meio de formulários, quanto por processamentos da própria aplicação, como por exemplo guardar dados retornados da API. Ao guardar um valor em uma variável de estado, pode-se por meio do React observar as alterações desse valor, pois todo o componente irá ser renderizado novamente e emitir eventos para a aplicação. Uma das maneiras de observar essas alterações do estado de um componente é utilizando a função do React chamada `useEffect`. Essa função recebe uma outra função como primeiro parâmetro e permite especificar quais variáveis de estado deseja-se observar, por meio de um array, no segundo parâmetro. Dessa maneira, na função especificada no primeiro parâmetro do `useEffect`, pode-se definir o que fazer quando algum estado específico é alterado.

Supondo que se deseja buscar os abastecimentos realizados de determinado veículo, para isso o serviço `get_abastecimentos` é utilizado, exemplificado na Figura 3.4. Após recebido o retorno do serviço, deseja-se armazenar o resultado em uma variável de estado do componente chamada de abastecimentos. Para isso, invoca-se a função do React chamada `useState`, que retornará um array com duas posições, a primeira é o valor do estado e a segunda uma função para alterar esse valor. É uma boa prática nomear essa função do segundo parâmetro utilizando o prefixo set, no caso do abastecimento, `setAbastecimento`, de maneira que fique explícito que se trata de uma função que irá alterar o estado do componente.

Figura 3.6 – Exemplo de estado, useEffect e utilização de serviço

```
import React, { useState, useEffect } from "react";
import { get_abastecimento } from "./service/Abastecimentos";

const Abastecimentos = () => {
  // estado
  const [abastecimentos, setAbastecimentos] = useState([]);

  const buscarAbastecimentos = async () => {
    const resposta = await get_abastecimento();
    const { erro } = resposta;

    if (erro == false) {
      setAbastecimentos(resposta.abastecimentos);
    }
  }

  useEffect(() => {
    buscarAbastecimentos();
  }, []);

  return (
    <h1>Abastecimentos</h1>
  );
}
```

Fonte: elaborada pelo autor

Na Figura 3.6 é definido um estado para armazenar os abastecimentos. Por meio da função `buscarAbastecimentos` utiliza-se o serviço para buscar informações na API e armazenar de fato a sua resposta. Para que a função seja chamada somente uma vez, quando o componente for renderizado pela primeira vez, define-se uma função useEffect passando para ela no primeiro parâmetro uma função que executa `buscarAbastecimentos` e, no segundo, um array vazio dizendo que este useEffect não executará quando qualquer estado seja alterado. Como a resposta da API é um array de objetos do tipo Abastecimento, percorre-se tal array, exibindo todas as suas posições para o usuário. Na Figura 3.7 é definada a função `renderizarAbastecimentos`, responsável por iterar sobre o estado abastecimentos e exibir um elemento div para cada uma das posições do array.

Figura 3.7 – Exibindo os dados com a função renderizarAbastecimentos

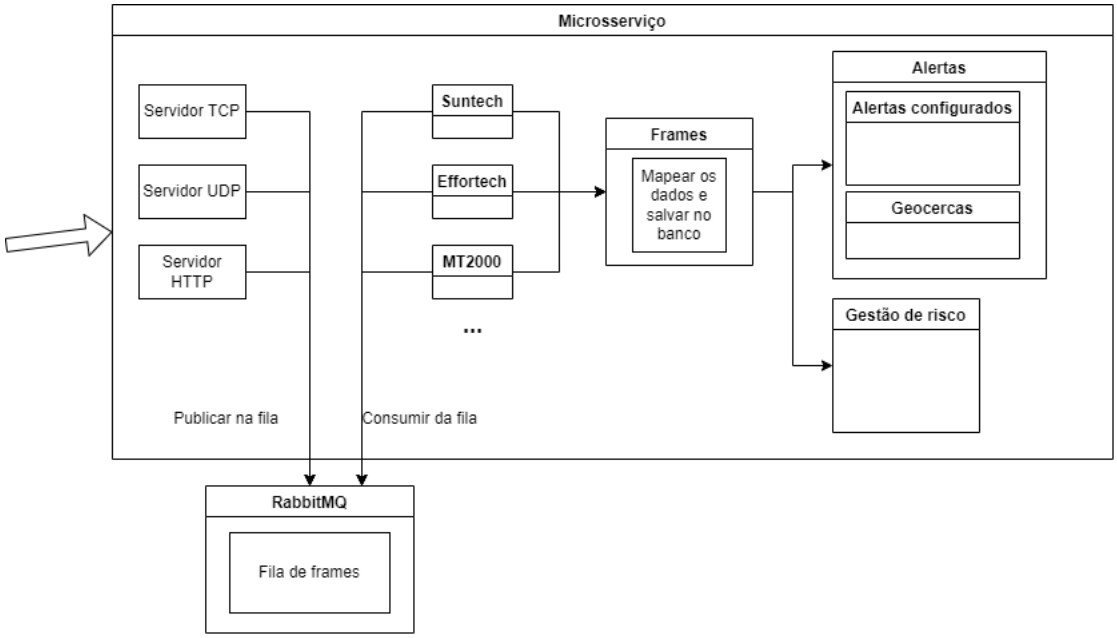
```
const renderizarAbastecimentos = () => {
if (abastecimentos) {
    return abastecimentos.map(abastecimento => {
        <div>
            <div>
                {abastecimento.qtd_litros}
            </div>
            <div>
                {abastecimento.preco_combustivel}
            </div>
            <div>
                {abastecimento.data_hora}
            </div>
        </div>
    })
}


return (
    {renderizarAbastecimentos}
);
```

Fonte: elaborada pelo autor

## 3.5 IMPLEMENTAÇÃO DO MICROSSERVIÇO

Como citado anteriormente, o DeltaFleet é um projeto de monitoramento veicular e, para tal, lê dados de rastreadores instalados nos veículos. O Microsserviço é o projeto em Nodejs responsável por receber, processar e guardar no banco de dados todas as informações que os rastreadores enviam, dessa maneira, é considerado o coração do sistema. Apesar do nome, o projeto possui poucas características que o enquadram em tal abordagem arquitetônica. Isso porque, tal projeto concentra diversas responsabilidades, como é possível observar na Figura 3.8, que ilustra de maneria geral o mesmo, divergindo do conceito de microsserviços de implementar pequenos serviços independentes que se comunicam.

Figura 3.8 – Arquitetura atual do microsserviço

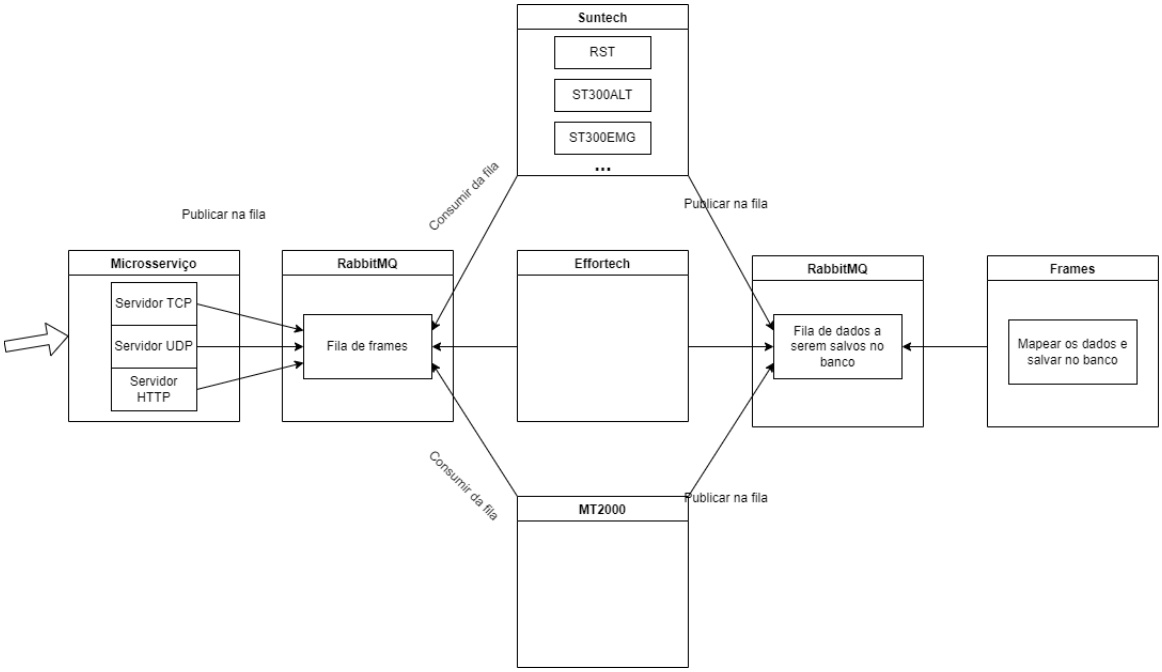



Fonte: elaborada pelo autor

A função principal do projeto é receber o pacote enviado pelo rastreador e por meio deste pacote identificar qual o modelo do rastreador para invocar o Parser adequado com a finalidade de tratar o pacote. Um Parser é uma funcionalidade implementada para tratar os dados de um rastreador específico, ou modelo específico caso uma marca de rastreador possua mais de um modelo. Citando como exemplo, temos a marca de rastreadores Suntech. Essa marca possui bastantes modelos de equipamentos, como por exemplo o RST, ST300ALT, ST300EMG, ST300STT, entre outros. Pode ocorrer de modelos diferentes possuírem pacotes codificados de maneiras diferentes, por isso é necessário mais de um Parser. Como é possível observar na Figura 3.8, a porta de entrada dos pacotes dos rastreadores são os servidores TCP, UDP e HTTP, que estão sempre ligados. O motivo desses três tipos de servidores serem necessários é pelo fato de equipamentos diferentes utilizarem diferentes protocolos para se conectarem ao servidor.

Para que o projeto realmente incorporasse mais o conceito de microsserviço, deveria ocorrer uma subdivisão nessa camada em diversos outros microsserviços, cada um responsável por tratar os dados de uma marca de rastreador diferente, todos orquestrados pelo serviço de filas RabbitMQ, como pode ser visto na Figura 3.9. Após processados por seus respectivos parsers, o pacote seria enviado para uma nova fila para ser salvo no banco de dados, de modo que a API possa consumir tais pacotes a fim de exibir as posições dos veículos no `frontend`.

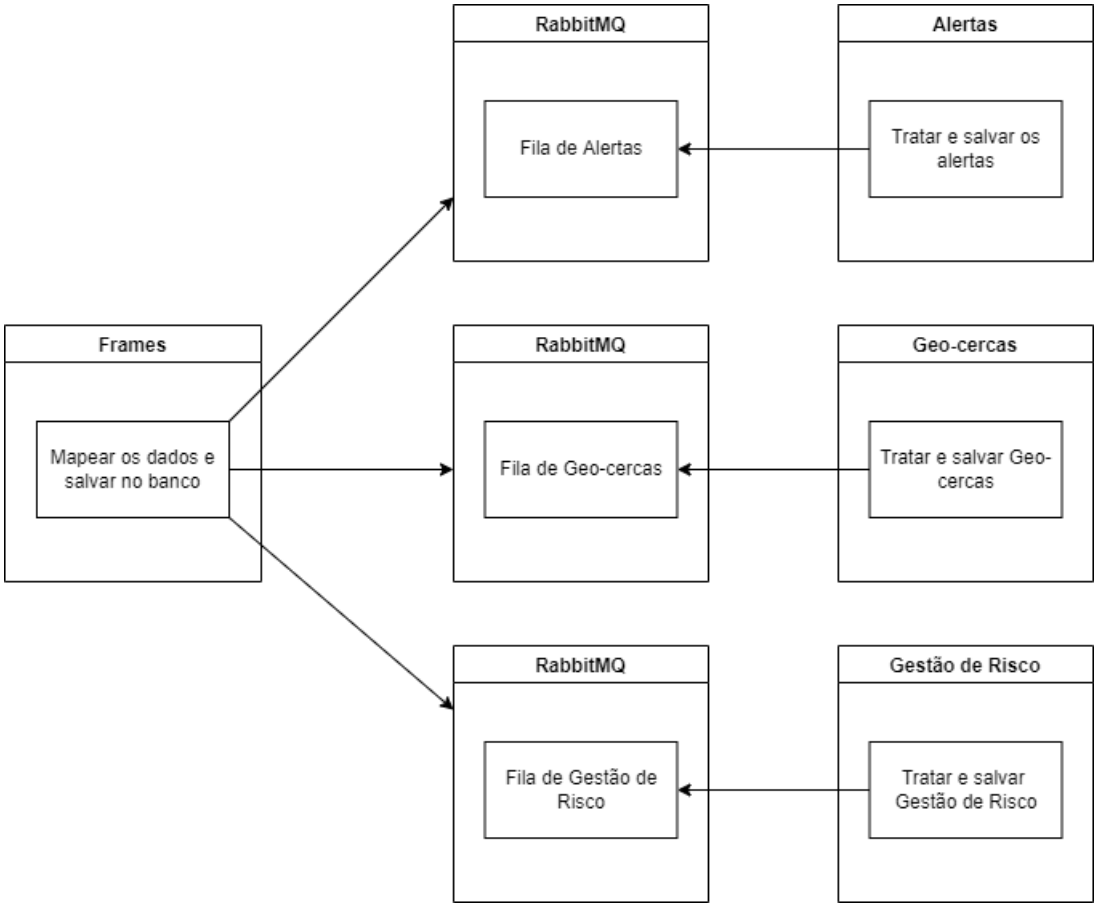
Figura 3.9 – Esquema de arquitetura dividido para cada marca de rastreador




Fonte: elaborada pelo autor

Após mapeados e salvos no banco esses dados seriam enviados para as filas de processamentos de alertas, geo-cercas e gestão de risco, que são funcionalidades do sistema. A Figura 3.10 começa onde a Figura 3.9 termina, ao mapear e salvar os pacotes dos rastreadores no banco de dados.

Figura 3.10 – Fluxo para tratar os alertas, geo-ceras e gestão de risco




Fonte: elaborada pelo autor

Esses microsserviços seriam responsáveis por validar as regras de negócio de suas respectivas funcionalidades, salvando registros no banco de dados, caso seja identificada a ocorrência de algum dos itens da Figura 3.10.

Esse modelo de arquitetura seria mais vantajoso de se utilizar em relação ao modelo atual por alguns motivos e o primeiro deles é a disponibilidade. De acordo com a ISO/IEC 25010 (ISO/IEC, 2011), a disponibilidade é o grau em que um sistema, produto ou componente está operacional e acessível quando necessário para uso. Imaginemos uma situação, no novo modelo sugerido para o microsserviço, onde um pacote foi direcionado para o parser do rastreador MT2000. Porém um erro não previsto, e consequentemente não tratado, quebra a aplicação. Por se tratar de uma aplicação com características monolíticas todo o software será impactado pelo erro, o que poderá acarretar na perda de dados de posições e alertas de todos os modelos de rastreadores, não só do MT2000. Caso o mesmo erro ocorra na nova abordagem sugerida o fluxo seria interrompido somente nos rastreadores MT2000, e o impacto não se propagaria

para os demais serviços. Dessa maneira a disponibilidade do sistema aumentaria já que todos os outros processos continuariam ativos.

O segundo motivo é a escalabilidade do sistema, caso o número de equipamentos de uma determinada marca cresça muito bastaria replicar o microsserviço responsável pela marca para que houvesse uma divisão de cargas entre os dois ou mais, melhorando o desempenho do sistema. Além disso, esse modelo de arquitetura melhoraria também a manutenibilidade, que de acordo com a ISO/IEC 25010 (ISO/IEC, 2011) é o grau de eficácia e eficiência com o qual um produto ou sistema pode ser modificado para melhorá-lo, corrigi-lo, ou adaptá-lo a mudanças no ambiente e nos requisitos. Essa melhora seria sustentada por um sistema mais modular dividido em serviços separados, o que também aumentaria o grau de reusabilidade.

# 4 CONSIDERAÇÕES FINAIS

O objetivo desta seção é, entre outros assuntos, abordar a experiência do estagiário. Qualquer trabalho que envolva atividades em equipe serve de grande experiência e nos força a desenvolvermos e aprimorarmos habilidades como a comunicação, item essencial para o bom entendimento e desenvolvimento das atividades diárias. Além disso, a inteligência emocional é outro aspecto importantíssimo de se desenvolver, pois geralmente o escopo do trabalho é desenvolver uma solução para um problema que precisa de ser resolvido. Dessa maneira, adversidades surgirão e que inicialmente talvez pareçam impossíveis ou muito difíceis de se solucionar. Nesses momentos se faz necessário tentar olhar para o problema por um prisma mais amplo, a fim de mapear todas as soluções possíveis para tais adversidades.

A curva de aprendizado no desenvolvimento de software é um pouco amarga para quem está em seu começo. Ao entrar em uma empresa, logo de início, o desenvolvedor precisa assimilar a nova base de código com a qual irá trabalhar, lidar com código legado e aprender uma linguagem nova, caso ainda não esteja familiarizado com a utilizada no projeto em que está alocado. No caso do estagiário, aprendi o PHP e o framework Codeigniter ao mesmo tempo em que aprendia sobre o projeto no qual estou alocado, e isso no começo foi do estágio foi um pouco sufocante. Apesar disso, meu tempo de empresa contribuiu ainda mais para o desenvolvimento da minha habilidade de me comunicar, pensar de maneira computacional e desenvolver soluções de software para problemas.

Em relação às disciplinas realizadas no curso de Sistemas de Informação, as disciplinas de Introdução aos Algoritmos, Estruturas de Dados, Práticas de Programação Orientada a Objetos, Sistemas Gerenciadores de Bancos de Dados e Engenharia de Software contribuíram bastante para a minha formação e, consequentemente, o uso do conhecimento na prática. Por meio delas, o estagiário aprendeu os principais conceitos do paradigma de programação orientada a objetos, tão utilizado por desenvolvedores. Além disso, aprendeu sobre a modelagem e manipulação de bancos de dados, padrões de projetos e arquitetura de software em geral. Todos os conhecimentos adquiridos acima fazem parte do dia a dia como desenvolvedor, e são cruciais para a construção de software de qualidade e a manutenção do mesmo. Por cumprir o papel de programador, os conhecimentos das disciplinas de Administração e de disciplinas mais teóricas como Processos de Software, não foram utilizados da mesma forma que os conhecimentos citados anteriormente, até então. Porém, aguardo o momento em que poderei aplicar tais conhecimentos. Com mais tempo de empresa, o maior conhecimento de seus processos e o acúmulo

de experiência profissional, o estagiário espera contribuir mais para a criação e otimização dos processos implementados até então.

# REFERÊNCIAS


AXIOS. **Axios**. [S.l.], 2020. Disponível em: <https://axios-http.com/ptbr/docs/intro>.

BASTOS, D. F. Framework codeigniter: do machado à motosserra. 2011. Disponível em: <https://www.oficinadanet.com.br/artigo/php/framework-codeigniter-do-machado-a-motosserra>.

DIAS, A. Conceitos básicos de controle de versão de software — centralizado e distribuído. 2016. Disponível em: <https://blog.pronus.io/posts/controle-de-versao/conceitos-basicos-de-controle-de-versao-de-software-centralizado-e-distribuido/>.

DOCS, M. W. **CSS**. [S.l.], 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/css>.

DOCS, M. W. **HTML: Linguagem de Marcação de Hipertexo**. [S.l.], 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/html>.

DOCS, M. W. **Javascript**. [S.l.], 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/javascript>.

DOSSOT, D. **RabbitMQ Essentials**. [s.n.], 2014. ISBN 978-1-78398-320-9. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=FoBvAwAAQBAJ&oi=fnd&pg=PT8&dq=what+is+rabbitmq&ots=86PbGAWVh3&sig=gMwao3IFf5YcZ-BeN4DrC7jFuuY#v=onepage&q&f=false>.

EDUCATION, I. C. Application programming interface (api). 2020. Disponível em: <https://www.ibm.com/cloud/learn/api>.

ISO/IEC. **Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models**. [S.l.], 2011. Disponível em: <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?start=3>.

KHOLODKOV, V. **Nginx Essentials**. [s.n.], 2015. Disponível em: <https://www.packtpub.com/product/nginx-essentials/9781785289538>.

L., A. O que é apache? uma visão aprofundada do servidor apache. 2021. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-apache>.

MANZANO, J. A. N. G. **MySQL 5.5 Interativo: Guia Essencial de Orientação e Desenvolvimento**. [s.n.], 2011. Disponível em: <https://integrada.minhabiblioteca.com.br/books/9788536519449/>.

OLIVEIRA CLáUDIO LUíS, V. e. H. A. P. Z. **Node.js: programe de forma rápida e prática**. [S.l.: s.n.], 2021.

REACT. **React: Uma biblioteca Javascript para criar interfaces de usuário**. [S.l.], 2022. Disponível em: <https://pt-br.reactjs.org>.

REDBOOKS, I. Microservices from theory to practice. 2015. Disponível em: <http://www.redbooks.ibm.com/redbooks/pdfs/sg248275.pdf>.

RIBEIRO, R. T. Mvc: a essência e a web. 2013. Disponível em: <http://rubsphp.blogspot.com.br/2013/02/mvc-essencia-e-web.html>.

SARAIVA MAURíCIO DE O.; BARRETO, J. d. S. **Desenvolvimento de sistemas com PHP**. Porto Alegre: [s.n.], 2018. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595023222/>.

VENTURA, P. Interfaces de programação. 2015. Disponível em: <https://www.ateomomento.com.br/o-que-e-api/>.