BOANERGES POTYGUARA SAES JÚNIOR

DESENVOLVIMENTO DE SOLUÇÕES WEB NA ÁREA DE MOBILIDADE CORPORATIVA

LAVRAS – MG

2022

BOANERGES POTYGUARA SAES JÚNIOR

DESENVOLVIMENTO DE SOLUÇÕES WEB NA ÁREA DE MOBILIDADE CORPORATIVA

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel.

Prof. DSc. Maurício Ronny de Almeida Souza

Orientador

LAVRAS – MG

2022

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Junior, Boanerges Potyguara Saes

Desenvolvimento de soluções web na área de mobilidade corporativa / Boanerges Potyguara Saes Junior. 1^a ed. rev., atual. e ampl. – Lavras : UFLA, 2022.

49 p. : il.

Relatório de estágio(graduação)–Universidade Federal de Lavras, 2022.

Orientador: Prof. DSc. Maurício Ronny de Almeida Souza.

Bibliografia.

1. TCC. 2. Monografia. 3. Dissertação. 4. Tese. 5. Trabalho Científico – Normas. I. Universidade Federal de Lavras. II. Título.

CDD-808.066

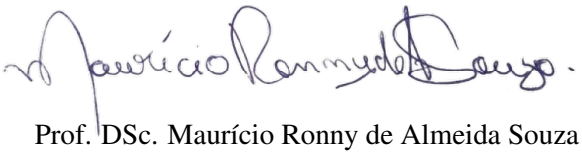
BOANERGES POTYGUARA SAES JÚNIOR

DESENVOLVIMENTO DE SOLUÇÕES WEB NA ÁREA DE MOBILIDADE CORPORATIVA

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel.

APROVADA em 16 de setembro de 2022.

Prof. DSc. Raphael Winckler de Bettio	UFLA
Prof. DSc. Rafael Serapilha Durelli	UFLA



Prof. DSc. Maurício Ronny de Almeida Souza
Orientador

LAVRAS – MG
2022

RESUMO

A VOLL é uma empresa de mobilidade corporativa, fundada em 2017 por quatro pessoas do setor de viagens. Tem como objetivo melhorar a experiência do viajante durante uma viagem de negócios e fornecer a melhor infra-estrutura de homologação de recibos, *tickets* e *vouchers*. Este trabalho tem como objetivo descrever as atividades de estágio realizadas durante o período de agosto de 2021 e maio de 2022 como um desenvolvedor *Web FullStack*. As atividades englobam duas áreas de atuação, *Back-End* e *Front-End*. As principais tecnologias utilizadas foram NodeJS, Flutter, Kafka, GraphQL, MongoDB, entre outras. Como resultado foram desenvolvidas atividades de melhoria de *layout*, novas formas de consulta de *vouchers*, análise de desempenho e atividades relacionadas.

Palavras-chave: Desenvolvimento Web. Back-End. Front-End. Estágio.

ABSTRACT

VOLL is a corporate mobility company, founded in 2017 by four people from the travel industry. It aims to improve the traveler's experience during a business trip and provide the best infrastructure for the approval of receipts, *tickets* and *vouchers*. This work aims to describe the internship activities carried out during the period of August 2021 and May 2022 as a *Web FullStack* developer. The activities encompass two areas of activity, *Back-End* and *Front-End*. The main technologies used were NodeJS, Flutter, Kafka, GraphQL, MongoDB, among others. As a result, activities were developed to improve *layout*, new ways of consulting *vouchers*, performance analysis and related activities.

Keywords: Web development. Back-End. Front End. Internship.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A mobilidade corporativa tem haver com o deslocamento dos colaboradores até o local de trabalho e o que a empresa pode fazer para melhorar esse trajeto. De acordo com o WRI[1] (*World Resources Institute*) 50% dos deslocamentos urbanos são realizados para a pessoa ir ao trabalho e a média é de 2 horas entre o ir e vir. Dessa forma, diversas empresas têm buscado um plano de mobilidade corporativa para melhorar esse deslocamento. Elas estão se modernizando e usando a tecnologia que está ao alcance das mãos a seu favor, com isso, elas aumentam a produtividade e diminuem gastos.

VOLL é uma *startup* que fornece serviços que auxiliam o processo de viagens corporativas. A empresa disponibiliza um aplicativo com diversos serviços para melhorar a experiência do usuário durante seu trajeto e centraliza todos os tipos de custos em um só lugar. Tratando-se de viagens corporativas, o modelo de negócio adotado é o B2B[2] (*Business to Business*), ou seja, os serviços estão disponíveis somente para o mundo corporativo, onde a contratação do serviço é feito apenas para CNPJ. Os serviços disponíveis variam desde realizar uma corrida de táxi até realizar uma reserva em um hotel em qualquer estado do Brasil. Dito isso, a melhor forma de disponibilizá-lo é através do seu principal produto, um aplicativo para dispositivos móveis de mesmo nome VOLL.

Portanto este relatório tem por finalidade descrever todas as atividades desenvolvidas pelo aluno durante o período de estágio na *startup* VOLL. O estágio é resumido em desenvolvimento web com foco no produto carro chefe da empresa, tendo como conceito o *mobile first*[3], o aplicativo como sendo prioridade, e tem como objetivo descrever a atuação do aluno utilizando os conhecimentos adquiridos durante o curso de Ciência da Computação e as tecnologias utilizadas na empresa. Também demonstra o processo de mudanças de times que ocorreram durante o período de estágio, abordando as mudanças de conceitos e funções realizadas.

O produto desenvolvido envolve as mais novas tecnologias de mercado, e está em constante atualização para atender aos clientes, portanto o objetivo do estágio foi o de desenvolver novas funcio-

[1] https://www.wri.org/initiatives/mobility-and-accessibility-program
[2] https://www.serasa.com.br/blog/o-que-e-o-modelo-de-negocio-e-vendas-b2b-e-como-ele-funciona/
[3] https://www.hostgator.com.br/blog/mobile-first-o-que-e/

nalidades, manter e dar suporte ao sistema legado e aplicar o conhecimento adquirido durante o curso de Ciências da Computação. O estágio foi realizado no período de agosto de 2021 a maio de 2022.

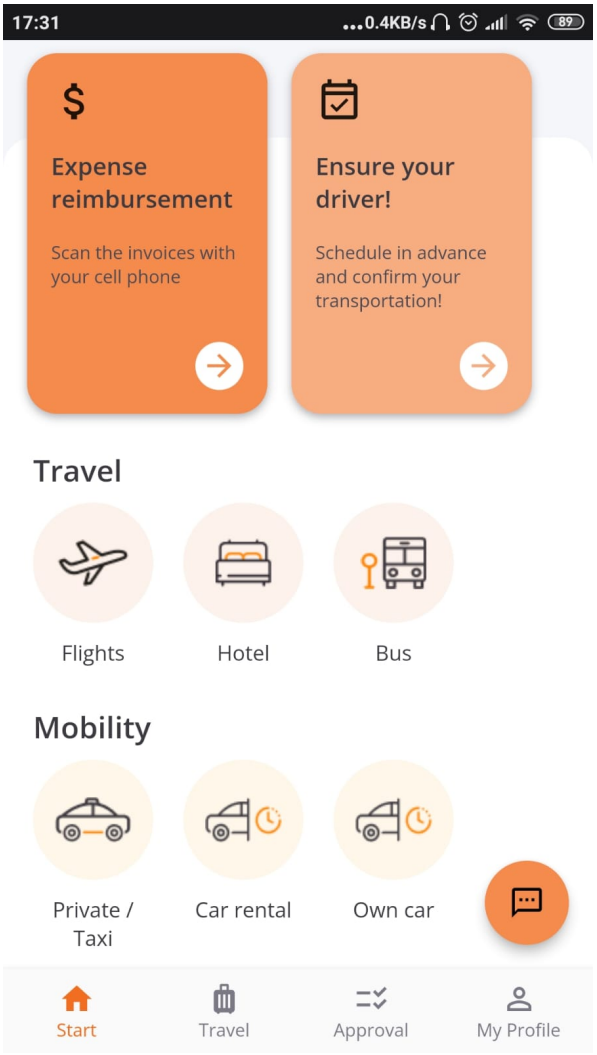
1.1 Sobre a VOLL

Fundada em 2017 por quatro pessoas que anteriormente trabalhavam em uma empresa de mobilidade chamada BTM Transportes, que tempos depois foi incorporada a VOLL, já possui mais de mil empresas utilizando seu serviço, espalhadas em 120 países e contam com mais de 300 colaboradores. A VOLL é uma empresa focada na gestão de viagens e mobilidade corporativa, especialmente projetada para simplificar processos, otimizar custos e facilitar a jornada de viagens dos colaboradores, tudo isso em apenas um só aplicativo que fornece todas as soluções e que consistem em:

- Gestão de despesas unificadas: eliminando as notinhas de papel e pedidos de reembolso por e-mail, centralizando a gestão de despesas para ter maior controle e visibilidade de toda a jornada de viagem.
- Gestão de viagens com eficiência e economia: A VOLL alia tecnologia, ferramentas e atendimento humanizado para simplificar processos, otimizar custos e facilitar as viagens corporativas.
- Atendimento humanizado e suporte 24h: A VOLL acredita no privilégio do calor humano e, por isso, reforça a presença humana em diferentes canais, para o colaborador ter sempre com quem contar.

Como mostrado na Figura 1.1, os serviços disponíveis são descritos como, reservas em hotéis, corridas de táxi e outras fornecedoras, passagens de aviões e ônibus com checagem de bagagem e assento, despesas com comidas e carros próprios, reembolsos de gastos extraordinários, entre outros

Figura 1.1 – Aplicativo VOLL



Fonte: VOLL

A empresa possui uma estrutura com diversas áreas de apoio ao cliente, como *Banking* que cuida das despesas que o cliente gera no aplicativo, Sucesso do Cliente que está lado a lado ao cliente para auxiliar nas dúvidas, receber *feedbacks* e relatar pedidos, a área de Tecnologia onde são feitos as inovações e manutenções do aplicativo juntamente com a segurança de dados, entre outros. Também conta com estrutura física, tendo como sede localizada na cidade de Belo-Horizonte - MG e parte da comunidade Cubo Itaú em São Paulo - SP

1.2 Organização do Trabalho

Além deste capítulo inicial, este documento está organizado nos seguintes capítulos. O Capítulo 2, **Conceitos e Tecnologias**, descreve o ferramental utilizado durante o desenvolvimento do aplicativo, e os principais conceitos necessários para a compreensão das atividades desenvolvidas. O Capítulo 3, **Atividades desenvolvidas**, descreve o planejamento e execução das atividades de desenvolvimento e manutenção realizadas durante o período de estágio. Finalmente, o Capítulo 4, **Conclusão**, descreve as considerações do autor sobre a experiência adquirida durante o estágio desenvolvido.

2 CONCEITOS E TECNOLOGIAS

Este capítulo apresenta as tecnologias e conceitos utilizados para o desenvolvimento das atividades durante o período de estágio. As Seções 2.1, 2.2 e 2.3 tratam de conceitos relacionados ao desenvolvimento Web, Front-End e Back-End, respectivamente. A Seção 2.4 trata de conceitos relacionado ao Controle de Versão, e a ferramenta utilizada para este fim na organização: o Git.

2.1 Desenvolvimento Web

Quanto mais serviços migram para a internet, mais o mercado de tecnologia se expande. O desenvolvimento de aplicações e soluções estão ficando cada vez mais populares quando o assunto é complexidade e sofisticação. Para solucionar esses problemas, surge o assunto Desenvolvimento Web. De acordo com Fain et al. (2014) o Desenvolvimento Web é responsável por codificar os sites, páginas, portais e aplicativos para a web. Com isso, é ele o responsável por toda a estrutura de um site, tanto da parte que o usuário consegue ver e interagir, quanto a parte que o usuário não vê, mas que faz toda a diferença na experiência ao usar o site.

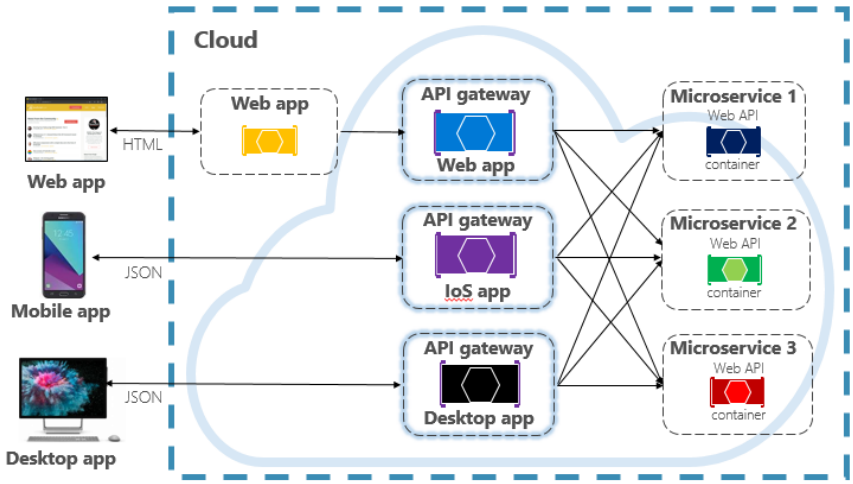
No Desenvolvimento Web existem diferentes contextos, sendo elas as principais Front-End que é toda parte que envolve a área visível de um site e toda a parte com a qual o usuário pode interagir, em outras palavras, sua interface externa e Back-end que é a parte que envolve todo sistema interno de um site, ou seja, tudo aquilo que diz respeito ao seu funcionamento e armazenamento de dados e informações e que serão aprofundadas nos capítulos 2.2 e 2.3 respectivamente (SHOFNER, 2017). Visto que estes são dois conceitos importantes distintos, pode-se esperar que suas aplicações também sejam distintas, portanto segundo Coulouris et al. (2013) as aplicações web são divididas em 3 partes: Interface de usuário, define a parte visível do sistema para o usuário que, ele se comunica para realizar suas tarefas; Servidor, ponte de comunicação entre a interface e o banco de dados, realizando as manipulações de dados necessárias antes de transmitir a mensagem entre um e outro; e Banco de dados, que armazena os dados produzidos dentro do sistema, de maneira a garantir a persistência das informações.

2.2 Front-End

Para acessar um site é preciso utilizar um navegador. A *interface* que o usuário para utiliza acessar recursos de um servidor web é chamada de Front-End (SMITH, 2012). É responsável pela comunicação com o usuário ao identificar as interações, seja em forma física ou exibir um conteúdo, por exemplo. Segundo Aquino e Gandee (2016) o Front-End é desenvolvido por um conjunto de tecnologias e cada uma tem seu papel na hora de exibir um conteúdo, formado básicamente por: HTML [1] (hyper text markup language), que é responsável por estruturar a página; o CSS [2], que representa a estilização da aplicação, como tamanhos, cores e alinhamentos; e o Javascript, responsável pela manipulação dos dados que são requisitados pelo cliente.

O Front-End exibe dados que dependem de um servidor, que geralmente está conectado a um banco de dados. Dependendo do tipo de interação do usuário, a lógica criada no Front-End faz uma chamada via requisição para este servidor e um serviço é executado. Este serviço pode modificar o dado no banco de dados ou apenas exibi-lo. A Figura 2.1 demonstra os diversos tipos de aplicações que o Front-End pode desenvolver utilizando transferência de dados com um servidor.

Figura 2.1 – Comunicação com o servidor em nuvem



Fonte: (PHANG, 2020)

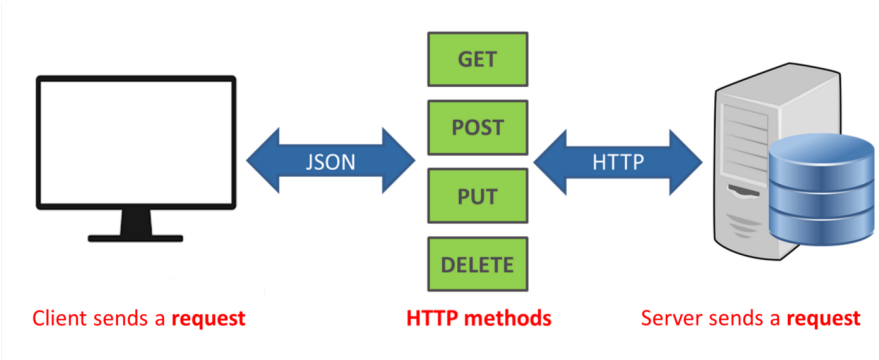
[1] https://developer.mozilla.org/pt-BR/docs/Web/HTML
[2] https://developer.mozilla.org/pt-BR/docs/Web/CSS

Esta comunicação é comumente feita por requisições HTTP [3] (*Hypertext Transfer Protocol*). HTTP é uma abreviação de *HyperText Markup Language*, é um padrão onde clientes requisitam dados de servidores através de requisições (PHANG, 2020) e que define um conjunto de requisições responsáveis por indicar a ação a ser executada para um dado recurso, sendo elas exemplificadas na Figura 2.2, que mostra quais palavras são utilizadas por padrão:

- *GET*, método que solicita a representação de um recurso específico. As requisições devem retornar apenas dados.
- *PUT*, método que substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.
- *POST*, método utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor.
- *DELETE*, método que remove um recurso específico.

e como tipo de transferência de dados a utilização do JSON[4] (JavaScript Object Notation), que é basicamente um formato de troca de informações ou dados entre sistemas, através do protoloco HTTP

Figura 2.2 – Métodos HTTP



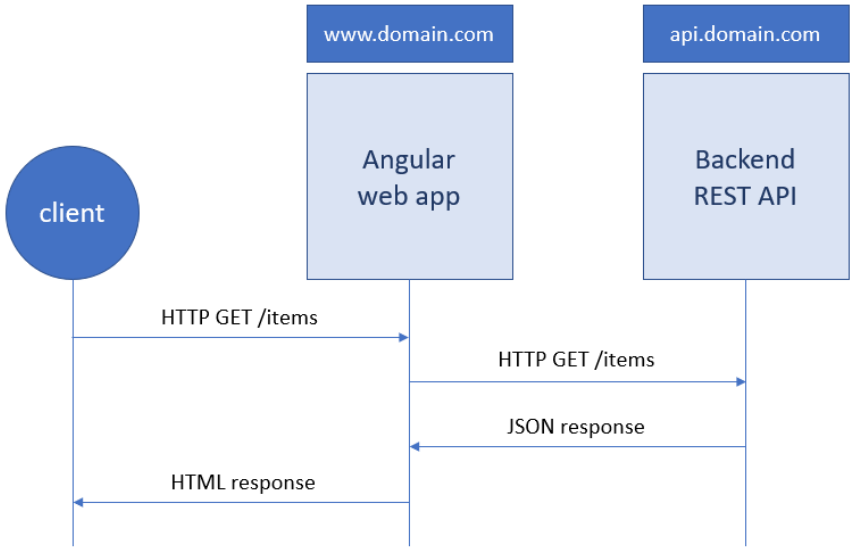
Fonte: https://phpenthusiast.com/blog/what-is-rest-api

O diagrama de sequência apresentado na Figura 2.3 exemplifica como seria feita uma requisição completa. O cliente inicia uma chamada via HTTP solicitando a listagem de *items*. O Front-End

[3] https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview
[4] https://www.json.org/json-en.html

repassa esta requisição para o servidor. Em resposta recebe um JSON (*JavaScript Object Notation*), um estilo de modelagem de dados. E finalizando o Front-End exibe um formato HTML para o cliente.

Figura 2.3 – Diagrama de sequência de uma requisição



Fonte: https://sites.google.com/site/cst315teamn/home/http-request-sequence-diagram

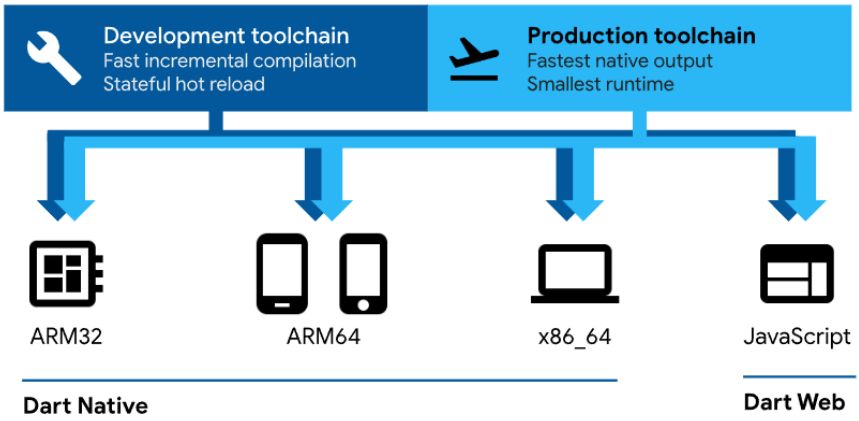
2.2.1 Dart

Dart [5] é uma linguagem de programação fortemente tipada onde os objetos e variáveis tem um tipo bem definido e que precisa ser informado no momento de sua declaração, como por exemplo um número tem que ser declarado com tipo Int ou Number, de acordo com a sintaxe da linguagem. Criada em 2011 pela Google, sua missão era substituir a linguagem JavaScript. Hoje, ela é uma linguagem multi-paradigma que apresenta fortes estruturas ligadas às linguagens orientadas a objetos. Projetada principalmente para se adequar ao desenvolvimento ao cliente, prioriza estados de atualização rápida e produção de alta qualidade e fornece um conjunto de bibliotecas nativas oferecendo o essencial para diversas tarefas na hora de programar (BITENCOURT, 2022) .

[5] https://dart.dev

As plataformas do Dart permitem que o código seja executado de diferentes maneiras, sendo elas nativas utilizando compiladores JIT [6] (*just-in-time*) que compilam em tempo de execução, e plataformas web que incluem um compilador de tempo de desenvolvimento (dartdevc) e um compilador de tempo de produção (dart2js) e ambos traduzem o código para JavaScript (Lars Bak e Kasper Lund, 2011). A figura 2.4 mostra como é feito o processo de compilação do código nativo para JavaScript.

Figura 2.4 – Compilador Dart



Fonte: https://dart.dev

2.2.2 Flutter

Criado pela Google, Flutter [7] é um *Framework* de código aberto para desenvolvimento de aplicações multi-plataforma. Focado em rapidez, produtividade e flexibilidade é utilizado em diversas empresas, como: Google, Alibaba e Nubank (NAPOLI, 2019). A linguagem base para utilizar Flutter é o Dart e sua estrutura é baseada em uma árvore de componentes (chamados de *Widgets*). De acordo com Napoli (2019), o flutter monta e renderiza os componentes neste formato, mas de maneira oculta utiliza *RenderObject*, que computa e implementam o *layout* básico e o protocolo de coloração. A Figura 2.5 representa como é feito uma classe em Flutter e como funciona a estrutura de árvore, mostrando como o *Widget Card* encapsula o *Widget Column*.

[6] https://www.ibm.com/docs/en/sdk-java-technology/7?topic=rja-just-in-time-jit-compiler
[7] https://flutter.dev/

Figura 2.5 – Exemplo de código com Flutter

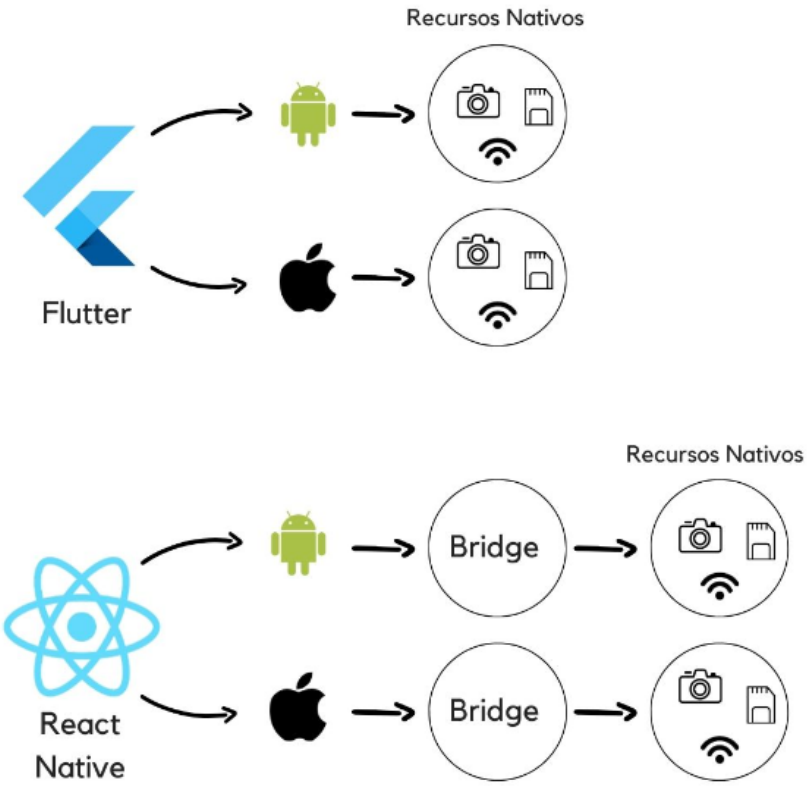
```dart
import 'package:flutter/material.dart';

class CustomCard extends StatelessWidget {  CustomCard({@required this.index});
    final index;

    @override
    Widget build(BuildContext context) {
        return Card(
            child: Column(
                children: <Widget>[Text('Card $index')],
            )
        );
    }
}
```

Fonte: https://flutter.dev/learn

O código Flutter compila para código de máquina ARM [8] ou Intel [9] e JavaScript para melhor performance em qualquer plataforma. Ao compilar para a linguagem base do dispositivo, suas aplicações se evidenciam realmente nativas, portanto conseguem acessar recursos do dispositivo sem interferências externas, como pacotes e softwares terceiros, tendo uma camada a menos e aumentando o desempenho da aplicação. A Figura 2.6 faz a comparação entre Flutter e React[10] de como o processo de compilação funciona.

[8] https://developer.arm.com/documentation
[9] https://www.nayuki.io/page/a-fundamental-introduction-to-x86-assembly-programming
[10] https://pt-br.reactjs.org/

Figura 2.6 – Comparação entre Flutter e React



Fonte: https://www.treinaweb.com.br/blog/o-que-e-flutter

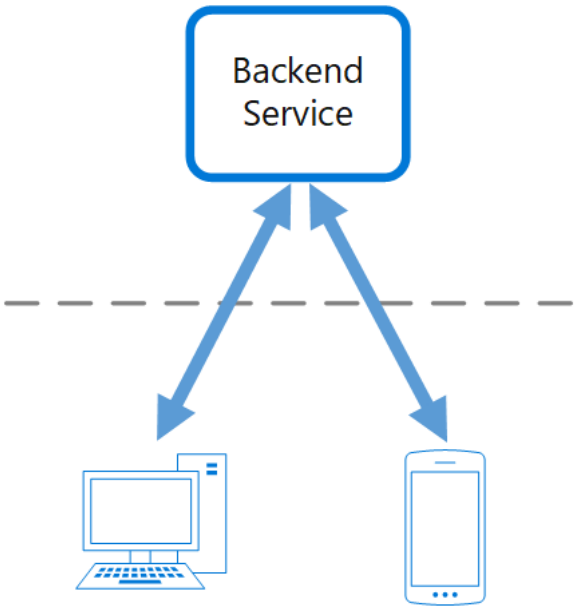
2.3 Back-End

De acordo com Dresher, Zuker e Friedman (2018), Back-End corresponde a uma ou mais aplicações rodando em máquinas que são comumente chamadas de servidores. O propósito dessas aplicações é disponibilizar requisições para outras aplicações e, se necessário, executar passos para validar essas requisições. Back-end é a parte responsável por gerenciar as regras de negócios de uma aplicação, ou seja, toda a estrutura que dá apoio às ações do usuário. Por exemplo, ao acessar um site o Back-End é responsável por toda comunicação em que é feita a troca de informações entre o navegador e o banco de dados (DRESHER; ZUKER; FRIEDMAN, 2018).

A responsabilidade de gerenciar o Back-End de uma aplicação é do desenvolvedor Back-End, e a principal diferença em relação ao desenvolvimento Front-End é que a aplicação é desenvolvida

independente de um usuário. Se a lógica de programação e as regras de negócio são atendidas, espera-se que a aplicação funcione em qualquer interface que conecte corretamente, ou seja, uma aplicação Back-End pode fornecer dados para uma ou mais aplicações Front-End, nas mais variadas plataformas (Figura 2.7).

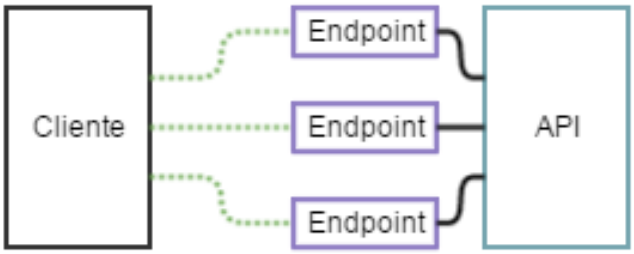
Figura 2.7 – Back-End



Fonte: https://docs.microsoft.com/pt-pt/azure/architecture/patterns/backends-for-frontends

Assim como no Front-End, a comunicação do Back-End é feita com requisições HTTP. As requisições são expostas em forma de *endpoints*, que representam os terminais de conexão entre uma API (Interface de programação de aplicações) e o cliente. As APIs são interfaces que funcionam como pontes, transportando dados entre um cliente e um servidor. Sem que esse processo seja sequer percebido pelo usuário, elas estão presentes por trás do funcionamento de diversos programas e aplicativos (WEIR; NEMEC, 2019). Como representado na Figura 2.8 uma aplicação pode ter mais de um *endpoint*, assim como um cliente pode se conectar a vários deles.

Figura 2.8 – Conexão Cliente-API com EndPoint



Fonte: https://www.cloudflare.com/pt-br/learning/security/api/what-is-api-endpoint/

As susbseções seguintes descrevem as principais tecnologias utilizadas no desenvolvimento Back-End durante o estágio realizado: JavaScript, Node.js, GraphQL, Apache Kafka e MongoDB.

2.3.1 JavaScript

JavaScript é uma linguagem desenvolvida para rodar no lado do cliente, isto é, a interpretação e o funcionamento da linguagem dependem de funcionalidades hospedadas no navegador do usuário (SILVA, 2010). É uma linguagem de alto nível, dinâmica, interpretada, não tipada e sintaticamente derivada do Java [11] (FLANAGAN, 2013). JavaScript é uma linguagem interpretada, isto é, o código é executado de cima para baixo e o resultado da execução do código é imediatamente retornado. Logo, não é preciso transformar o código em algo diferente antes do navegador executá-lo, como no caso de linguagens compiladas.

Além de ser possível executar no lado do cliente, também é possível executar JavaScript no lado do servidor. Códigos do lado do cliente são executados no computador do usuário, assim, quando uma página web é visualizada, o código do lado do cliente é baixado, executado e exibido pelo navegador. Códigos do lado do servidor, por outro lado, são executados no servidor e o resultado da execução é baixado e exibido no navegador.

Segundo Fain et al. (2014), na programação web, o JavaScript faz parte da tríade de tecnologias utilizadas que todo desenvolvedor deve conhecer: Javascript, CSS e HTML, sendo que, toda vez

[11] https://www.java.com/pt-BR/

que uma página web faz mais do que simplesmente exibir um conteúdo estático, como atualizações na página, mapas interativos, gráficos 2D/3D é devido a ação da linguagem. Dados da empresa *Hostinger* (Hostinger,) diz que Javascript é a quarta linguagem mais popular do mundo, ficando atrás apenas das linguagens Python, C# e C. Isso é consequência de 97,7% dos desenvolvedores utilizarem JavaScript para o desenvolvimento web com interface visível para os usuários.

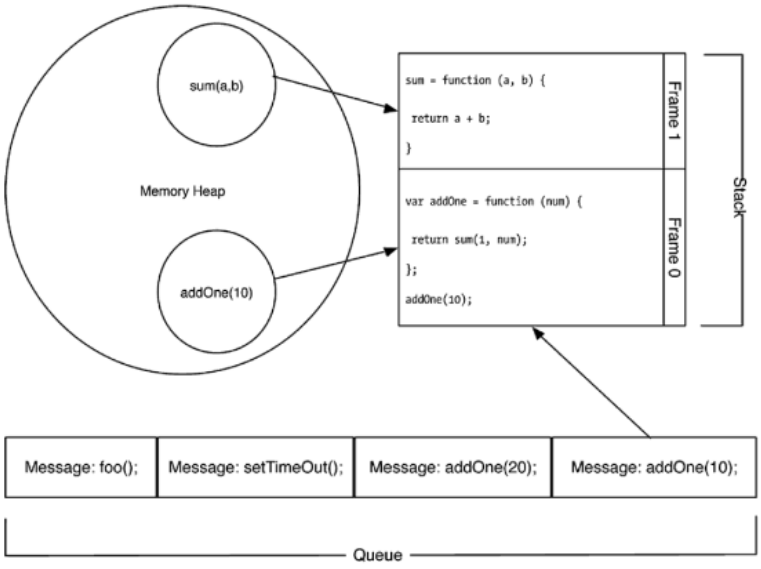
2.3.2 NodeJS

Em 2009, Ryan Dahl criou o NodeJS, um *framework* usado primariamente para criar servidores altamente escaláveis para aplicações web escrito em C++[12] e JavaScript (IHRIG, 2013). O Node é um ambiente de execução JavaScript *server-side*, isso significa que com Node.js é possível criar aplicações independentes para rodar em uma máquina, que não dependam de um navegador para executar como as aplicações *client-side*. Suas principais características são a alta capacidade de escalar aplicações, arquitetura, flexibilidade e o baixo custo que o torna uma boa escolha para implementação de micro serviços (IHRIG, 2013).

Por ser uma linguagem *Single Thread*, que só pode tratar uma requisição por vez e o processamento não pode ser demorado nem bloqueado, o Node.js funciona de maneira assíncrona de modo a evitar esses problemas. Seu funcionamento consiste em uma técnica de *Event Loop* que é um laço de repetição semi-infinito, sem bloqueio e assíncrono. De acordo com Daggett (2013), a estrutura do *Event Loop* consiste em: (i) Uma memória de armazenamento; (ii) uma estrutura de dados Pilha; e (iii) uma Fila. A memória de armazenamento (i) usa uma estrutura de dados *Heap* para armazenar as variáveis e objetos que estão em uso. A pilha (ii), que é uma unidade sequencial de serviço, contém *Frames* que são os contextos de execução, unindo funções, objetos e variáveis em algum lugar da Heap. A fila (iii) contém uma lista de mensagens esperando para serem processadas, onde cada mensagem se refere a uma função JavaScript. A estrutura do *Event Loop* é apresentada na Figura2.9.

[12] https://cplusplus.com/

Figura 2.9 – Estrutura do Event Loop



Fonte: (DAGGETT, 2013)

Para explicar como o *Event Loop* funciona o Código abaixo simula uma espera proposital no código para colocar em funcionamento os passos vistos anteriormente. Ao chamar a função *TestEventLoop()* é esperado que apareça no terminal os valores 1 e 2 que estão na função *console.log()*, porém a ordem em que irão aparecer é afetada pelo *Event Loop* pois uma das características da linguagem é ser não bloqueante, e, a função *setTimeout()* força a espera de um segundo para voltar a executar.

Ao chamar a função *setTimeout()*, uma mensagem é colocada na Fila, as referências de objetos e variáveis colocado na memória Heap e como a pilha está vazia a função ocupará o topo. Neste momento é dado continuidade no código, fazendo o mesmo processo com o segundo *console.log()* e colocando-o no topo da pilha. A pilha está com dois *Frames* agora, e sua execução é dada por LIFO (*last in, first out*) – nesse sistema o último que entra é o primeiro a sair da pilha. Ao finalizar a execução das funções, como o topo da pilha é o *console.log(2)* o resultado no terminal segue a ordem de desempilhamento como visto abaixo.

Listing 2.1 – Exemplo *Event Loop*

```
function TestEventLoop() {
  setTimeout(function () {
    console.log(1);
  }, 1000);
  console.log(2);
}

TestEventLoop();



Result: 2
Result: 1
```

2.3.3 GraphQL

GraphQL é uma linguagem de consulta para as API e também um *runtime* que independe do meio de transporte, mas, em geral, é servido sobre HTTP (PORCELLO; BANKS, 2018). Uma consulta GraphQL pede somente os dados de que necessita, na Figura 2.10 podemos observar uma consulta ao lado esquerdo que solicita dados de uma pessoa. Os dados requeridos são: *name*, *birthYear* e *created*. À direita o resultado é dado em um formato JSON, e, a resposta contém somente os dados que foi pedido.

Figura 2.10 – Uma consulta GraphQL

Fonte: (PORCELLO; BANKS, 2018)

Assim como SQL[13], o GraphQL pode realizar consultas de alterar, remover, criar e listar dados. Porém apesar das duas serem uma linguagem de consulta, seus ambientes são totalmente distintos. De acordo com Porcello e Banks (2018) enquanto as consultas SQL vão para o banco de dados, as consultas GraphQL são enviadas para uma API. Dados para SQL são armazenados em tabelas de dados, dados para GraphQL podem ser armazenados em qualquer lugar, portanto define-se que GraphQL é uma linguagem de consulta para a internet.

A estrutura do GraphQL utiliza apenas duas palavras especiais para todo tipo de manipulação de dados, sendo elas *Mutation* (Figura 2.11) e *Query* (Figura 2.12). Fazendo um paralelo com a linguagem SQL, a *Mutation* é similar ao *CREATE*, *UPDATE* E *DELETE*, e a *Query* representa o comando *SELECT*.

Figura 2.11 – Exemplo de uma estrutura de uma Mutation

Fonte: https://graphql.org/learn/queries/

Figura 2.12 – Exemplo de uma estrutura de uma Query

Fonte: https://graphql.org/learn/queries/

2.3.4 Apache Kafka

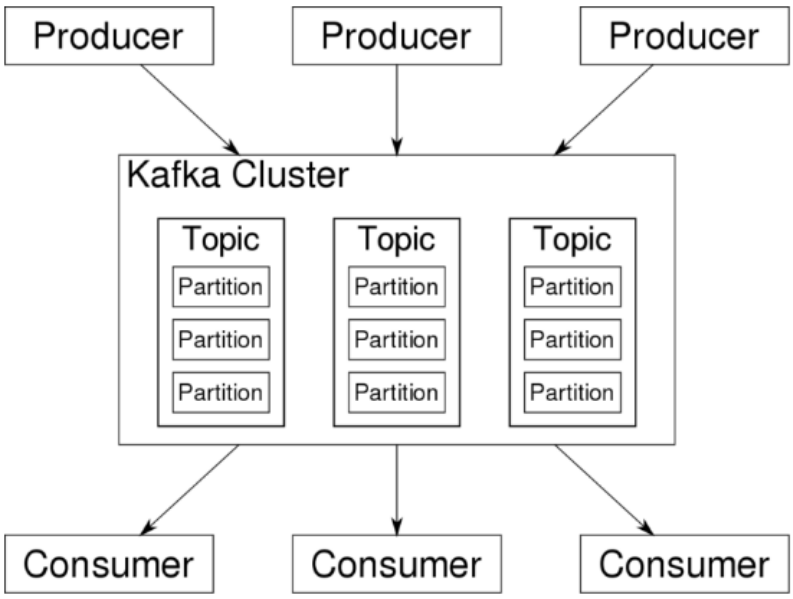
O Apache Kafka é uma plataforma de transmissão de dados de código aberto que é capaz de publicar, subscrever, armazenar e processar fluxos de registro em tempo real (ESTRADA, 2018). Essa plataforma foi desenvolvida para processar fluxos de dados provenientes de diversas fontes e

[13] https://www.w3schools.com/sql/

entregá-los a vários clientes. O Kafka é uma alternativa aos sistemas de mensageria empresariais tradicionais. Inicialmente, ele era um sistema interno desenvolvido pela LinkedIn para processar 1,4 trilhão de mensagens por dia, mas agora é uma solução de transmissão de dados aplicável a variadas necessidades empresariais.

Ele realiza as suas transmissões de maneira assíncrona e pode ser considerado como um sistema distribuído. Segundo (NARKHEDE; SHAPIRA; PALINO, 2017), em sua arquitetura na Figura 2.13, o Kafka possui algumas estruturas como: *Broker*, um servidor de envio de mensagens. É ele que irá disparar e receber essas mensagens; Tópicos, um tópico é um identificador. Por meio desse identificador, será possível se conectar ao *broker* e disparar e receber as mensagens; Partição, subdivisão de um tópico, a partição é um recurso para ajudar a balancear a carga; *Cluster*: resumidamente, um cluster é um conjunto de brokers, nele serão contidos os servidores e a instância principal do Kafka.

Figura 2.13 – Arquitetura do Apache Kafka

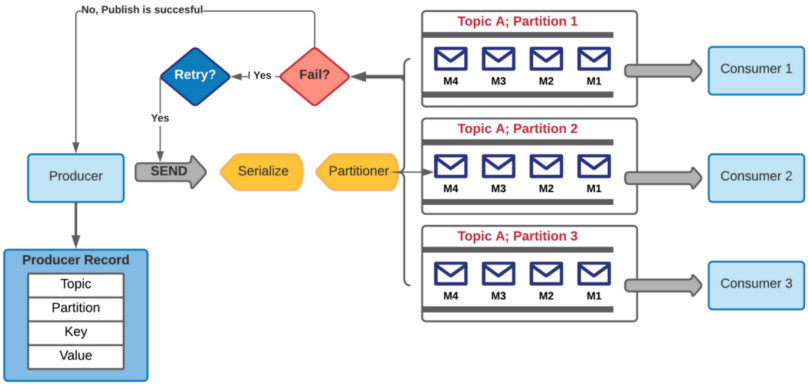


Fonte: https://luby.com.br/desenvolvimento-de-software/programacao/o-que-e-apache-kafka/

O transporte de mensagem possui dois processos, um chamado de *producer* que envia mensagens para um tópico, que é gerenciado e armazenado por um *cluster* de *brokers*, é um processo chamado *consumer* que subscreve ao tópico para realizar a leitura e processamento de tais mensagens.

As mensagens produzidas são enviadas para um tópico e qualquer aplicação que se conectar a esse tópico poderá consumir a mensagem que está na fila, portanto entende-se que uma fila pode ser consumida por uma ou mais aplicações, assim como várias aplicações podem produzir mensagem para um ou mais tópicos. Na Figura 2.14 é possível observar o fluxo de mensagens que é feito pelo Kafka

Figura 2.14 – Fluxo de mensagens do Apache Kafka



Fonte:
https://medium.com/swlh/how-to-make-kafka-producer-consumer-production-ready-d40915b78c9

2.3.5 MongoDB

MongoDB é um banco de dados não relacional que não possui conceitos de tabelas, esquemas, SQL ou linhas. Não há transações, conformidade com ACID, junções e chaves estrangeiras (HOWS; MEMBREY; PLUGGE, 2019). MongoDB trabalha com o conceito de documentos. Um conjunto ordenado de chaves associado a valores. A representação destes documentos difere dependendo da linguagem de programação, mas a maioria delas possui uma estrutura de dados que os relaciona naturalmente (CHODOROW; DIROLF, 2010). Em JavaScript por exemplo, documentos são representados por objetos como na Figura 2.15.

Figura 2.15 – Documento MongoDB

```
{"greeting" : "Hello, world!"}
```

Fonte: (CHODOROW; DIROLF, 2010)

No MongoDB os documentos são armazenados em uma estrutura chamada de coleção. Segundo Chodorow e Dirolf (2010) uma coleção pode armazenar documentos que não são os mesmos em estrutura. Isso é possível porque o MongoDB é um banco de dados sem esquema. Em um banco de dados relacional como MySQL, Oracle, etc, um esquema define a organização/estrutura de dados no banco de dados. O MongoDB não exige tal conjunto de fórmula que define a estrutura de dados. Assim, é perfeitamente possível armazenar documentos de diferentes estruturas em uma coleção. A Figura 2.16 exemplifica com o banco de dados *sample training*, que possui as coleções *animals*, *companies*, *grades*, etc e ao lado direito da figura um documento da coleção *animals*.

Figura 2.16 – Coleção MongoDB

Fonte: Do autor

2.4 Controle de Versão

Controle de versão é uma prática da gerência de configuração responsável por gravar as mudanças em arquivos durante o tempo em que é construído. Sendo utilizado por sistemas de controle de versão que implementam a prática que torna possível criar versões específicas da aplicação (CHACON; STRAUB, 2014). Por exemplo, um cliente A precisa de certas funcionalidades de uma aplicação, mas a empresa cria novas funcionalidades e atualiza a aplicação. Visto que o cliente A não utilizaria essas novas funcionalidades, a empresa pode utilizar a estratégia de controle de versão, onde em uma versão 1.0 é possível satisfazer o cliente A, e uma versão 2.0 estão as novas funcionalidades.

O software de controle de versão mantém registro de todas as modificações no código em um tipo especial de banco de dados. Se um erro for cometido, os desenvolvedores podem voltar no tempo e comparar versões anteriores do código para ajudar a corrigir o erro enquanto diminuem interrupções para todos os membros da equipe. Desenvolvedores de software que trabalham em equipes estão sem-

pre escrevendo novo código-fonte e modificando código-fonte existente. O código de um componente de projeto, aplicativo ou software é, no geral, organizado em uma estrutura de pasta ou "árvore de arquivos". Um desenvolvedor na equipe pode estar trabalhando em um novo recurso, enquanto outro desenvolvedor corrige um *bug* não relacionado, alterando o código; cada desenvolvedor pode fazer as alterações em várias partes da árvore de arquivos.

Os benefícios de utilizar controle de versão no projeto é o armazenamento de um histórico de alterações completo e a longo prazo de todos os arquivos, isso significa todas as alterações feitas por muitas pessoas ao longo dos anos. As alterações incluem a criação e exclusão de arquivos, assim como as edições em seus conteúdos. Ramificação e mescla, o trabalho simultâneo da equipe é certo, mas mesmo os indivíduos que trabalham sozinhos podem se beneficiar da capacidade de trabalhar em fluxos independentes de mudanças. Tal como rastreabilidade, ser capaz de rastrear cada alteração feita no software e conectar ao software de gestão de projetos e rastreamento de *bugs*, e ser capaz de anotar cada alteração com uma mensagem descrevendo o objetivo da mudança ajuda não só com a análise de causa raiz e outras análises forenses (Atlassian,).

2.4.1 Git

Segundo Uzayr (2022) Git é um *software* de versionamento de controle para rastreamento de mudanças em conjuntos de arquivos, para assegurar o trabalho em conjunto de programadores que estão desenvolvendo um software. É um *software* de código aberto distribuído sobre a licença de GNU[14], desenvolvido por *Linus Torvalds* em 2005 (UZAYR, 2022).

Com Git, a maioria das operações requer apenas recursos locais e arquivos para propósito de operações. Visto isso, Git possui uma enorme velocidade de processamento, fazendo com que as operações sejam praticamente instantâneas. Utilizando apenas duas estrutura de dados para processamento, index mutável, o que se refere ao processo de *stage* ou *cache* e uma estrutura *append-only object database*, ou repositório, que é imutável e contém os arquivos a serem submetidos.

Outro ponto importante do Git é que ele possui três principais estágios que contém os arquivos, *Modified*, implica que houve mudanças no arquivo mas que ainda não foram submetidas ao repositório;

[14] https://gcc.gnu.org/

Stage, o estado em que marca as modificações feitas que estão aptas para submissão; e *Committed*, que indica que as modificações foram submetidas no repositório (UZAYR, 2022).

2.4.2 BitBucket

Bitbucket é uma ferramenta de hospedagem e colaboração com ambiente é favorável para o compartilhamento de dados, criação e implantação de códigos e automatização de testes no melhor conceito de *cloud computing* e infraestruturas *on premises* (VOGEL; BLEWITT, 2014). Utilizada para hospedagem e colaboração de código baseada em Git, criada para equipes profissionais de engenharia de software e gestão de projetos. A marca BitBucket foi adquirida pela empresa Atlassian em 2010, e garante integração com suas outras ferramentas de gestão.

O BitBucket possui conectividade de rede e largura de banda E/S[15], dessa forma é possível ter um maior poder de computação sempre que for necessário dar upgrade. Com isso, mesmo com uma alta demanda de trabalho do servidor a performance não será afetada. Assim, vários usuários podem trabalhar simultaneamente e um dos grandes diferenciais é a sua alta disponibilidade. Quando um elemento do *cluster*[16] tem problema, os demais conseguem suprir e atender as solicitações sem perda de eficiência.

[15] https://www.ibm.com/docs/pt/powerlinux-systems?topic=scsi-bandwidth
[16] https://www.opservices.com.br/o-que-e-um-cluster/

3 ATIVIDADES DESENVOLVIDAS

Este capítulo descreve as atividades desenvolvidas no estágio supervisionado realizado na empresa Voll. A Seção 3.1 apresenta o processo de desenvolvimento adotado na organização. As Seções 3.2 a 3.4 descrevem as atividades realizadas em cada um dos times da empresa em que o estagiário atuou, sendo, respectivamente, os times "Front-end", "Travel", "Mobility". Por fim, a Seção 3.5 discute os desafios e aprendizados obtidos durante o estágio.

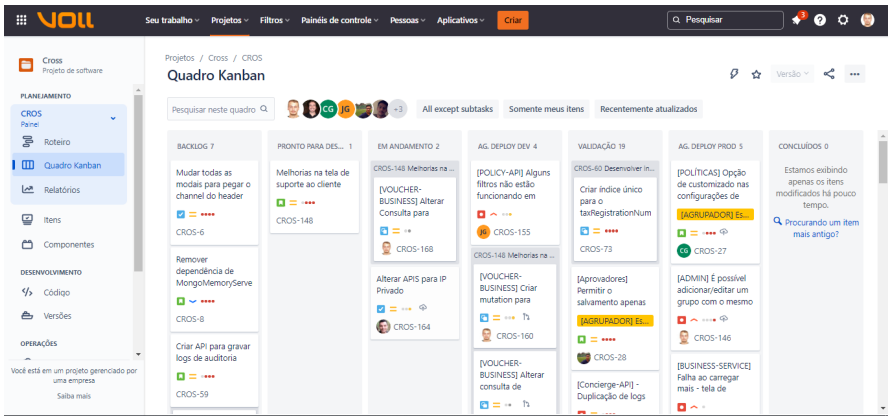
3.1 Organização, armazenamento e metodologias

A VOLL fornece serviços de mobilidade para empresas contratantes por meio do aplicativo VOLL. Desta forma, a empresa adota o conceito *Mobile First*, onde o foco inicial da arquitetura e desenvolvimento é direcionado aos dispositivos móveis.

A gestão dos times de desenvolvimento adota conceitos ágeis. Os times possuem de 3 a 5 pessoas. As demandas de desenvolvimento são executadas em ciclos curtos, baseados nas necessidades de cada tarefa. São realizadas reuniões diárias (*dailies*) curtas e objetivas, similares as reuniões diárias do *Scrum* [1]. As tarefas são gerenciadas com apoio da ferramenta Jira [2], utilizada para manter o quadro Kanban do time. O Kanban[3] exemplificado na figura 3.1 possui 7 colunas padrões: "*BACKLOG*", onde ficam as tarefas que ainda estão sendo preparadas para desenvolver; "PRONTO PARA DEV", onde ficam as tarefas prontas para desenvolver; em andamento, tarefas em desenvolvimento; "AG. DEPLOY DEV´´, tarefas esperando para homologação; "VALIDAÇÃO DE PRODUTO´´, esperando confirmação de funcionalidade; "AG. DEPLOY PROD´´, envio para produção e concluído. Esta divisão pode mudar dependendo do time, porém o fluxo de serviço é sempre o mesmo: preparação, em andamento, validação e concluído.

[1] https://www.atlassian.com/br/agile/scrum
[2] https://4infra.com.br/jira-software-o-que-e-e-para-que-serve/
[3] https://kanbanize.com/pt/recursos-kanban/primeiros-passos/o-que-e-kanban

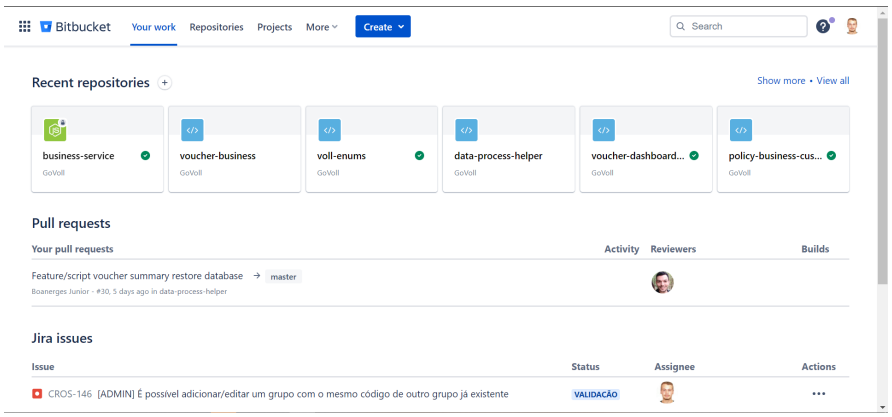
Figura 3.1 – Jira



Fonte: Jira - VOLL

Para gerenciar o código produzido, é utilizada a plataforma BitBucket (Figura 3.2), que integra a ferramenta Git para versionamento de código. A plataforma possui todas as funcionalidades necessárias para CI/CD (fluxo de integração contínua e entrega contínua), configurações de acesso e integração com Jira.

Figura 3.2 – BitBucket



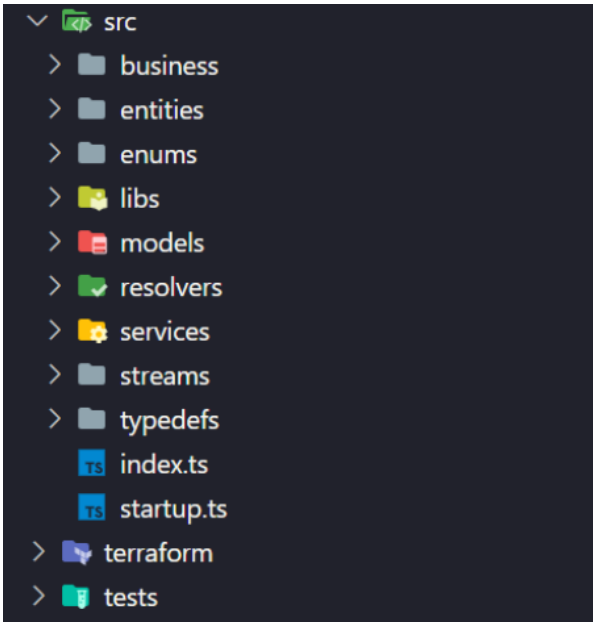
Fonte: BitBucket - VOLL

Sempre que uma tarefa é entregue, uma *pull request* é iniciada pelo BitBucket. O processo de *pull request* consiste nos passos de análise e aprovação da tarefa feita, que é solicitado quando um código irá ser mesclado na *Branch* de desenvolvimento ou produção. A aprovação das *pull requests* é responsabilidade do líder técnico de cada time. Para isto, são feitas revisões de código (*code reviews*)

analisando a lógica, a sintaxe, testes unitários e de integração. colocar imagem aprovação pr. Uma vez aprovada, o código é integrado ao ambiente de desenvolvimento para realização de outros testes funcionais.

Todos os repositórios de projetos da empresa seguem uma organização padrão de diretórios considerando linguagem (JavaScript, TypeScript e Dart) e área de aplicação (*Front-End* e *Back-End*). O objetivo é garantir a modularização dos micro serviços (separados por diretórios, com nomeações autoexplicativas). Esta organização atende ao primeiro tópico do princípio SOLID – Single Responsibility Principle (Princípio da Responsabilidade única). A Figura 3.3 e Figura 3.4 ilustram a organização de diretórios para projetos Back-End e Front-End, respectivamente.

Figura 3.3 – Organização de diretórios para Back-End

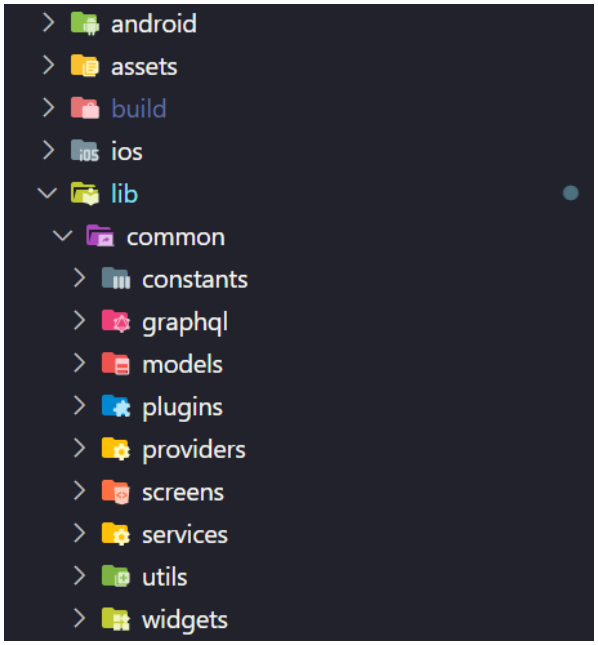


Fonte: VOLL

Para o Back-end das aplicações (Figura 3.3), o diretório “src” contém o coração da API, tem o arquivo “index.ts” que é o ponto inicial e “startup.ts” com as configurações de inicialização. Em seguida,o diretório “entities” é padrão em toda aplicação VOLL, e armazena as entidades do graphQL. O diretório “enums” armazena os tipos fixos que servirão como *template*. O diretório “libs” armazena funções específicas como conversão de data. O diretório “models” intercepta o acesso da

regra de negócio ao banco de dado. resolvers: definem as *queries* e *mutations* do graphQL; services: implementam as regras de negócio definidas pelo líder; stream: fazem o transporte das mensagens recebidas via Kafka; typedefs: descreve as variáveis e tipos que as querys e mutations devem possuir; terraform: armazena as variáveis de ambiente para configuração de infra-estrutura do projeto; tests: armazena os testes integrados e unitários.

Figura 3.4 – Organização de diretórios para Front-End



Fonte: VOLL

Para o Fronmt-end das aplicações (Figura 3.3), temos a pasta principal “lib” onde todas as pastas e arquivos principais são concentrados e a pasta “common” é a mais relevante para explicação. Podemos observar que assim como o Back-End, que cada pasta possui seu propósito bem definido já pelo nome. Seguindo temos as pastas, constants: armazena todas os tipos e funções imutáveis; graphQL: contém as entidades do graphQL; models: funções que interceptam o acesso direto ao banco de dados; plugins: integra telas, componentes e serviços externos; providers: responsável por armazenar os estados da aplicação; screens: armazena as telas do aplicativo; services: as regras de negócio que a empresa possui; utils: funções e métodos utilitários, como por exemplo uma conversão de data específica e widgets: que fornece todos os componentes modularizados para uso e reuso.

3.2 Time de Front-end

O produto carro chefe da VOLL é um aplicativo. Assim é importante garantir a excelência e eficácia da experiência do usuário. O Time de Front-end é focado em realizar todos os serviços ligados ao cliente e trata de toda a parte visual da aplicação web. Ou seja, tudo que o usuário interage no sistema é gerado e regido por este time. Porém, as atividades do time vão muito além de apenas fazer telas e filtros. Também são responsabilidades do time a análise e melhoramento do tempo de carregamento das telas, usabilidade da aplicação, identidade visual da aplicação, e outras. Portanto, é comum a aplicação de conhecimentos diversos como teoria das cores, definição dos processos e etapas que o usuário terá de percorrer e na interação que ele terá com as interfaces de seu site ou aplicativo.

Utilizando a linguagem de programação Dart e seu *framework* Flutter, o primeiro trabalho realizado no estágio foi a elaboração de uma tela de filtros para pesquisa de *vouchers* criados na base de dados. O filtro consiste em pesquisas com período de criação do *voucher*, centro de custo, cidade, valores, *status* e categoria.

A tela da Figura 3.14, apesar de simples, envolve algumas técnicas e tecnologias específicas. Para comunicar a aplicação Front-End com a aplicação Back-End foi utilizado uma linguagem de *query* chamada GraphQL. Esta linguagem fornece meios para criar, listar, atualizar e remover dados do banco de dados, ou seja, realizar um CRUD[4]. Para as operações de modificação de dados, é utilizado a palavra reservada *mutation* e para listagem de dados a palavra *query*. Na Figura 3.5 é possível observar a estrutura de uma chamada utilizado *query*.

Figura 3.5 – Estrutura de Query

```
const String voucherFilterDetails = r'''
query vouchersFilter($input: VoucherFilter!, $pagination: PageInfoInput) {
  vouchersFilter(pagination: $pagination, input: $input) {
```

Fonte: VOLL

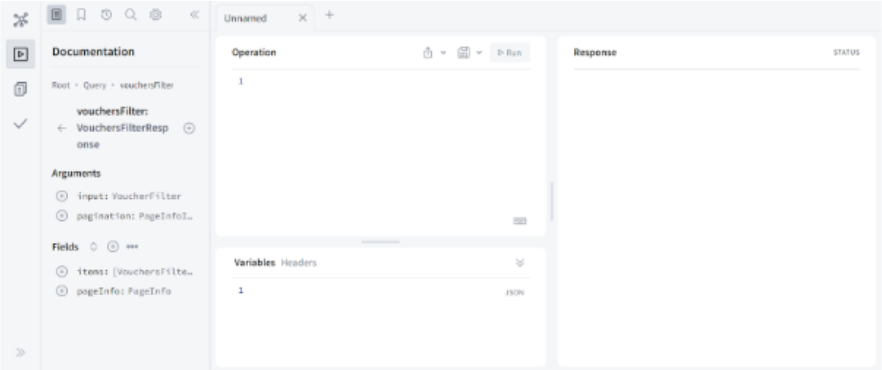
Estas *queries* e *mutations* são armazenadas em nuvem pela plataforma de serviço Apollo Supergraph[5]. Este serviço centraliza todas as chamadas da aplicação e funciona disponibilizando-as

[4] https://developer.mozilla.org/pt-BR/docs/Glossary/CRUD

[5] https://www.apollographql.com/blog/announcement/backend/the-supergraph-a-new-way-to-think-about-graphql/

como sendo uma biblioteca de chamadas, assim é possível gerenciá-las como sendo um micro serviço. A Figura 3.6 apresenta um exemplo de *Dashboard* da plataforma Apollo.

Figura 3.6 – Dashboard Apollo



Fonte: VOLL - Apollo

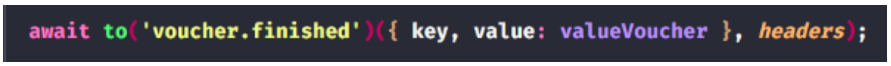
3.3 Time de Travel

O Time de Travel remete aos serviços relacionados as viagens longas e duradouras, como por exemplo a necessidade de um colaborador viajar a outra filial da empresa em outro estado, todo contexto de passagens de avião, ônibus, estadia em hotéis e consumação são de responsabilidades deste time. No Time de Travel, os serviços realizados passam a ter um foco maior em Back-End, sendo mais especificamente um time de resolução de problemas, ligado diretamente a problemas dos usuários, como *bugs* visuais, verificar informações incorretas na aplicação, dados não consistentes e funcionalidades em desenvolvimento para soluções existentes. Tem como objetivo desenvolver, resolver e solucionar problemas encontrados no dia a dia dos produtos fornecidos e cuidar de toda parte que gere a experiência do cliente do lado do Back-End utilizando o software NodeJS.

Em Seguida, tem a *voucher-api*, uma das principais API que a VOLL mantém, sendo ela praticamente o núcleo da empresa, onde é concentrado toda a finalização de um processo de requisição de *voucher*. Esta API mantém, desenvolve e controla serviços de passagens aéreas, ônibus, aluguel de carro, entre outros.

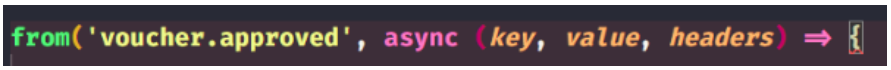
Como os serviços da empresa são baseados em micro serviços, há ferramentas para auxiliar a comunicação entre eles. Utilizando o conceito de “mensageria”, que define que sistemas distribuídos possam se comunicar por meio de troca de mensagens, foi utilizado a plataforma de código aberto Kafka. Seu funcionamento no código consiste em um um produtor, podendo ser o próprio micro serviço em questão, ou um externo que produz uma mensagem, e um consumidor responsável por consumir a mensagem produzida. A VOLL possui diversos pacotes que abstraem o conteúdo dessas ferramentas, como por exemplo o `voll-kafka` que traz uma abstração de como é implementado essas técnicas do Kafka. Nas Figuras 3.7 e 3.8 temos um exemplo de como essa abstração é feita em código.

Figura 3.7 – GraphQL - Producer



Fonte: VOLL

Figura 3.8 – GraphQL - Consumer



Fonte: VOLL

Após a recepção dessas mensagens, o conteúdo é manipulado de acordo com as regras de negócios da empresa. Tendo os dados tratados é feito o envio para o banco de dados MongoDB. Este envio é feito por meio de modelos padronizados chamados de *models*, na Figura 3.9 é possível observar um *model* simples padronizado. Estes *models* interceptam os dados antes da inserção, criando uma camada antes de utilizar códigos nativos do banco de dados (Figura 3.10), deste modo caso haja alguma alteração futura na estrutura de conexão com o banco, não há necessidade de refatorar a maneira que os dados são salvos no banco.

Figura 3.9 – Exemplo de model

```
module.exports = async (voucher) => {
  await insertOne('vouchers', voucher);
  return voucher;
};
```

Fonte: VOLL

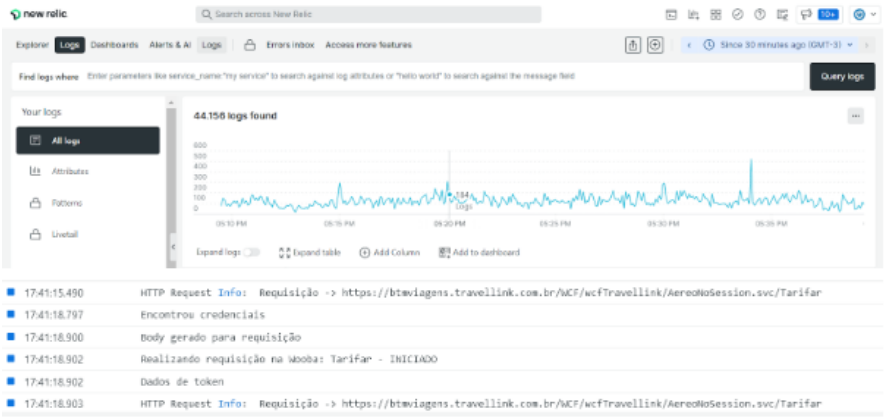
Figura 3.10 – Método de inserção nativo do MongoDB

```
const insertOne = (collectionName, object) =>
  cacheDb.insertOne(collectionName, object);
```

Fonte: VOLL

Todo o percurso é monitorado pelas ferramentas Cloudwatch da AWS e a ferramenta de monitoramento New Relic (Figura 3.11), deste modo é possível rastrear todo o caminho percorrido pelo dado e ter uma noção da estrutura de comunicação entre micro serviços, e caso haja algum erro durante a manipulação dos dados, essas ferramentas irão sinalizar em *Logs* programados.

Figura 3.11 – Dashboard de logs do NewRelic



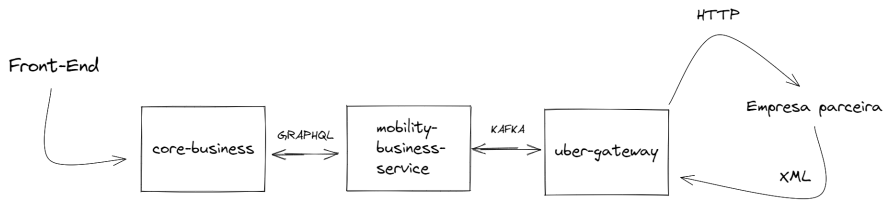
Fonte: VOLL

3.4 Time de Mobilidade

O time de mobilidade tem a função de gerenciar as corridas relacionados a *App's* de terceiros que tratam com veículos, como por exemplo uma corrida feita pela *uber* é chamada pelo aplicativo VOLL (Figura 3.13). Assim os dados não serão consumidos pela empresa parceira e sim pela VOLL. Considerando esta situação, a responsabilidade do time consiste em automatizar o processo de consumo destes serviços, gerenciar serviços de agendamento de corridas, melhorar a infraestrutura de comunicação das *API*, suporte e manutenção nos micro serviços relacionados.

Para receber os dados de *API* externas foram criados microsserviços para manipular os conteúdos antes de serem utilizados, estes são chamados de *Gateways*. Eles possuem a função de fazer o tratamento e normalização dos dados para que possam ser compatíveis com a estrutura interna da VOLL. Por exemplo, o micro serviço responsável pelo gerenciamento dos dados da empresa parceira recebe os dados no formato *XML* e mapeia para *JSON*, também implementa os métodos *WebHook's* da aplicação sendo feita a chamada entre os micro serviços da seguinte forma: *core-business* é responsável pelas chamadas do cliente (*Front-End*), *mobility-business-service* onde é definido as regras de negócio de corrida e *uber-gateway* a parte da comunicação com a api da uber (Figura 3.12)

Figura 3.12 – Dashboard de logs do NewRelic



Fonte: Do autor

Figura 3.13 – EndPoint para API da uber

```
export async function syncUber(clientId: number) {
  const baseURL = config.get('UBER_SYNC_EXPENSE_URL');
  const url = `${baseURL}/sync/expense`;
```

Fonte: VOLL

Para faturar a corrida é disponibilizado para o cliente as formas de crédito e débito. Para o crédito é feito um depósito no aplicativo VOLL e atualizado o saldo do cliente. Com este saldo é possível realizar corridas consumindo-o de acordo com regras estabelecidas pela própria empresa contratante. O débito é feito da maneira tradicional no cartão do cliente ou em dinheiro. Essas regras são configuradas dentro do aplicativo pela empresa do cliente que contratou o serviço VOLL. É possível especificar valores máximos de corridas que o usuário pode solicitar, locais onde é permitido ir, saldo disponível para gastar, entre outros, desta forma o cliente pode viajar sem preocupações com o reembolso.

3.5 Atividades desenvolvidas em cada time

Explicado a área de atuação em cada time, esta seção tem como objetivo exemplificar ao menos uma tarefa realizada pelo estagiário durante o período em que passou em cada time.

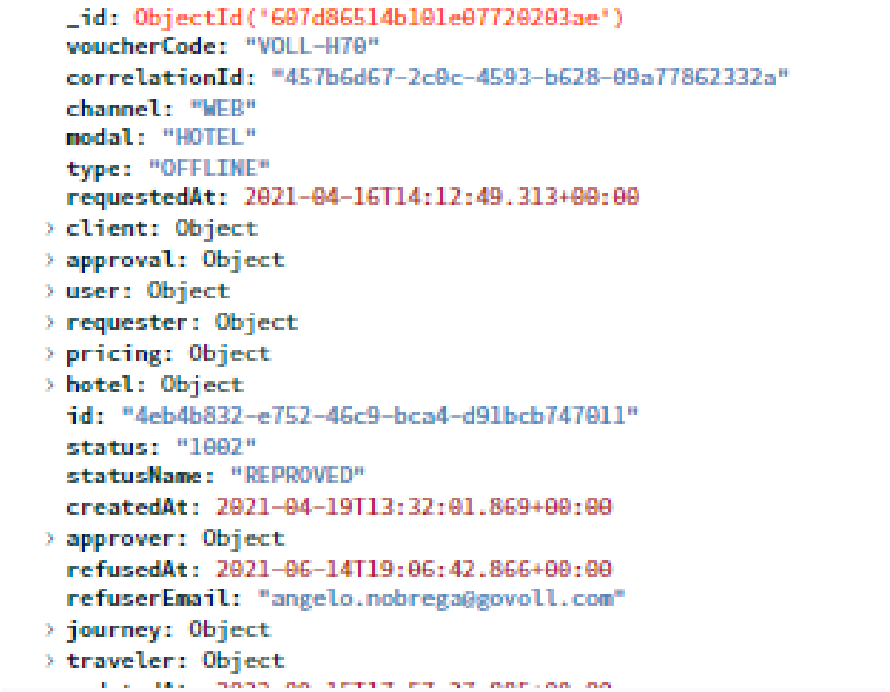
Como dito na seção 3.2 o time é responsável principalmente pela interface do usuário, e uma das tarefas realizadas foi a implementação e componentização de interfaces. A tela apresentada na Figura 3.14 é uma tarefa genérica para implementação de telas de filtros. Isto é, todas as demais telas de filtros desenvolvidas para a aplicação utilizam esta tela como base, apenas alterando os atributos usados para a filtragem

Figura 3.14 – Tela genérica do aplicativo VOLL

Fonte: VOLL

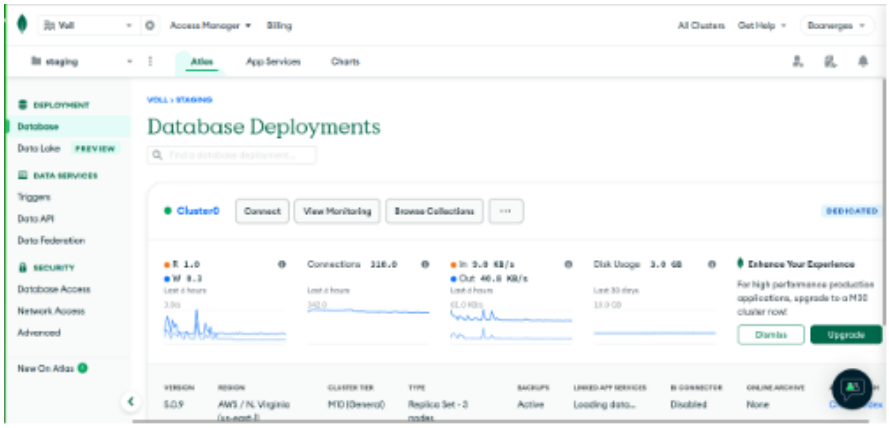
No Time de Travel, mais focado em suporte ao cliente, Uma tarefa rotineira que era requisitada com mais influência direta de clientes, era a alteração de dados direto na base de dados, como por exemplo valor total do *voucher*, *status* e data de emissão, entre outros como visto na Figura 3.15. Nesta tarefa o time responsável por criar a ordem de serviço é o time de *Customer Success*. A tarefa consiste em alterar os dados requisitados na base de dados de produção, acessando diretamente a nuvem onde são armazenados os dados, chamado de MongoDB Atlas Database (Figura 3.16).

Figura 3.15 – Dados de um voucher - MongoDB



Fonte: VOLL - Mongo

Figura 3.16 – Atlas - MongoDB



Fonte: VOLL - Atlas

Por fim, no Time de Mobilidade uma das tarefas mais importantes era a aferição da saúde dos dados. Como são feitas milhares de requisições por dia no aplicativo VOLL, é extremamente importante manter a consistência dos dados durante todo o processo de criação do *voucher*, por isso é utilizado software de observabilidade para ajudar no processo. O mais utilizado é o NewRelic e com eles podemos monitorar todos os processos envolvidos no trajeto do dado, sendo possível identificar falhas de segurança, inconformidade nos dados e a criação efetiva de um *voucher*. Na Figura 3.17 podemos observar em destaque quais os caminhos que os dados estão tomando os longo do processo

Figura 3.17 – NewRelic Logs

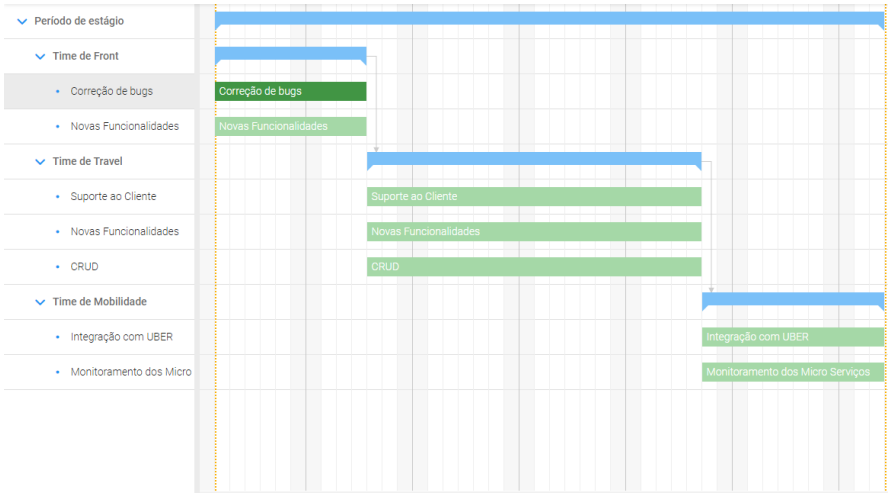
timestamp	message
13:54:00.594	SEND.TO.CONCLUDED.SERVICE -->
13:54:02.222	send.notification.requested
13:54:02.255	Generic scheduler requested:
13:54:02.312	Envio de email - INICIO.
13:54:02.313	HTTP Request Info: Requisição -> https://business-service.core.govoll.com/graphq
13:54:02.314	HTTP Request Info: Requisição response -> https://business-service.core.govoll.c
13:54:02.315	Já existe email enviado para a mesma chave, será feito um reagendamento.
13:54:02.315	Consultando cache - THROUGHPUT_CONTROL.

Fonte: VOLL - NewRelic

Explicado algumas tarefas, a divisão em relação ao tempo pode ser vista no gráfico abaixo (Figura 3.18), que mostra como teria sido o processo a época de passagem pelos times e as tarefas

inclusas levando em consideração que o tempo passado nos time de Front e Mobilidade são pouco mais de um mês, e o restante no time de Travel.

Figura 3.18 – Gráfico de Gantt do periodo de estágio



Fonte: Do Autor

3.6 Considerações finais

Com base no conteúdo apresentado, este capítulo mostrou as atividades realizadas em cada time em que o estagiário fez parte. Foi possível analisar como e onde foi aplicado o conhecimento adquirido durante o curso e como eles impactaram no desenvolvimento do aplicativo VOLL. Além disso, também foi adquirido conhecimento de novas técnicas e metodologias que o mercado mais utiliza, tendo em vista que as tecnologias usadas são as mais modernas no momento.

Em relação às técnicas utilizadas, foi um desafio entender e aplicar principalmente os conceitos utilizados nas API de *gateways*, tal como a manipulação de arquivos XML que as empresas parceiras utilizam, porém com o auxílio dos supervisores de estágio e professores da universidade tornou-se um obstáculo conquistado pelo estagiário e contribuiu para um processo de grande aprendizado que afirmou a permanência do estagiário na área e o envolvimento com a indústria tecnológica sênior.

Com o conhecimento adquirido durante o estágio, ficou clara a diferença entre o início e o fim do período. Levando em consideração que o estagiário já tinha conhecimento de Back-End com

NodeJS, ao fim do estágio estava apto a realizar tarefas multidisciplinares, como Front-End utilizando Flutter, tarefas de banco de dados e arquitetura no sentido de otimização dos serviços. Deste modo é expressivo o crescimento profissional do estagiário em vista do tempo investido e conhecimento aplicado, que agora faz parte dos colaboradores efetivos da empresa.

4 CONCLUSÃO

O estágio consistiu na aplicação de conhecimentos de desenvolvimento web, tal como o estudo de novas tecnologias e metodologias que a empresa aplica no processo de construção do aplicativo.

As atividades realizadas são as mais diversas e abordam uma alta gama de assuntos. Tendo em vista o preparo do curso de Ciência de Computação, principalmente nas matérias de Estrutura de Dados e Engenharia de Software, é possível concluir que o conteúdo aprendido foi efetivamente aplicado, desde tecnologias que o estagiário já conhecia anteriormente, até metodologias de pesquisa que eram necessárias para conhecer outras ferramentas.

Ao final do período de estágio, é possível concluir que o contato dos estudantes de graduação com as tecnologias atuais de mercado é indispensável na hora de realizar o serviço, tal como a experiência de problemas reais que surgem durante o desenvolvimento. Uma vez que os problemas enfrentados durante o curso de graduação não englobam todas as variáveis de uma empresa real é de extrema importância a experiência adquirida em um estágio na área e a melhor maneira de testar os conhecimentos do aluno, pois ele consegue sentir como é estar em um ambiente do mercado de trabalho e ao mesmo tempo é reservado das responsabilidades críticas que uma empresa demanda.

Vale ressaltar também a importância das empresas juniores que existem nas universidades, onde é possível adquirir um conhecimento prévio de como é vivenciar um trabalho real, mas que demanda responsabilidade similares às empresas seniores. Tomando como prova, o período em que o estagiário foi membro de uma empresa júnior da Universidade Federal de Lavras, que aplicava os conceitos aprendidos durante o curso e mantinha comunicação com empresas seniores do mercado de trabalho para trazer conteúdo e visões profissionais para os membros. E que, portanto, empresas juniores deveriam ser o ponto principal de comunicação, conhecimento e experiências do aluno.

Por fim, baseado nas experiências adquiridas como estagiário, como sugestões para o curso de Ciências da Computação, seria de grande valia disciplinas que abordassem efetivamente assuntos do mercado de trabalho, como por exemplo o uso de Conteinerização de aplicações que é uma das principais tecnologias quando o assunto é disponibilizar o serviço ao cliente, tal como a automação de processos, CI/CD, conhecimentos de pipeline e infraestrutura de projetos. Já para a empresa, como sendo uma *startup* e estar desenvolvendo sua própria cultura, como sugestão para o time de desenvol-

vimento seria investir mais tempo no estagiário no que diz respeito a apresentação dos processos da empresa, pois a adaptação de novas pessoas no time é um dos principais pontos que um desenvolvedor considera ao entrar ou mudar para uma empresa, considerando também o tempo em que ele demora para conhecer o fluxo de trabalho até que esteja apto a atuar em diversas tarefas.

REFERÊNCIAS

AQUINO, C.; GANDEE, T. **Front-End Web Development: The Big Nerd Ranch Guide**. Pearson Education, 2016. (Big Nerd Ranch Guides). ISBN 9780134432571. Disponível em: <https://books.google.com.br/books?id=Wlu8DAAAQBAJ>.

Atlassian. Https://www.atlassian.com/br/git/tutorials/what-is-version-control.

BITENCOURT, J. **O guia de Dart: Fundamentos, prática, conceitos avançados e tudo mais**. Casa do Código, 2022. ISBN 9788555192999. Disponível em: <https://books.google.com.br/books?id=oDJtEAAAQBAJ>.

CHACON, S.; STRAUB, B. **Pro Git**. Apress, 2014. (The expert's voice). ISBN 9781484200766. Disponível em: <https://books.google.com.br/books?id=jVYnCgAAQBAJ>.

CHODOROW, K.; DIROLF, M. **MongoDB: The Definitive Guide**. O'Reilly Media, 2010. ISBN 9781449396985. Disponível em: <https://books.google.com.br/books?id=BQS33CxGid4C>.

COULOURIS, G. et al. **Sistemas Distribuídos - 5ed: Conceitos e Projeto**. Bookman Editora, 2013. ISBN 9788582600542. Disponível em: <https://books.google.com.br/books?id=6WU3AgAAQBAJ>.

DAGGETT, M. **Expert JavaScript**. Apress, 2013. (Expert's voice in Web development). ISBN 9781430260981. Disponível em: <https://books.google.com.br/books?id=HpoQAwAAQBAJ>.

DRESHER, T.; ZUKER, A.; FRIEDMAN, S. **Hands-On Full-Stack Web Development with ASP.NET Core: Learn end-to-end web development with leading frontend frameworks, such as Angular, React, and Vue**. Packt Publishing, 2018. ISBN 9781788627757. Disponível em: <https://books.google.com.br/books?id=Tel1DwAAQBAJ>.

ESTRADA, R. **Apache Kafka Quick Start Guide: Leverage Apache Kafka 2.0 to simplify real-time data processing for distributed applications**. Packt Publishing, 2018. ISBN 9781788992251. Disponível em: <https://books.google.com.br/books?id=DtCBDwAAQBAJ>.

FAIN, Y. et al. **Enterprise Web Development: Building HTML5 Applications: From Desktop to Mobile**. O'Reilly Media, 2014. ISBN 9781449357061. Disponível em: <https://books.google.com.br/books?id=xF_xAwAAQBAJ>.

FLANAGAN, D. **JavaScript: O Guia Definitivo**. Bookman Editora, 2013. ISBN 9788565837484. Disponível em: <https://books.google.com.br/books?id=zWNyDgAAQBAJ>.

Hostinger. Disponível em: https://www.hostinger.com.br/tutoriais/linguagens-de-programacao-mais-usadas.

HOWS, D.; MEMBREY, P.; PLUGGE, E. **Introdução ao MongoDB**. Novatec Editora, 2019. ISBN 9788575228081. Disponível em: <https://books.google.com.br/books?id=MvPCDwAAQBAJ>.

IHRIG, C. **Pro Node.js for Developers**. Apress, 2013. (Expert's voice in Web development). ISBN 9781430258605. Disponível em: <https://books.google.com.br/books?id=nO9ZAgAAQBAJ>.

Lars Bak e Kasper Lund. 2011. Disponível em: <https://dart.dev/>. Acesso em: 19 de Agosto 2022.

NAPOLI, M. **Beginning Flutter: A Hands On Guide to App Development**. Wiley, 2019. ISBN 9781119550822. Disponível em: <https://books.google.com.br/books?id=ex-tDwAAQBAJ>.

NARKHEDE, N.; SHAPIRA, G.; PALINO, T. **Kafka: The Definitive Guide : Real-time Data and Stream Processing at Scale**. O'Reilly Media, 2017. ISBN 9781491936160. Disponível em: <https://books.google.com.br/books?id=qIjQjgEACAAJ>.

PHANG, C. **Mastering Front-End Web Development (HTML, Bootstrap, CSS, SEO, Cordova, SVG, ECMAScript, JavaScript, WebGL, Web Design and many more.): 14 Books in 1. Introducing 200+ Extensions. An Advanced Guide.** Amazon Digital Services LLC - KDP Print US, 2020. ISBN 9798567640135. Disponível em: <https://books.google.com.br/books?id=Y-UJEAAAQBAJ>.

PORCELLO, E.; BANKS, A. **Introdução ao GraphQL: Busca de dados com abordagem declarativa para aplicações web modernas**. Novatec Editora, 2018. ISBN 9788575227039. Disponível em: <https://books.google.com.br/books?id=yrFwDwAAQBAJ>.

SHOFNER, M. **Web Developer**. Rosen Publishing Group, 2017. (Behind the Scenes with Coders). ISBN 9781508155614. Disponível em: <https://books.google.com.br/books?id=7kBgDwAAQBAJ>.

SILVA, M. **JavaScript - Guia do Programador: Guia completo das funcionalidades de linguagem JavaScript**. Novatec Editora, 2010. ISBN 9788575222485. Disponível em: <https://books.google.com.br/books?id=BB9WDQAAQBAJ>.

SMITH, P. **Professional Website Performance: Optimizing the Front-End and Back-End**. Wiley, 2012. (PROGRAMMER TO PROGRAMMER). ISBN 9781118487525. Disponível em: <https://books.google.com.br/books?id=eYl2jmcRGSMC>.

UZAYR, S. **Mastering Git: A Beginner's Guide**. CRC Press, 2022. (Mastering Computer Science). ISBN 9781000552898. Disponível em: <https://books.google.com.br/books?id=bdteEAAAQBAJ>.

VOGEL, L.; BLEWITT, A. **Distributed Version Control with Git: Mastering the Git command line - Third Edition**. Lars Vogel, 2014. (vogella series). ISBN 9783943747126. Disponível em: <https://books.google.com.br/books?id=KIfYBQAAQBAJ>.

WEIR, L.; NEMEC, Z. **Enterprise API Management: Design and deliver valuable business APIs**. Packt Publishing, 2019. ISBN 9781787285613. Disponível em: <https://books.google.com.br/books?id=0OikDwAAQBAJ>.

APÊNDICE A – Ficha de avaliação do Estágio

FICHA DE AVALIAÇÃO DE ESTÁGIO

ERROS NO PREENCHIMENTO DESTA FICHA SERÃO DE RESPONSABILIDADE DO SUPERVISOR DE ESTÁGIO.

dentificação do Estagiário

me: BOANERGES POTYGUARA SAES JÚNIOR

so: Ciência da Computação

ricula: 201821136	**E-mail:** boanerges.junior@estudante.ufla.br

Dados da Empresa

me: VOLL SOLUCOES EM MOBILIDADE CORPORATIVA LTDA	**CNPJ:** 26.613.837/0001-08
mero do convênio:	**Modalidade:** Pessoa Jurídica de Direito Privado
nicípio / UF: Belo Horizonte / Minas Gerais	**CEP:** 30160-011
rro: Centro	
radouro: Rua da Bahia	**Nº:** 1032

Dados do Estágio

do Termo de Compromisso: 1679	**Data de aprovação:** 04/10/2021

íodo de realização do estágio: 01/10/2021 a 04/05/2022

al do estágio: Belo Horizonte

ga horária total: 858:00

Atividades Desenvolvidas

liar na elaboração de requisitos para as melhorias do setor de TI.

perar como membro de uma equipe de desenvolvimento de produtos digitais, incluindo atividades de análise de requisitos, envolvimento back-end, desenvolvimento mobile, desenvolvimento web.

liar a equipe na garantia da qualidade dos produtos desenvolvidos através de desenvolvimento de testes uais e/ou automatizados.

borar para o cumprimento da legislação, bem como as normas e políticas institucionais da Empresa, assim o o cumprimento dos padrões estabelecidos e zelando pelo patrimônio da Instituição.

peitar o sigilo das informações referentes aos processos que tenha contato.

Avaliação

pectos Considerados	**Conceitos**		
	I	**R**	**B**
ssiduidade e pontualidade nos horários			
onhecimentos demonstrado na prática das atividades			
umprimento das atividades programadas			
umprimento das normas internas da Empresa			
isposição para aprender			
ualidade do trabalho dentro de um padrão aceitável			

NIFICADOS: I - Insuficiente, **R** - Razoável, **B** - Bom

Observações do Supervisor (opicional)

Data da avaliação: ___ / ___ / _____	DocuSigned by: Marcus Costa D74F488B16C74D7... **MARCUS OLIVEIRA COSTA** **SUPERVISOR**