CAIO GOTTMANN FERNANDES MORAIS

DESENVOLVIMENTO DE SERVIÇOS WEB PARA GRUPO DE EMPRESA VAREJISTA

LAVRAS – MG

2023

CAIO GOTTMANN FERNANDES MORAIS

DESENVOLVIMENTO DE SERVIÇOS WEB PARA GRUPO DE EMPRESA VAREJISTA

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

2023

CAIO GOTTMANN FERNANDES MORAIS

DESENVOLVIMENTO DE SERVIÇOS WEB PARA GRUPO DE EMPRESA VAREJISTA

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 23 de Fevereiro de 2023.

Profa. DSc. Renata Teles Moreira	UFLA
Prof. DSc. Paulo Afonso Parreira Junior	UFLA

Prof. DSc. Maurício Ronny de Almeida Souza
Orientador

LAVRAS – MG
2023

Dedico a minha família e amigos por todo incentivo e ajuda.

AGRADECIMENTOS

Agradeço a minha mãe e avós por todo apoio e sacrifícios que fizeram, para que eu alcançasse meus objetivos, vocês foram peças fundamentais para mim.

Aos amigos que fiz no percurso, meu muito obrigado por serem a minha segunda família, porque quando a distância de casa batia, sabia que podia contar com vocês.

Ao Prof. DSc. Maurício Ronny de Almeida Souza, pela orientação durante a escrita do presente documento e todos os conselhos.

Aos docentes do curso de Ciência da Computação da Universidade Federal de Lavras, por compartilharem seus conhecimentos.

A todos aqueles que de uma maneira direta ou indireta fizeram desses anos uma etapa inesquecível.

Não importa quanto a vida possa ser ruim, sempre existe algo que você pode fazer, e triunfar.
Enquanto há vida, há esperança.
(Stephen Hawking)

RESUMO

A DTI Digital é uma empresa que atua na transformação digital e tecnológica de grandes organizações, garantindo uma parceria ideal para organizações que querem tornar-se ágeis. Nesse contexto, um grupo de empresas varejistas demandam soluções para melhorar os processos na venda de produtos, garantindo uma maior velocidade e segurança. O objetivo deste trabalho é descrever as atividades de estágio supervisionada desenvolvidas pelo autor no período de março de 2021 a fevereiro de 2022. Durante o período, o estagiário teve oportunidade de desenvolver habilidades referentes ao desenvolvimento de software nas áreas de *Front-End* e *Back-End*, garantindo uma grande abrangência em diversas áreas, times e linguagens de programação. As principais tecnologias utilizadas foram Java, React, React Native, entre outras. Como resultado foram desenvolvidas atividades de criação de uma API, site para processar os dados de planilhas e aplicativo para venda de produtos *offline*.

Palavras-chave: Relatório de estagio. Desenvolvimento de software. Front-End. Back-End.

ABSTRACT

DTI Digital is a company that operates in the digital and technological transformation of large organizations, ensuring an ideal partnership for organizations that want to become agile. In this context, a group of retail companies demand solutions to improve processes in the sale of products, guaranteeing greater speed and security. The objective of this work is to describe the supervised internship activities developed by the author from March 2021 to February 2022. During the period, the intern had the opportunity to develop skills related to software development in the Front-End and Back-End areas, allowing a wide range of areas, teams and programming languages. The main technologies used were Java, React, React Native, among others. As a result, activities were developed to create an API, a website to process data from spreadsheets and an application for selling products offline.

Keywords: Internship report. Software development. Front-End. Back-End.

LISTA DE FIGURAS

Figura 1.1 – Estrutura organizacional da DTI 11
Figura 2.1 – Diagrama de sequência de uma requisição HTTP 15
Figura 2.2 – Tecnologias mais utilizadas na Web 16
Figura 2.3 – Exemplo de componente JSX 17
Figura 2.4 – Renderização em React e React Native 18
Figura 2.5 – Funcionamento de uma API REST 19
Figura 2.6 – Exemplo de teste utilizando JUnit 21
Figura 2.7 – Eventos e artefatos do Scrum 23
Figura 3.1 – Código de teste 29
Figura 3.2 – Organização dos diretórios da API 32
Figura 3.3 – Controller 33
Figura 3.4 – Código do serviço para gravar o aceite da campanha 34
Figura 3.5 – Tabela de ciclos 36
Figura 3.6 – Estrutura do arquivo contendo a tabela dos ciclos 38
Figura 3.7 – Estrutura do arquivo contendo o histórico do ciclo 39
Figura 3.8 – Código do componente da Tab Bar 41
Figura 3.9 – Código da Tab Bar 42
Figura 3.10 – Exemplo da Tab Bar 42

LISTA DE TABELAS

Tabela 3.1 – Tabela dos meses trabalhados em cada Squad 26
Tabela 3.2 – Tabela de nomenclatura das colunas 32
Tabela 3.3 – Descrição dos diretórios da API 33

SUMÁRIO

1 Introdução 10
1.1 Sobre a DTI Digital 10
1.2 Organização do Trabalho 12
2 Conceitos e Tecnologias 13
2.1 Desenvolvimento Web 13
2.1.1 React 16
2.1.2 React Native 17
2.1.3 Spring Boot 18
2.2 Teste de Software 19
2.2.1 JUnit 20
2.3 Controle de Versão 21
2.3.1 Git 22
2.4 Metodologia Ágil 23
3 Atividades Desenvolvidas 26
3.1 Tribanco 27
3.1.1 Pix 27
3.1.2 Campanha 30
3.2 Conciliação financeira 35
3.2.1 Website 36
3.3 Martins Atacado 39
3.3.1 Aplicativo 40
3.4 Considerações Finais 42
4 Conclusão 44
REFERÊNCIAS 46

1 INTRODUÇÃO

A DTI Digital é uma empresa que atua na transformação digital e tecnológica de grandes organizações, garantindo uma parceria ideal para organizações que visam se tornar ágeis. De acordo com Riga (2021), 76,2% das empresas já implementaram ou estão desenvolvendo planos de transformação digital. A atuação da DTI visa garantir soluções específicas para cada modelo de negócio, a partir da cultura da empresa. Assim, para cada cliente é pensado e proposto uma solução que melhor se adéqua às suas necessidades. Desta forma, a DTI com seus diversos projetos e experiências no mercado, apresenta a solução e os meios para atingi-la.

O Grupo Martins é um grupo de empresas que tem como propósito o desenvolvimento do varejo brasileiro (MARTINS, 2023). O Grupo Martins possui empresas de diversas áreas voltadas ao varejo, sendo duas empresas principais: o Tribanco, que entrega soluções financeiras aos clientes, desenvolvendo e apoiando o crescimento dos varejistas (TRIBANCO, 2023); e o Martins Atacado que fornece diversos produtos em grande escala para outras empresas.

Este relatório descreve as atividades desenvolvidas pelo autor durante o estágio supervisionado realizado na empresa DTI Digital. O objetivo do estágio foi o desenvolvimento de software e soluções tecnológicas em projetos ligados ao Grupo Martins. Devido à natureza do cliente (Grupo Martins), um conglomerado de empresas, o estagiário teve a oportunidade de atuar em diversos projetos do mesmo cliente. Para cada projeto o estagiário atuou em diferentes focos e utilizou diferentes tecnologias, como: Java, JUnit, React, React Native, entre outras. O estágio foi realizado no período de março/2021 à fevereiro/2022, totalizando 1440 horas.

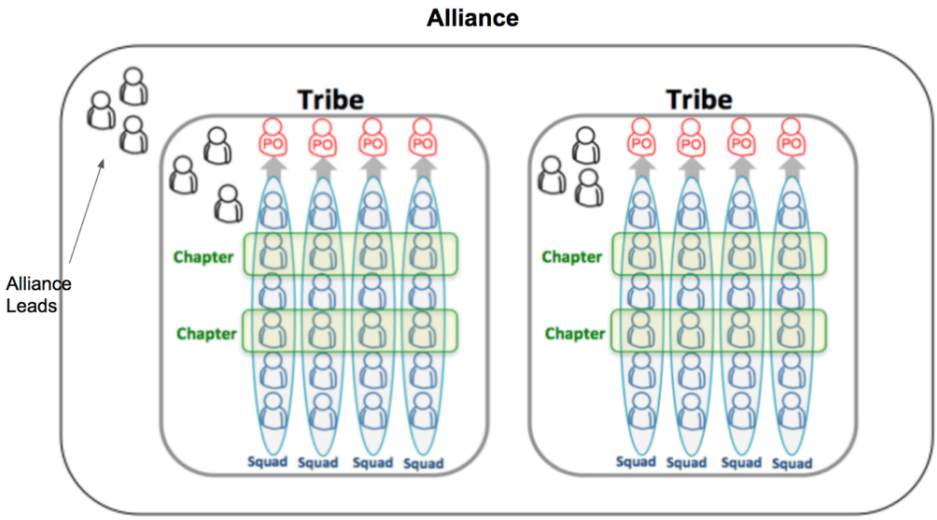
1.1 Sobre a DTI Digital

Fundada em 2009 por Marcelo Szuster e sócios, a DTI Digital está localizada na cidade de Belo Horizonte - MG e possui mais de mil colaboradores espalhados pelo Brasil. Sua estrutura física conta com dois andares em um prédio comercial e teve um escritório sediado na cidade de Lavras - MG, dando apoio aos colaboradores que ficavam em Lavras e região. A DTI é uma empresa que tem como seu público alvo empresas que necessitam de soluções tecnológicas ou melhorias em seus processos. Possui uma ampla gama de clientes, entre eles: MRV, Localiza, Vale, Martins, entre outros

(DIGITAL, 2023). Por não fornecer um produto em específico, para cada empresa é desenvolvido um sistema único voltado para as necessidades da empresa.

A Figura 1.1 demonstra a estrutura organizacional da DTI, composta por três níveis: *Squads*, Tribos e Alianças. *Squads* são a unidade mais básica de trabalho, constituindo uma equipe de desenvolvimento formada por desenvolvedores de software, designers e o PO (*Product Owner*). O PO é responsável por liderar e trazer a visão do cliente para o time de desenvolvimento. A Tribo é a junção de *Squads*, que compartilham tecnologias ou que atuam em projetos de um mesmo cliente. Dentro das Tribos é possível criar uma rede de auxílio chamado *chapter*, juntando as pessoas com habilidades semelhantes para compartilhar experiências e assim melhorar tecnicamente a Tribo. Por fim, a Aliança engloba as Tribos de diversos clientes e tecnologias, sendo uma visão mais gerencial relacionada aos projetos. Vale ressaltar que em cada nível há também os líderes responsáveis por gerenciar e dar suporte para seus subordinados.

Figura 1.1 – Estrutura organizacional da DTI



Fonte: https://www.pmtoday.co.uk/spotify-scaling-agile-model/

1.2 Organização do Trabalho

Além deste capítulo introdutório, o trabalho é organizado nos seguintes capítulos. O Capítulo 2 apresenta uma explicação dos conceitos e tecnologias utilizadas durante o desenvolvimento das atividades. O Capítulo 3 descreve de forma detalhada o desenvolvimento e atividades realizadas nos três projetos que o estagiário participou. Por fim, o Capítulo 4 apresenta a conclusão do relatório, contendo as habilidades adquiridas no estágio, as dificuldades enfrentadas e a experiência do estagiário de passar por projetos distintos.

2 CONCEITOS E TECNOLOGIAS

Este capítulo apresenta as tecnologias e conceitos acerca das atividades desenvolvidas durante o período de estágio. As seções a seguir tratam sobre o desenvolvimento Web (Seção 2.1), testes de software (Seção 2.2) e controle de versão (Seção 2.3). Estes conceitos são importantes para a execução das atividades do Capítulo 3.

2.1 Desenvolvimento Web

O desenvolvimento web é a área voltada para a elaboração de sites, aplicativos, banco de dados e ferramentas. Segundo IBM (2023), os serviços da web podem ser usados individualmente ou em conjunto com outros serviços para que diferentes aplicativos e plataformas trabalhem juntos de forma mais eficiente e coesa. Desta forma, o seu desenvolvimento tem como foco duas frentes: *Front-End* e *Back-End*. O desenvolvimento *Front-End* (cliente) concentra-se na criação de elementos visuais e interativos para o usuário, deixando o sistema mais amigável para a navegação. O desenvolvimento *Back-End* (servidor) envolve a elaboração do banco de dados e os serviços para comunicar com as telas (*Front-End*). Segundo Shah (2009) as aplicações Web são divididas em 3 camadas: apresentação, responsável por exibir as informações relacionadas ao serviço para o usuário; aplicação, camada que controla as funcionalidades e valida as regras de negócio; e a camada de dados, que consiste no banco de dados, para armazenar os dados da aplicação.

Para cada camada Web existem tecnologias que possibilitam a sua implementação. Para a camada de apresentação existem: HTML [1] (*HyperText Markup Language*) para definir a estrutura do conteúdo; CSS [2] (*Cascading Style Sheets*), representando a estilização dos componentes na tela e JavaScript [3] para programar o comportamento da tela a partir da ocorrência de um evento. Também existem *frameworks* como React [4], criados para facilitar o desenvolvimento, sendo soluções de alto nível para a reutilização de trechos de código, permitindo compartilhar funções e lógicas genéricas (SALAS-ZÁRATE et al., 2015).

[1] https://developer.mozilla.org/pt-BR/docs/Web/HTML
[2] https://developer.mozilla.org/pt-BR/docs/Web/CSS
[3] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript
[4] https://pt-br.reactjs.org

Para a camada de aplicação, por sua vez, existem tecnologias como o PHP [5], que é especialmente interessante para desenvolvimento web porque pode ser mesclada dentro do código HTML (PHP, 2023). Há também o Java que pode ser utilizado para a criação de serviços REST (*Representational State Transfer*), que é um estilo de arquitetura de software, principalmente utilizada para criar serviços web e auxiliar na integração de sistemas (LECHETA, 2015). Para a utilização das tecnologias, possuem *frameworks* que auxiliam no desenvolvimento, como o Laravel [6] que utiliza o PHP ou Spring Boot [7] que utiliza o Java.

Por fim a camada de dados possui dois tipos de tecnologias de banco de dados: banco de dados relacional, que armazena e fornece acesso a pontos de dados relacionados entre si, utilizando o SQL (*Structured Query Language*) para comunicação (ORACLE, 2023) ou não relacional com o NoSQL (*Not Only SQL*), que utilizam objetos contendo chaves associados a valores para armazenar os dados. Os bancos de dados não relacionais são otimizados especificamente para aplicativos que exigem modelos de grande volume de dados, baixa latência e flexibilidade (AWS, 2023). Os exemplos de banco de dados relacionais e não relacionais são respectivamente, Oracle database [8] e MongoDB [9].

Um dos protocolos de comunicação mais conhecidos entre as camadas é o protocolo HTTP [10] (*Hypertext Transfer Protocol*), que define um conjunto de métodos de requisição indicando uma ação a ser feita para um dado recurso. Segundo Jackson (2006), o principal recurso de um servidor Web é aceitar as solicitações HTTP dos clientes e retornar um recurso apropriado na resposta. Desta forma, ao enviar as requisições para o servidor utilizando métodos HTTP, fica claro para o servidor qual ação o requerente solicita que seja executada. Entre os métodos mais utilizados do protocolo HTTP, destacam-se:

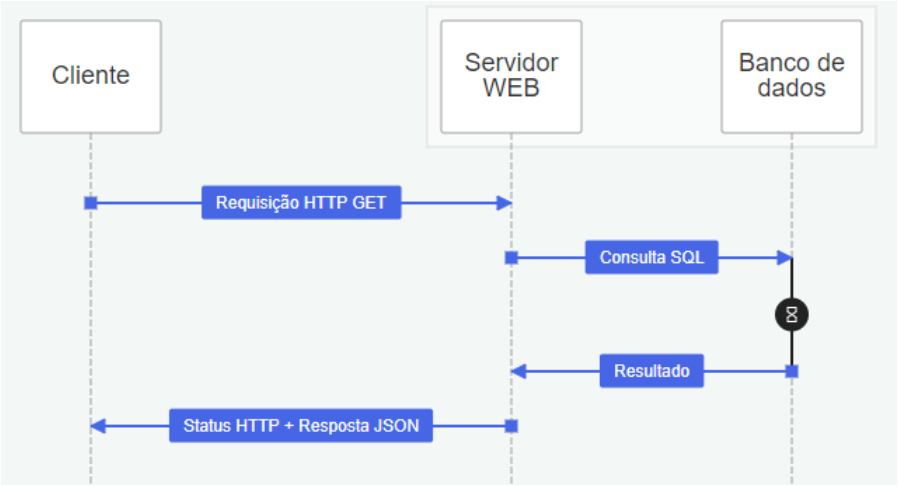
- GET: Solicita uma representação de um recurso.
- POST: Utilizado para submeter um dado a um recurso específico, causando mudanças no recurso ou servidor.

[5] https://www.php.net/manual/pt_BR/preface.php
[6] https://laravel.com
[7] https://spring.io
[8] https://www.oracle.com/database/
[9] https://www.mongodb.com
[10] https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview

- PUT: Substitui as atuais representações do recurso pela nova representação passada na requisição.
- DELETE: Remove um recurso em específico.

Para a transferência de dados dos métodos utiliza-se o JSON [11] (*JavaScript Object Notation*), uma notação de troca de dados entre os serviços (JSON, 2023). A Figura 2.1 demonstra o diagrama de sequência para uma requisição feita no servidor. Uma vez que o cliente efetua uma requisição com o método **GET**, o servidor entende que precisa fornecer os dados que estão armazenados no banco de dados para o cliente, portanto executa a consulta no banco e retorna o *status* HTTP [12] indicando se a requisição foi corretamente concluída e retorna um JSON com os dados consultados.

Figura 2.1 – Diagrama de sequência de uma requisição HTTP



Fonte: Do autor

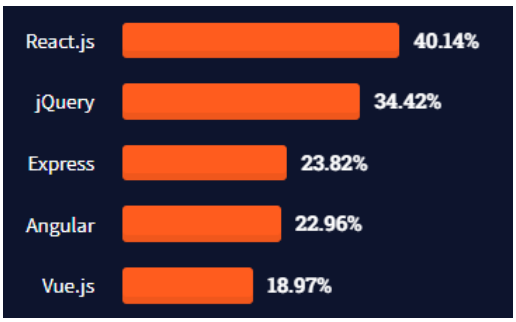
As atividades descritas neste relatório adotam os *frameworks* React para a criação de páginas web, React Native para a criação de aplicativos e Java Spring Boot para a implementação do servidor. Foram utilizadas estas tecnologias pelo sistema desenvolvido ser robusto e complexo, desta forma ao desenvolver o software pode-se aumentar a agilidade e a reutilização de código.

[11] https://www.json.org/json-en.html
[12] https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status

2.1.1 React

O React é uma biblioteca *Front-End* JavaScript para a criação de interfaces de usuários (BANKS; PORCELLO, 2017). O React foi criado em 2013 pela empresa americana Meta [13], com a proposta de oferecer código declarativo (i.e. componentes simples para cada estado na sua aplicação) e baseado em componentes (i.e. componentes encapsulados que gerenciam seu próprio estado) (META, 2023). Segundo Stackoverflow (2021), o React alcançou o primeiro lugar de tecnologia mais utilizada em desenvolvimento Web (Figura 2.2).

Figura 2.2 – Tecnologias mais utilizadas na Web



Fonte: (STACKOVERFLOW, 2021)

Uma das principais utilizações do React é em aplicações Web complexas, porque possui funções que ajudam na atualização e renderização de componentes na tela (META, 2023). Desta maneira, o React permite que somente uma parte da interface seja atualizada de forma rápida e eficiente. Por ser uma biblioteca em que os componentes são separados, sua reutilização de código e escalabilidade são altas. Assim, a criação de interfaces para o usuário é feita de forma simplificada, ajudando a garantir a melhor experiência para os usuários ao utilizar o site. Portanto, a junção destas técnicas, cria-se uma interface agradável para o usuário e seu desenvolvimento simplificado.

Para a criação de componentes é comumente utilizada a sintaxe JSX [14], uma extensão do JavaScript que adota estrutura semelhante a do HTML, preservando a robustez do JavaScript. A Figura 2.3 exemplifica uma aplicação de JSX, sendo possível verificar a utilização dos dois tipos de código. Na linha 4, o código em JavaScript é descrito dentro de chaves ({}), deixando claro para

[13] https://about.meta.com/br/
[14] https://pt-br.reactjs.org/docs/introducing-jsx.html

o React que o conteúdo dentro das chaves deve ser interpretado como um código em JavaScript. A *props* ou propriedades, é uma forma de passar conteúdo entre os componentes. Com a passagem de dados para o componente **Apresentacao**, o conteúdo é feito para que se adéque a cada página, ou seja, cada componente tem seu escopo para utilizar as propriedades. Desta forma, um componente pode ser reutilizado diversas outras vezes utilizando a mesma estrutura, variando somente as propriedades passadas a ele. Um exemplo é apresentado nas linhas 11 e 12 da Figura 2.3, sendo utilizado um mesmo componente, entretanto com o valor da *props* nome alterada.

Figura 2.3 – Exemplo de componente JSX

Fonte: Do autor

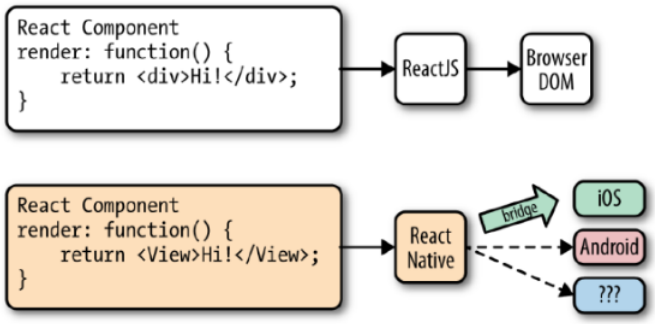
2.1.2 React Native

O React Native é uma biblioteca JavaScript que incorpora ao React (Seção 2.1.1) a conversão do JavaScript para a linguagem nativa do dispositivo. Segundo Banks e Porcello (2017), além dos benefícios do desenvolvimento em React, o React Native une os benefícios do React aos aplicativos nativos em IOS e Android. Portanto, ganha-se produtividade ao desenvolver um único código que possibilita a geração de aplicativos para as duas plataformas.

A Figura 2.4 demonstra a diferença entre o React e o React Native. Não há grandes diferenças na estrutura do componente, entretanto o React Native renderiza seu código para a linguagem nativa

do dispositivo. A compilação do React é feita no DOM [15](*Document Object Model*), sendo a representação dos objetos que compõem a estrutura e conteúdo de uma pagina Web. Já o React Native possui uma ponte (*bridge*) que invoca a renderização nativa em IOS e Android (EISENMAN, 2015). Desta forma, o código descrito em JavaScript é convertido para Swift [16] (IOS) e Kotlin [17] (Android).

Figura 2.4 – Renderização em React e React Native



Fonte: (EISENMAN, 2015)

2.1.3 Spring Boot

O Spring Boot é um *framework Back-End* em Java de código aberto, criado a partir do Spring [18] para facilitar as configurações e publicações do projeto. Segundo Azure Microsoft (2023), o Spring Boot oferece caminhos mais fáceis e rápidos para configurar e executar aplicativos, eliminando o trabalho pesado de configurar os aplicativos baseados em Spring. Portanto, sua utilização simplifica e acelera o desenvolvimento de aplicativos Web e microsserviços.

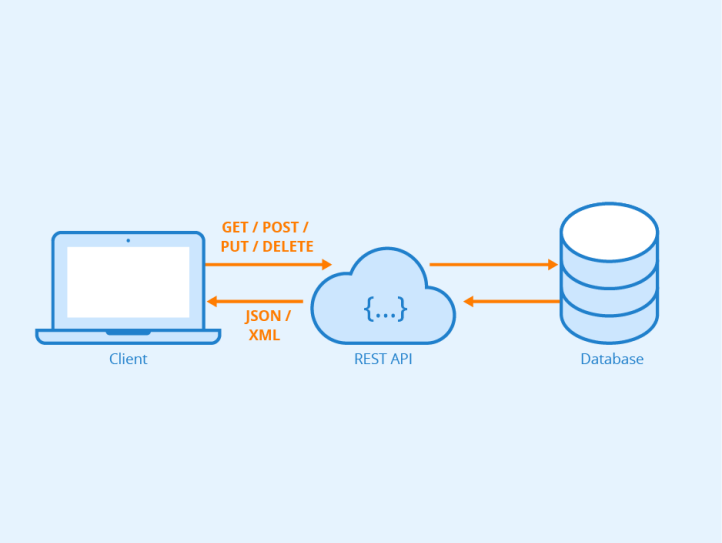
A utilização do Spring Boot está voltada para a implementação de API REST, porque já possui servidores HTTP internos, como o Tomcat [19] e Jetty [20]. Além disso, por ser uma derivação do Spring, torna-se possível a combinação de outros *frameworsk* do Spring como: Spring Data, que fornece uma

[15] https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model/Introduction
[16] https://www.swift.org/documentation/
[17] https://kotlinlang.org/docs/home.html
[18] https://spring.io
[19] https://tomcat.apache.org
[20] https://www.eclipse.org/jetty/

abordagem consistente para acesso a dados e o Spring Cloud, que fornece um conjunto de ferramentas para criar e implementar microsserviços na nuvem.

Desta forma, com a junção destas ferramentas cria-se um servidor para a comunicação com o cliente. A Figura 2.5 demonstra o cliente enviando uma solicitação para o servidor que, com os dados enviados, faz toda a validação e verificações necessárias para retornar com a devida resposta.

Figura 2.5 – Funcionamento de uma API REST



Fonte: https://www.seobility.net/en/wiki/REST_API

2.2 Teste de Software

Teste de software é o processo de execução de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado (NETO; CLAUDIO, 2007). É uma parte crítica do processo de desenvolvimento de software, porque ajuda a garantir que o software seja seguro, confiável e ofereça uma experiência positiva ao usuário.

Para que não existam erros no software ou para tentar mitigar ao máximo sua aparição, uma abordagem é descobrir o erro antes que o software seja liberado para a utilização. Desta forma, segundo Delamaro, Jino e Maldonado (2013) existe uma série de atividades, chamadas de “Validação, Verificação e Teste” (VV&T), para garantir que tanto o modo pelo qual o software está sendo cons-

truído quanto o produto em si estejam em conformidade com o especificado. Para contemplar estes aspectos, atualmente existem tecnologias que auxiliam na elaboração dos testes, como o JUnit para aplicações em Java.

2.2.1 JUnit

JUnit é um *framework* de código aberto, criado por Kent Beck e Erich Gamma para automatizar os testes em Java (TUDOSE, 2020). O JUnit permite a escrita de testes de unidade, sendo uma unidade entendida como o menor trecho de código de um sistema que pode ser testado (BERNARDO; KON, 2008). Desta forma, com sua implementação no projeto é possível criar teste para cada método de maneira separada, garantindo uma ampla cobertura de teste no software.

Segundo Beck (2004) as facilidades que o JUnit possui são:

- Executar testes automaticamente;
- Executar muitos testes ao mesmo tempo e resumir os resultados;
- Comparar os resultados reais com os resultados esperados, demonstrando as diferenças.

O JUnit fornece anotações e métodos para facilitar a implementação de testes, estão entre eles a anotação **@Test** que é utilizada para indicar o método que será testado e o método **assertEquals** responsável por verificar se o valor esperado é igual ao recebido. Desta forma, já se torna possível a implementação de diversos testes de unidade.

A Figura 2.6 demonstra um exemplo de teste, nele verifica-se que na linha 3 contém a anotação **@Test**, logo o JUnit interpretará que o método **somaTeste()** será executado como um teste. Para o teste em questão, ele está verificando se o método **somarValores()** soma corretamente os parâmetros passados. Neste caso, inicia-se o teste declarando as variáveis **valor1** e **valor2**, sendo os valores que serão somados e o **valorEsperado**, sendo o valor esperado desta operação. Na linha 9 é executado o método **somarValores()** que é atribuído a variável **valorSomado**. Desta forma, por último é utilizado o método **assertEquals** na linha 11 para verificar se o **valorEsperado** é igual ao valor **valorSomado**, assim caso os dois valores sejam iguais o teste retorna sucesso.

Figura 2.6 – Exemplo de teste utilizando JUnit

```java
class JunitExemplo {

    @Test
    void somaTeste() {
        int valor1 = 5;
        int valor2 = 5;
        int valorEsperado = 10;

        int valorSomado = calculo.somarValores(valor1, valor2)

        assertEquals(valorEsperado, valorSomado);
    }
}
```

Fonte: Do autor

No estágio foi utilizado o JUnit para a elaboração de testes de unidade, porque sua utilização no projeto ajuda a garantir a integridade do sistema, possibilitando os desenvolvedores a adicionar novos trechos de código, sem correr o risco de ter comprometido uma outra funcionalidade. Desta mesma forma, para o cliente ajuda a garantir que as regras apresentadas por ele estão sendo seguidas e testadas a cada novo desenvolvimento.

2.3 Controle de Versão

O controle de versão é o ato de acompanhar as alterações feitas em um arquivo ou um conjunto de arquivos (UMALI, 2015). Para a empresa, a utilização de controle de versão fornece um histórico do andamento do projeto e tudo que um dia já esteve no projeto. Assim, caso ocorra algum empecilho com a última versão disponibilizada do sistema, em poucos minutos é possível recuperar a versão anterior e verificar onde ocorreu o erro. Para os desenvolvedores a utilização de controle de versão favorece o trabalho individual e em grupo, porque para o indivíduo é possível traçar todas as modificações e acompanhar passo a passo cada alteração local, enquanto que para o trabalho em grupo, é possível que cada desenvolvedor trabalhe em um local do mesmo arquivo, ficando rastreada cada alteração individualmente.

Segundo Louridas (2006) existem dois tipos de sistemas de controle de versão, as centralizadas e as distribuídas. O controle de versão centralizado corresponde a um repositório central, sendo o repositório um servidor no qual todos os arquivos rastreados ficam salvos. Os controles de versão centralizados mais famosos são: CVS [21] (*Concurrent Version System*) e o Apache Subversion [22]. Para o controle de versão distribuído tem como diferença que não tem a necessidade de um servidor central. Desta forma, cada usuário possui um repositório local em seu computador. Exemplos de sistemas de controle de versão distribuída são o Git e o Mercurial [23].

2.3.1 Git

O Git é um sistema de código aberto para controle de versão criado por Linus Torvalds para ser inicialmente utilizado no desenvolvimento do *kernel* Linux. Seu desenvolvimento tinha como meta ser rápido, simples, distribuído e robusto (CHACON; STRAUB, 2014). Segundo Laster (2016) o Git permite que os usuários criem, usem e alternam entre as ramificações, com a mesma facilidade que alternam seus arquivos no seu fluxo de trabalho diário.

A principal diferença entre o Git e os outros sistemas de controle de versão segundo o Chacon e Straub (2014) é a maneira com o Git trata seus dados. Conceitualmente, a maioria dos outros sistemas armazena seus dados como uma lista de mudanças, sendo um conjunto de arquivos e as mudanças feitas ao longo do tempo. O Git armazena seus dados como uma série de *snapshots*, ou seja, como se fossem fotos do sistema. Assim, toda vez que há uma alteração nos arquivos, o Git tira uma foto de como está o sistema e salva o estado no projeto. Para deixar o processo de salvar os itens mais eficiente, se os arquivos não forem alterados, o Git não armazena o arquivo novamente, ele armazena somente a referência do arquivo salvo posteriormente. Neste relatório, foi utilizado o Git para o controle de versões de todos os projetos citados na seção 3.

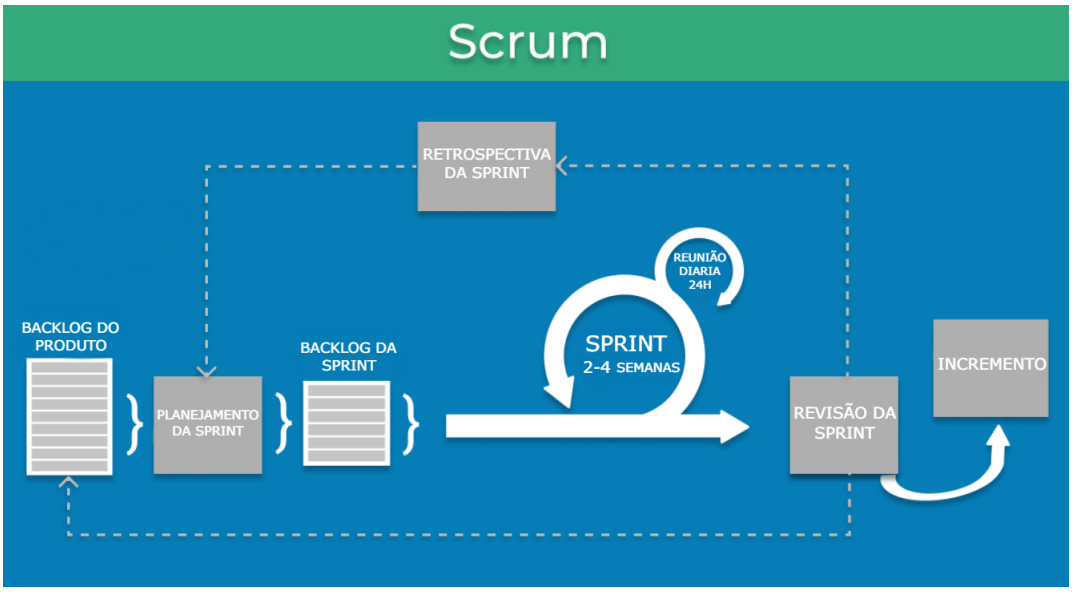
[21] https://www.gnu.org/software/trans-coord/manual/cvs/cvs.html
[22] https://subversion.apache.org
[23] https://www.mercurial-scm.org

2.4 Metodologia Ágil

As metodologias ágeis constituem uma nova classe de metodologias de desenvolvimento de software criada para atender à crescente pressão do mercado por processos mais ágeis e leves, com ciclos de desenvolvimento cada vez mais curtos (PONTES; ARTHAUD, 2018). Para a implementação dos métodos ágeis um dos *frameworks* mais famosos é o Scrum.

O Scrum é um *framework* para projetos ágeis utilizado para o gerenciamento e desenvolvimento de produtos, com a característica de ser iterativo e incremental (CRUZ, 2013). O foco do Scrum é ajudar pessoas, equipes e organizações a gerar valor por meio de soluções adaptativas para problemas complexos (SCHWABER; SUTHERLAND, 2021). O Scrum é composto por (i) papéis, (ii) eventos e (iii) artefatos. A Figura 2.7 ilustra a dinâmica de realização dos eventos e os artefatos do Scrum.

Figura 2.7 – Eventos e artefatos do Scrum



Fonte: Adaptado de (SCHWABER; SUTHERLAND, 2011)

Os papéis do Scrum são (SCHWABER; SUTHERLAND, 2021):

- Scrum Master: responsável por garantir que a metodologia do Scrum seja aplicada corretamente, ajudando todos a entenderem as teorias e práticas do Scrum;

- Product Owner: responsável por maximizar o valor do produto e do trabalho do time de desenvolvimento, sendo o único a gerenciar o *backlog* do produto;
- Time: desenvolvedores que estão comprometidos em criar qualquer aspecto do incremento a cada *sprint*.

Os eventos do Scrum são (SCHWABER; SUTHERLAND, 2021):

- Planejamento da *sprint*: tarefas a serem executadas (escopo) durante a *sprint*, sendo planejado nesta reunião por toda a equipe de desenvolvimento;
- *Sprint*: período de 2 a 4 semanas em que a equipe trabalha em conjunto para concluir um incremento;
- Reunião diária: reunião de 15 minutos para fazer com que todos os integrantes do Squad estejam atualizados com as mesmas informações, alinhados com a meta da *sprint* e planejar as próximas 24 horas;
- Revisão da *sprint*: no final da *sprint*, a equipe se reúne para uma sessão informal a fim de ver uma demonstração do incremento ou inspecioná-lo;
- Retrospectiva da *sprint*: momento em que a equipe se reúne para documentar e discutir o que funcionou e o que não funcionou na *sprint*.

Os artefatos do Scrum são (SCHWABER; SUTHERLAND, 2021):

- *Backlog* do produto: principal lista de tarefas que precisam ser feitas no projeto, gerenciada somente pelo PO;
- *Backlog* da *sprint*: lista de itens, histórias de usuários ou correções de *bugs* selecionada pela equipe de desenvolvimento para a implementação no ciclo atual da *sprint*;
- Incremento: produto final proveniente de uma *sprint*.

A implementação do Scrum nos projetos da DTI utilizam as dinâmicas da Figura 2.7 e possuem todos os papéis, eventos e artefatos demonstrados anteriormente, entretanto a utilização do SM (Scrum Master) pode não ocorrer em projetos maduros com a metodologia do Scrum, fazendo com que as responsabilidades e tarefas atribuídas ao SM fiquem para o PO. A DTI oferece também aos seus clientes a possibilidade do papel do PO ser fornecido pelo cliente, fazendo com que o cliente tenha uma maior participação no projeto. Assim, todo o gerenciamento do projeto, como as datas de entrega e as priorizações das tarefas a serem desenvolvidas fiquem diretamente com o cliente.

Portanto, segundo Pereira, Torreão e Marçal (2007) usar Scrum nos projetos ajuda a construir somente o que o cliente valoriza e não mais que isto, criando produtos melhor adaptados à realidade do cliente. As práticas do Scrum também trazem vantagens no gerenciamento do projeto, como: participação da equipe mais efetiva; os desenvolvedores sabem o que estão fazendo e porque; e maior visibilidade do desempenho da equipe.

3 ATIVIDADES DESENVOLVIDAS

Este Capítulo descreve as atividades realizadas pelo estagiário na DTI Digital. Durante o período, o estagiário participou de três projetos distintos para um mesmo cliente. O cliente era o Grupo Martins, um conglomerado de empresas que tem como propósito o desenvolvimento do varejo brasileiro (MARTINS, 2023).

A Tabela 3.1 apresenta o cronograma da participação do estagiário em cada projeto no período de Março/2021 a Fevereiro/2022. As primeiras atividades desenvolvidas pelo estagiário foram no projeto Tribanco (Seção 3.1), que consistiram no desenvolvimento de testes de unidade para garantir a confiabilidade do sistema e na implementação de uma API para cadastrar a adesão dos clientes do Martins em campanhas de marketing para o banco Tribanco.

O segundo projeto foi o Conciliação Financeira (Seção 3.2). As atividades desenvolvidas consistiram em um site para a importação de planilhas e demonstração das informações consolidadas. As planilhas eram fornecidas pelos clientes parceiros que efetuaram vendas em suas plataformas digitais, como sites e aplicativos. Esta abordagem consiste em anunciar os produtos do Martins Atacado nas empresas parceiras, ou seja, é um modelo de negócio B2B (*Business to Business*), sendo uma empresa efetuando as vendas de produtos de outra empresa em suas plataformas digitais.

O último projeto em que o estagiário participou foi no Martins Atacado (Seção 3.3). As atividades consistiram no desenvolvimento de um aplicativo para vendas de mercadorias *offline*. O aplicativo de vendas é interno do Martins, sendo utilizado por seus colaboradores para efetuar vendas para outras empresas. As seções subsequentes descrevem com detalhes as atividades realizadas em cada projeto.

Tabela 3.1 – Tabela dos meses trabalhados em cada Squad

Projeto	3	4	5	6	7	8	9	10	11	12	1	2
Tribanco	X	X	X	X								
Conciliação Financeira					X	X	X	X				
Martins Atacado									X	X	X	X

Fonte: Do autor

3.1 Tribanco

O projeto Tribanco foi criado para solucionar as demandas do banco Tribanco pertencente ao Grupo Martins. Essas demandas consistiram na sustentação de projetos existentes e no desenvolvimento de novas funcionalidades. Neste caso, existiam duas demandas distintas a serem desenvolvidas, ainda assim, no mesmo escopo do Tribanco. Por conta disto, foram criados subprojetos, entre eles o Pix e o Campanha. Para cada subprojeto derivado foi alocado um Squad para satisfazer todas as demandas atribuídas.

A primeira demanda solicitada pelo Tribanco foi na sustentação do subprojeto Pix. Em virtude do sistema estar nas fases finais, o autor ficou pouco tempo no desenvolvimento, somente um período de uma a duas semanas. O estagiário foi responsável, principalmente, pela elaboração de testes de unidade para garantir a integridade e confiabilidade do sistema.

Em um segundo momento, com o fim do subprojeto Pix, surgiu uma nova demanda do aplicativo de vendas online do Grupo Martins. Tratava-se de campanhas de marketing para determinados grupos de clientes do aplicativo. Foram levantados os dados de vendas no aplicativo e foi constatado que muitos dos clientes que efetuaram compras, não possuíam uma conta no banco do Grupo Martins. Desta forma, o Tribanco (banco do Grupo Martins) gostaria de implementar uma API para cadastrar e consultar as campanhas de marketing.

3.1.1 Pix

O subprojeto Pix era responsável pela implementação da função de pagamento instantâneo brasileiro, o Pix [1], ou seja, receber e/ou enviar os dados para o Banco Central e internalizá-los no Tribanco, garantindo a confiabilidade e a velocidade no processo. O Pix foi criado pelo Banco Central do Brasil para garantir as transferências entre contas de forma rápida e a qualquer momento do dia (BANCO CENTRAL DO BRASIL, 2022). Por tratar-se de uma forma de transferência direta e rápida era preciso que não houvesse erros no processo de internalizar os dados. Para garantir esta segurança o estagiário ficou encarregado de elaborar diversos casos de testes da API.

[1] https://www.bcb.gov.br/estabilidadefinanceira/pix

No Squad Pix haviam quatro integrantes: um Desenvolvedor Líder, um desenvolvedor pleno e dois estagiários (contando com o autor). Sendo o desenvolvedor líder uma pessoa com mais experiência para ajudar o restante da equipe, como também cuidar da comunicação com o cliente.

Com a chegada do estagiário no Squad, houve uma reunião com os desenvolvedores para a apresentação do projeto e as tecnologias adotadas. O projeto estava em fase de teste e validação, desta forma, as atividades atribuídas ao estagiário foram a elaboração de testes de unidade.

Em seus primeiros dias, o estagiário ficou acompanhando o desenvolvedor líder para entender o funcionamento do projeto e as práticas adotadas, utilizando-se da programação em par. A programação em par (*Pair programming*), consiste em dois programadores trabalharem juntos, sendo um deles o desenvolvedor líder da sessão, ficando com a parte de criação do código e o outro desenvolvedor para analisar e questionar o trabalho do líder (VALENTE, 2020). Neste contexto, o estagiário começou atuando no segundo papel, para adquirir conhecimento sobre o código e experiência para que em seguida houvesse a inversão de papéis na elaboração dos testes.

No projeto foi utilizado Java Spring Boot no desenvolvimento da API para a comunicação nas diversas áreas do banco, juntamente com o JUnit para a elaboração de testes automatizados. O JUnit é um *framework* de código aberto para a automatização de testes em Java, facilitando a geração e manutenção dos testes. Sua implementação no projeto ajuda a testar os métodos de forma isolada, garantindo que a lógica aplicada pelo desenvolvedor esteja conforme o esperado. Juntamente com o JUnit foi utilizado o *framework* Mockito [2] para a criação de objetos *mock*, simulando o comportamento de objetos reais, entretanto de forma controlada. Assim, o dado retornado dos métodos serão aqueles definidos pelo desenvolvedor e caso algum método transforme os dados de forma incorreta, sabe-se que a lógica do método em questão está incorreta.

A Figura 3.1 demonstra o resultado de uma atividade executada pelo estagiário: uma aplicação de teste para verificar se uma conta corrente do banco está ativa. No teste é avaliado se a lógica adotada para recuperar as contas correntes estão corretas. O JUnit e o Mockito utilizam anotações (*annotations*) para facilitar a escrita dos testes, como as palavras contendo um arroba (@) antes. Diante disso, para o exemplo da Figura 3.1 as anotações do JUnit são **@Test**, indicando qual o método que

[2] https://site.mockito.org

executa o teste e o **@Before** para demonstrar o método que será executado antes dos testes começarem. O Mockito também utiliza anotações como **@Mock** e o **@InjectMocks**, sendo eles respectivamente: designar que a classe terá um *mock* dos dados; e para designar que neste arquivo possuirá algum método *mock* injetado dentro da classe.

Figura 3.1 – Código de teste

```
1  public class ContaServiceTest {
2
3      @InjectMocks
4      private ContaCorrenteService contaCorrenteService;
5
6      @Mock
7      private ContaCorrenteRepository contaCorrenteRepository;
8
9      @Before
10     public void initMocks() {
11         MockitoAnnotations.initMocks(this);
12     }
13
14     @Test
15     public void consultaContaCorrenteSucesso() {
16         Long tipoConta = 123L;
17         String descricao = "Teste";
18         String categoria = "CC";
19
20         DetalhesContaDTO contaDto = new DetalhesContaDTO();
21         contaDto.setTipoConta(tipoConta);
22         contaDto.setDescricao(descricao);
23         contaDto.setCategoria(categoria);
24
25         Mockito.when(contaCorrenteRepository.consultaConta(anyLong()))
26                 .thenReturn(contaDto);
27
28         DetalhesContaDTO contaEncontrada = contaCorrenteService.consultaConta(123L);
29
30         assertNotNull(contaEncontrada);
31         assertEquals(tipoConta, contaEncontrada.getTipoConta());
32         assertEquals(descricao, contaEncontrada.getDescricao());
33         assertEquals(categoria, contaEncontrada.getCategoria());
34     }
35 }
```

Fonte: Do autor

Primeiramente são declaradas as classes que farão parte do teste, junto com as anotações. Para o teste ocorrer sem problemas, utiliza-se o **@Before** para inicializar o **mockitoAnotacions** antes dos testes iniciarem, desta forma é possível utilizar as anotações do Mockito nas classes. Para a elaboração do teste, na linha 20 até a 23 é criado um *mock* da estrutura de dados **DetalhesContaDTO** com os dados da conta, desta forma ela será o resultado esperado na consulta. Com o *mock* dos dados da conta, na linha 25 utiliza-se o método **Mockito.when()**, para que quando o método **contaCorrenteRe-**

pository.consultaConta() for executado dentro do método **contaCorrenteService.consultaConta()**, retornar o objeto *mock* declarado dentro do método **.thenReturn()**.

Após o *mock* do retorno do método, é executada a linha 28 com o método **contaCorrenteService.consultaConta()** que terá sua implementação testada. Nele é feito o processamento e as tarefas necessárias para a obtenção da conta e retorná-la. Por fim é feita a validação dos dados retornados e se eles estão de acordo com os dados *mocks* criados anteriormente. Para isto utiliza-se o método **assertNotNull()**, para verificar se o objeto não é nulo e o método **assertEquals()**, para validar se os dois dados passados são iguais. Desta forma, verifica se os dados da conta que foi buscada pelo método são iguais ao criado na **contaDto**. Portanto, caso os dados do *mock* estejam iguais ao retornado do método, o teste será aprovado. Entretanto, caso algum dado não esteja de acordo, o teste falhará e será necessário uma análise do código, porque a lógica aplicada no método pode conter algum erro.

O estagiário desenvolveu ao todo seis casos de teste durante o período que estava no projeto. Um dos desafios enfrentados na elaboração dos testes de unidade é garantir a cobertura dos testes, ou seja, garantir que todas as combinações possíveis de entradas e condições no processamento sejam contempladas. Porque trata-se de um sistema que é muito importante para o banco e com dados sensíveis, desta forma não pode ocorrer erros no processamento do Pix, podendo causar danos monetários para o Tribanco.

3.1.2 Campanha

O subprojeto Campanha foi uma demanda vinda do aplicativo de vendas online do Martins Atacado. Tratava-se de campanhas de marketing para a fidelização dos clientes do aplicativo no Tribanco. As abordagens utilizadas foram campanhas para estimular a utilização dos programas do banco e atrair novos clientes. Como exemplo a campanha: “Solicite o cartão empresarial e garanta crédito rápido para abastecer sua loja”. Buscando assim, atrair os clientes a utilizarem o cartão empresarial do Tribanco.

Para suprir a demanda, um Squad ficou encarregado de criar uma API REST para tratar as campanhas de marketing do Tribanco. O Squad era composto por dois integrantes: um desenvolvedor

pleno e um estagiário (autor). Para o desenvolvimento foi feito uma reunião com o cliente e ficou acordado que seria utilizado Java Spring Boot para a API REST, Oracle DB [3] para o banco de dados.

No início do desenvolvimento foi feita uma reunião para a análise de requisitos, entre o Squad Campanha, integrantes responsáveis do projeto por parte interna do Tribanco e também o time responsável pelo aplicativo de vendas online. Nesta reunião foram discutidas as melhores abordagens para o fluxo do usuário no aplicativo, deixando claro as campanhas para o cliente. Por fim, foi definido a criação de um *banner* no aplicativo logo após o *login* no sistema, nele seriam reveladas as campanhas que o cliente participa.

A primeira atividade foi a elaboração dos dados da campanha, ou seja, a estrutura de como seria a campanha no banco de dados. Foi disponibilizado um documento contendo todas as padronizações e como utilizar na criação do banco de dados. Por tratar-se de um desenvolvimento integrado com o banco de dados do Tribanco, há uma padronização das nomenclaturas dos dados e um responsável do Tribanco pelo banco de dados. Desta forma, ao finalizar a criação dos dados, foi repassado para o administrador de banco de dados do Tribanco para criar as estruturas do banco de dados.

Para a criação dos dados seguindo as padronizações estabelecidas, os primeiros passos detalham a composição do nome das tabelas e para a criação foi utilizado o prefixo **TBL_** e subsequente o nome descritivo da tabela no singular e com as letras maiúsculas. Como exemplo, a tabela de pessoas **TBL_PESSOA** contendo os dados vinculados a uma pessoa. A nomenclatura das colunas dentro da tabela também possuía seus padrões. As colunas seguem um prefixo que varia dependendo do tipo de dado que será armazenado, como determinado na Tabela 3.2 e subsequente com a descrição do significado da coluna, com cada palavra separada por *underscore* ("_"), no singular e com as letras maiúsculas.

Com a estrutura dos dados elaborada, a próxima tarefa foi a elaboração de uma API REST em Java Spring Boot que retornasse os dados necessários para apresentar a campanha e adicionar a adesão do cliente no banco de dados. A DTI não possui um padrão na organização dos diretórios, porque entende-se que cada projeto tem suas peculiaridades, deixando assim a critério dos desenvolvedores que estão atuando escolher o melhor padrão e organização de projeto. Com isto, para garantir um

[3] https://docs.oracle.com/en/database/oracle/oracle-database/

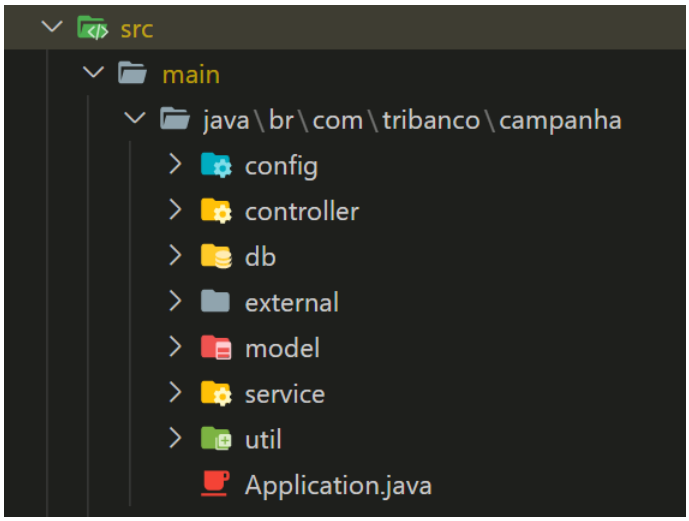
Tabela 3.2 – Tabela de nomenclatura das colunas

Identificação	Tipo da coluna	Característica	Exemplo
CD	Código	Identifica o elemento.	CD_PESSOA
NM	Nome		NM_CLIENTE
VL	Valor		VL_COMPRA
DS	Descrição		DS_SERVICO
FL	Flag	Dados booleanos (Ex.: 0 ou 1).	FL_ATIVO
TP	Tipo	Valores fixos (Ex.: P = pequeno).	TP_LOTE
NR	Número		NR_CPF
DT	Data		DT_ALTERACAO
ID	Identificador	Nunca será dividido (Ex.: username)	ID_REDE
PC	Percentual		PC_DESCONTO
QT	Quantidade		QT_MAXIMA

Fonte: Do autor

fácil entendimento e manutenção, utilizou-se como base o primeiro conceito do princípio SOLID – *Single Responsibility Principle* (Princípio da Responsabilidade única) para a base de criação dos diretórios e arquivos. A Figura 3.2 ilustra os diretórios e a Tabela 3.3 descreve a organização do projeto, respectivamente.

Figura 3.2 – Organização dos diretórios da API



Fonte: Do autor

Após a criação do banco de dados, o próximo passo foi a elaboração da API com a rota **/adesao** com o método HTTP POST para o cliente aderir a campanha, sendo comumente utilizado no projeto

Tabela 3.3 – Descrição dos diretórios da API

Diretório	Descrição
`config`	Contêm todos os arquivos de configuração do projeto
`controller`	Trata do protocolo de comunicação, definindo o comportamento da aplicação a determinadas requisições HTTP
`db`	Mapeamento das estruturas e repositórios para comunicação do banco de dados
`external`	Implementa a consulta de sistemas externos ao projeto
`model`	Intercepta o acesso às regras de negócio referente ao banco de dados e requisições
`service`	Implementa a regra de negócio
`util`	Métodos e funções utilitárias para a aplicação

Fonte: Do autor

a expressão “aceite” da campanha. Desta forma, quando o cliente clicar no banner do aplicativo e aceitar os termos, o aplicativo envia uma requisição para a API com os dados referente ao cliente e a campanha. Após o recebimento e validação dos dados, o cliente adere a campanha, gravando no banco de dados os dados informados. Assim, a partir do aceite da campanha, outras áreas do Tribanco ficam encarregadas de prosseguir com a efetivação da campanha.

A Figura 3.3 apresenta o *controller* com o método **gravarAceite()** que será executado ao chamar a rota de adesão da campanha. O *controller* contém as anotações **@PostMapping** (linha 1) e **@RequestBody** (linha 2), sendo respectivamente para definir que a rota **/adesao** é um POST e que os dados no corpo da requisição sejam do tipo **AceiteRequestModel**, ou seja, os dados referentes ao cliente e a campanha. O retorno é definido a partir do **ResponseEntity** uma classe que o Spring fornece para manipular os dados HTTP de retorno. Portanto na linha 2, o retorno do método será uma **ResponseEntity** do tipo **AceiteCampanhaEntity**, contendo os dados do cliente e da campanha salvos no banco de dados. Em seguida na linha 3, define-se qual o status que será retornado, a partir do **ResponseEntity.status()** e o **ResponseEntity.body()** para o corpo da resposta.

Figura 3.3 – Controller

```
1  @PostMapping(path = "/adesao")
2  public ResponseEntity<AceiteCampanhaEntity> gravarAceite(@RequestBody AceiteRequestModel aceiteRequestModel){
3          return ResponseEntity.status(HttpStatus.CREATED).body(aceiteService.gravarAceite(aceiteRequestModel));
4  }
```

Fonte: Do autor

Após a execução e caso tudo esteja correto, o retorno será os dados do aceite com o status HTTP 201 [4] (*Created*), indicando que a adesão do cliente foi internalizado com sucesso no banco de dados. Entretanto, caso seja lançada qualquer tipo de exceção é retornada a mensagem de erro com o status HTTP de erro.

Outra atividade realizada foi o desenvolvimento do serviço **gravarAceite()** (ilustrado na Figura 3.4) sendo o local que valida toda a regra de negócio, para este caso é feita a validação do CNPJ e se o cliente participa da campanha. A primeira validação é a do CNPJ, porque ela é utilizada para localizar o cliente na base de dados. Foi utilizado o método **ValidacaoRequest.validaCnpj()** para a verificação se os dígitos verificadores estão corretos para o CNPJ em questão.

Figura 3.4 – Código do serviço para gravar o aceite da campanha

```
1  public AceiteCampanhaEntity gravarAceite(AceiteRequestModel aceite) {
2    if (ValidacaoRequest.validaCnpj(aceite.getCnpj())) {
3      try {
4        Optional<CampanhaClienteEntity> campanhaCliente =
5          campanhaClienteRepository.findByDsCnpjAndCdCampanhaAndFlStatusAtivo(
6            aceite.getCnpj(),
7            aceite.getCdCampanha(),
8            1);
9
10       if (campanhaCliente.isPresent()) {
11         return aceiteCampanhaRepository.save(buildOfertaCampanhaEntity(aceite));
12       }
13
14     } catch (Exception e) {
15       throw new InfrastructureException("Falha ao gravar aceite no banco de dados.");
16     }
17
18     throw new BusinessException("Falha ao gravar dados no banco. O usuario
19       não esta habilitado para esta campanha");
20   } else {
21     throw new BusinessException("CNPJ: Formato inválido");
22   }
23 }
```

Fonte: Do autor

Após o CNPJ ser validado, significa que a empresa existe, entretanto é preciso verificar se ela faz parte da campanha. Para esta verificação é feita uma consulta na tabela de relacionamento

[4] https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status/201

entre cliente e campanha, demonstrada na Figura 3.4 da linha 4 até a 8, a partir do CNPJ, código da campanha e o número um (verdadeiro em *booleano*) para informar que a campanha precisa estar ativa. Com o resultado da consulta é feita uma segunda validação na linha 10, agora para o registro retornado, caso não tenha resultado. Isto pode significar que não foi encontrado o cliente ou campanha, ou que a campanha escolhida não está mais ativa para o cliente, logo, ele não está apto a prosseguir com o processamento.

Desta forma, caso o CNPJ não seja válido ou não seja possível encontrar a campanha para o cliente é retornado uma exceção **BusinessException** com o status HTTP 422 [5] (*Unprocessable Entity*) e o texto referente ao erro (linha 18 e linha 21). Isto indica que o conteúdo da requisição está correto, mas não foi possível processar os dados. Para o caso de qualquer outro problema, é retornada a exceção **InfrastructureException** com o status HTTP 500 [6] (*Internal Server Error*) junto com o texto do erro (linha 15), indicando que um erro inesperado foi encontrado e não é possível continuar com o processamento.

Quando é encontrada a campanha para o cliente, a linha 11 é executada e adiciona no banco de dados a campanha e o cliente, finalizando assim o aceite da campanha, retornando para o aplicativo os dados do aceite que foram salvos no banco de dados.

3.2 Conciliação financeira

A Conciliação Financeira é um projeto voltado no processamento e internalização de dados de vendas provenientes do *marketplace* de clientes parceiros do Martins. Uma análise interna verificou que todos os dados fornecidos pelas empresas parceiras eram enviados por planilhas e processadas manualmente por um colaborador. Desta forma, o cliente necessitava de uma solução tecnológica para automatizar a internalização dos dados e visualizar todos os ciclos já processados (planilhas internalizadas) no sistema. O projeto consiste em uma aplicação *WEB* (*Front-End* e *Back-End*) para importação de planilhas e todo seu processamento.

[5] https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status/422
[6] https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status/500

A partir das demandas do cliente foi criado o Squad Conciliação Financeira para suprir todas as exigências do sistema. O Squad alocado era composto por cinco desenvolvedores: um Desenvolvedor Líder; um desenvolvedor pleno *Front-End*, um Desenvolvedor pleno *Back-End* e dois estagiários (contando com o autor).

A tecnologia adotada no *Front-End* foi React, Redux [7] para o gerenciamento dos estados, TypeScript [8] para adicionar tipagem no desenvolvimento e para os componentes o Material UI [9], uma biblioteca de componentes. O *Back-End* foi implementado em ASP.NET Web API [10] para tratar as requisições feitas no site. No projeto o estagiário optou por ser desenvolvedor *Front-End*.

3.2.1 Website

Com a chegada do estagiário no projeto, as primeiras atividades foram estudar as ferramentas utilizadas e acompanhar as reuniões do Squad. Desta forma, o autor poderia estudar a documentação do React enquanto adequava-se ao novo projeto. Após estudar as tecnologias utilizadas no projeto, uma tarefa de desenvolvimento foi atribuída para o estagiário. A Figura 3.5 demonstra a tabela desenvolvida pelo estagiário. Nesta atividade, foi repassada a criação de uma tabela para visualizar o histórico dos ciclos das empresas.

Figura 3.5 – Tabela de ciclos

Id. do ciclo	Data	Valor total do ciclo	Valor total de repasse	Etapa	Status
1	01/07/2022	R$ 45.819,32	R$ 7.560,62	--	Finalizada
2	01/08/2022	R$ 15.449,73	R$ 6.466,64	--	Finalizada
3	01/09/2022	R$ 77.381,22	R$ 8.861,84	Validação	Processamento

Fonte: Do autor

Para a implementação da tabela foi utilizado o componente **Table** do Material UI. Por tratar-se de um componente que em outras partes do projeto já foram utilizadas, sua implementação foi

[7] https://redux.js.org/introduction/getting-started
[8] https://www.typescriptlang.org
[9] https://mui.com/pt/
[10] https://learn.microsoft.com/pt-br/aspnet/web-api/

separada em duas partes: cabeçalho e o conteúdo. Sendo o cabeçalho já elaborado em um componente separado e o conteúdo da tabela ainda a ser desenvolvido.

A Figura 3.6 apresenta o código para a elaboração da tabela de ciclos. Para a atribuição dos valores no cabeçalho, na linha 7 utilizou-se um vetor de objetos, com os seguintes dados:

- *id*, um identificador único do conteúdo;
- *numeric*, uma variável *booleana* para identificar se o tipo de dado é numérico;
- *disablePadding*, uma variável *booleana* para identificar se é preciso aplicar um espaçamento entre os dados e
- *label*, o valor exibido no cabeçalho.

Este vetor é passado por *props* na linha 22 com o nome de **headCells** para o componente de cabeçalho **<EnhancedTableHead />**.

Para a elaboração da tela é preciso dos dados sobre os ciclos executados. Para isto o componente **TabelaCiclos** recebe um vetor de ciclos por *props*, que na linha 17 é desestruturado para variável **ciclos**. Para adicionar o conteúdo da tabela, precisa criar dentro da *tag* **<TableBody>**. Desta forma, indica para o componente **<Table>** que os dados a seguir farão parte do corpo da tabela.

Para a criação do conteúdo foi elaborado um componente chamado **<Historico />**, que recebe os dados de um ciclo. Por tratar-se de diversos ciclos a renderização do componente é feita na linha 24, a partir de um *map* [11] do vetor de ciclos. O *map* consiste em passar por cada item do vetor e retornar um novo vetor, neste exemplo, para cada item do **ciclos** é retornado um componente. Assim, para cada ciclo é renderizado na tela um componente do histórico, formando as linhas da tabela (ilustrada na figura 3.5).

O componente de histórico ilustrado na Figura 3.7 utiliza as *tags* do Material UI, entre elas a **TableRow** e **TableCell**, definindo respectivamente a linha da tabela e o conteúdo para cada coluna. Para a exibição da data é utilizado o método **formateDate()** na linha 9. Nele é feito a transformação necessária para que a apresentação da data fique condizente com as data do padrão brasileiro, como

[11] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Figura 3.6 – Estrutura do arquivo contendo a tabela dos ciclos

```
1  import { Box, Table, TableBody } from '@material-ui/core'
2  import EnhancedTableHead from '../../../components/TableHeader'
3  import { HeadCell } from '../../../interfaces/interfaces'
4  import { Ciclo } from '../../../types/ciclosApiType'
5  import Historico from './Historico'
6
7  const headCellContent: HeadCell[] = [
8    { id: 'cicloId', numeric: true, disablePadding: true, label: 'Id. do ciclo' },
9    { id: 'dataCadastro', numeric: true, disablePadding: false, label: 'Data' },
10   { id: 'valorTotalCiclo', numeric: true, disablePadding: false, label: 'Valor total do ciclo' },
11   { id: 'valorTotalRepasse', numeric: true, disablePadding: false, label: 'Valor total de repasse' },
12   { id: 'etapa', numeric: true, disablePadding: false, label: 'Etapa' },
13   { id: 'observacao', numeric: true, disablePadding: false, label: 'Status' }
14 ]
15
16 const TabelaCiclos = (props: { ciclos: Ciclo[] }) => {
17   const { ciclos } = props
18
19   return (
20     <Box>
21       <Table>
22         <EnhancedTableHead headCells={headCellContent} />
23         <TableBody>
24           {ciclos.map((ciclo) => {
25             return <Historico ciclo={ciclo}  />
26           })}
27         </TableBody>
28       </Table>
29     </Box>
30   )
31 }
32
33 export default TabelaCiclos
```

Fonte: Do autor

por exemplo: DD/MM/YYYY (DD: dia com dois dígitos; MM: mês com dois dígitos e YYYY: ano com quatro dígitos). Desta mesma forma o método **formateCurrency()** na linha 10 e 11, transformam um tipo de dado inteiro ou real em um valor em reais (R$), como por exemplo o valor real 2999.99 é transformado em R$ 2.999,99.

Alguns dados que são fornecidos no ciclo tem a possibilidade de não existir ou ser um valor zerado. Com isto, utilizamos o operador ternário [12] para demonstrar determinado dado a partir de uma condição. Assim é garantido que para os dados indefinidos (*undefined*) ou contendo o valor zero (para *booleano* zero é considerado falso) é adicionado dois traços, caso contrário é apresentado os valores da variável com suas possíveis transformações. A estrutura de um operador ternário é da seguinte forma:

[12] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Conditional_Operator

Figura 3.7 – Estrutura do arquivo contendo o histórico do ciclo

```
1  import { TableCell, TableRow } from '@material-ui/core'
2  import { Ciclo } from '../../../types/ciclosApiType'
3  import { formateCurrency, formateDate } from '../../../utils/stringUtils'
4
5  const Historico = ({ ciclo }: { ciclo: Ciclo }) => {
6    return (
7      <TableRow>
8        <TableCell>{ciclo.cicloId}</TableCell>
9        <TableCell>{formateDate(ciclo.dataCadastro)}</TableCell>
10       <TableCell>{ciclo.valorTotalCiclo ? formateCurrency(ciclo.valorTotalCiclo) : '--'}</TableCell>
11       <TableCell>{ciclo.valorTotalRepasse ? formateCurrency(ciclo.valorTotalRepasse) : '--'}</TableCell>
12       <TableCell>{ciclo.etapa ? ciclo.etapa : '--'}</TableCell>
13       <TableCell>{ciclo.observacao ? ciclo.observacao : '--'}</TableCell>
14     </TableRow>
15   )
16 }
17
18 export default Historico
```

Fonte: Do autor

condição ? verdadeiro : falso

Portanto, dependendo da condição apresentada no primeiro parâmetro, será executado um local do ternário. Assim como foi utilizado na linha 13, caso a observação exista e contenha um texto, a condição é verdadeira, logo apresentará o valor da variável. Caso contrário, a variável de observação não exista, ou seja, indefinida, a condição para ela será falsa, desta forma os dois traços serão exibidos.

3.3 Martins Atacado

O projeto Martins Atacado é uma demanda do Grupo Martins, na venda de mercadorias em atacado para grandes comércios e lojas locais. Um dos requisitos primordiais para a elaboração do aplicativo seria a funcionalidade de vendas *offline*, porque em muitos locais do Brasil, não possuem uma internet de qualidade. Com o aplicativo, há garantia que a venda seja feita independente da conexão móvel ou *wifi* no local da venda. Toda a busca e validação dos itens vendidos deve ser feita no próprio aplicativo, divergindo dos comércios eletrônicos que geralmente utilizam API's para este tipo de validação. Além disso, desejava-se que o aplicativo fosse multiplataforma, ou seja, disponível para dispositivos *Android* e *IOS*.

A elaboração do Squad App *Offline* era composto por quatro integrantes: um desenvolvedor Líder; um desenvolvedor pleno e dois estagiários (contando com o autor). Por parte do Martins foi

disposto um PO e um desenvolvedor sênior. Os funcionários do Martins eram do time que atuavam no desenvolvimento do aplicativo antigo, e participaram do novo projeto para fornecer o conhecimento e experiência adquiridas na versão anterior.

A tecnologia adotada na criação do aplicativo foi o React Native, Typescript para a tipagem, SQLite [13] para o banco de dados local e para o gerenciamento de estado o MobX [14]. Todo o *Back-End* para alimentar a tabela local, é feita pelo time de desenvolvimento do Martins, desta forma o projeto ficou focado na elaboração do aplicativo.

3.3.1 Aplicativo

O primeiro passo dentro do Squad foi acompanhar os outros desenvolvedores para entender o código e os métodos adotados no projeto. Assim, o estagiário pode estudar a documentação do React Native enquanto adequava-se ao novo projeto. A atividade repassada ao estagiário foi a pesquisa de uma nova *Tab Bar*, porque a que já estava implementada continha muitas falhas.

A *Tab Bar* é a navegação do aplicativo por meio de abas, podendo ser utilizada na parte inferior ou superior, neste caso foi utilizado o superior. Com sua utilização, fica mais claro o contexto em que a página está inserida. Como ilustrada na Figura 3.10, nota-se que o pedido é o contexto geral, ademais temos três abas para navegar no pedido. A forma de trocar de página também é favorecida, porque caso arraste a tela para o lado ou clicar na aba, muda-se o conteúdo da página, porém ainda mantém o mesmo contexto.

Ao final da pesquisa foi escolhida a *Tab Navigation* [15] do *React Navigation*, sendo a que mais atende a demanda e sem apresentar falhas em sua composição. Foi construído o componente **<TabBar />** (Figura 3.8), para que seja possível a reutilização em diversas partes do código.

A elaboração do componente, ilustrado na Figura 3.8 passa primeiramente na inicialização do método **createMaterialTopTabNavigator()** na linha 1, garantindo que os componentes do *Tab Navigation* sejam instanciados. Assim, os componentes **<Tab.Navigator>** que possibilita a transição entre as telas e **<Tab.Screen>** para configurar qual componente será renderizado, poderão ser utilizados.

[13] https://www.sqlite.org/index.html
[14] https://mobx.js.org/README.html
[15] https://reactnavigation.org/docs/tab-based-navigation/

Figura 3.8 – Código do componente da Tab Bar

```
const Tab = createMaterialTopTabNavigator();
const TabBar = ({ tabs }: Props) => (
  <NavigationContainer independent={true}>
    <Tab.Navigator>
      {tabs.map((tab) => (
        <Tab.Screen
          key={tab.tabName}
          name={tab.tabName}
          options={{ tabBarLabel: tab.tabName }}
          children={tab.children}
        />
      ))}
    </Tab.Navigator>
  </NavigationContainer>
);

export default TabBar;
```

Fonte: Do autor

Para o conteúdo da aba de navegação, o componente recebe uma *props* chamada **tabs**, contendo um vetor com os dados das abas. Para cada posição do vetor temos os seguintes dados: A **tabName** contendo o nome da página e o **children**, um método que retorna um componente. Desta forma, o componente recebe o nome que será exibido na aba e o componente que será renderizado na tela.

Como a navegação pode ter diversas abas, utiliza-se o *map*, para que cada aba seja renderizada. Seguindo os critérios do **Tab.Screen** para a criação das abas, foram utilizadas as seguintes propriedades: *key* um valor único para aba, neste caso o próprio nome, deixando claro que não pode haver duas abas com o mesmo nome; *name* para o nome da aba; *tabBarLabel* dentro do *option* para adicionar o valor que será exibido na aba e o *children* o componente que será renderizado na tela.

A Figura 3.9 apresenta o código referente a chamada do componente **TabBar** ilustrado na Figura 3.10, contendo a *props* **tabs** com o **tabName** e **children**. Portanto, sempre que for necessário a criação das abas, basta utilizar o **<TabBar>** com a *props* **tabs**.

Figura 3.9 – Código da Tab Bar

```
<TabBar
  tabs={[
    {
      tabName: "CABEÇALHO",
      children: () => <Cabecalho />,
    },
    {
      tabName: "ITENS",
      children: () => <Itens />,
    },
    {
      tabName: "CAMINHÃO",
      children: () => <Caminhao />,
    },
  ]}
/>
```

Fonte: Do autor

Figura 3.10 – Exemplo da Tab Bar

Fonte: Do autor

3.4 Considerações Finais

Este capítulo apresentou as atividades desenvolvidas em cada projeto que o estagiário participou. A trajetória do estagiário foi no desenvolvimento de testes de unidade, passando pelo desenvolvimento de site, chegando até a elaboração de aplicativos móveis. O estagiário adquiriu em cada projeto o conhecimento de técnicas e metodologias de trabalho, que são amplamente utilizadas no mercado de trabalho.

Um dos principais desafios para o estagiário foi a troca constante de tecnologias, pois a cada quatro meses as tecnologias e projetos foram alterados. Assim, a cada período era preciso deixar o que foi aprendido de lado e buscar aprender as novas tecnologias que eram implementadas no projeto. Entretanto, esta troca constante também garantiu um grande aprendizado em diversas áreas, deste modo o estagiário pode verificar quais tecnologias gostaria de continuar para sua caminhada no mercado de trabalho.

Por fim, com o conhecimento que foi adquirido nos projetos juntamente com o conhecimento adquirido no curso de Ciência da Computação, pode-se dizer que o estagiário teve um grande avanço profissionalmente, desenvolvendo em seu currículo diversas tecnologias e experiências no desenvolvimento web. Desta forma, todos os projetos que o estagiário participou contribuíram muito para a formação profissional e acadêmica.

4 CONCLUSÃO

Este relatório de estágio realizado na DTI Digital, descreveu as atividades desenvolvidas em diversos projetos durante o período do estágio. As atividades realizadas consistiram no desenvolvimento de aplicações web para empresas do ramo varejista do Brasil, passando por diversas tecnologias e aplicações, possibilitando assim uma vasta experiência no desenvolvimento de softwares para grandes empresas.

A atuação no projeto "Pix", na concepção de testes unitários, permitiu um grande aprendizado na elaboração de testes e no pensamento de diversos caminhos para verificar possíveis inconsistências no código. Além disso, permitiu a prática do uso de uma ferramenta de teste (JUnit). A atuação no projeto "Campanha" no desenvolvimento de servidores *Back-End* utilizando o Spring Boot, possibilitou o aprendizado na criação de APIs e na comunicação entre o cliente e servidor utilizando os métodos HTTP. Os projetos "Conciliação Financeira" e "Martins Atacado" viabilizaram a experiência de utilização de *frameworks* para elaborar sites e aplicativos, e fomentaram o desenvolvimento de habilidades e técnicas para a implementação de componentes reutilizáveis.

Uma característica peculiar do estágio foi a alta rotação do estagiário entre diferentes projetos. Por um lado, isto permitiu maior amplitude de conhecimentos, pelo contato com diferentes tecnologias e tipos de atividades. Por outro lado, o curto tempo em cada projeto impossibilitou um aprofundamento em tecnologias específicas. Todavia, como resultado do estágio, o autor foi efetivado na organização, atuando como desenvolvedor *Front-End* júnior.

Durante todo o período de estágio, o curso de Ciência da Computação proporcionou uma base sólida e ampla sobre as teorias para a elaboração das tarefas apresentadas neste relatório. As principais disciplinas foram "Engenharia de Software", que garantiu a base para a elaboração dos projetos, e "Teste de Software", que forneceu os conhecimentos teóricos sobre teste. No entanto, o curso proporcionou poucas oportunidades de contato com tecnologias e ferramentas atuais adotadas na indústria. Assim, esta lacuna na formação do autor somente foi sanada com a atuação no estágio e com a participação na empresa júnior Comp Júnior, onde teve a oportunidade de aplicar todo conhecimento teórico em projetos práticos e em equipe, proporcionando habilidades na solução de problemas e trabalho em equipe.

Portanto, vale ressaltar a importância de empresas juniores para o desenvolvimento dos graduandos na universidade. Muitas das tecnologias e vivências empresariais apresentadas neste relatório, o estagiário teve a oportunidade de estudar e praticar dentro da empresa júnior (Comp Júnior), garantindo um local para errar e aprender sem ter o impacto de ser empresas seniores. Da mesma forma que a empresa júnior apresenta as principais tecnologias do mercado de trabalho, ela também abre portas para uma comunicação direta com empresas de tecnologia, possibilitando um meio de conseguir estágio muito mais facilmente, por já possuir uma rede de empresas em contato.

REFERÊNCIAS

AWS. **O que é NoSQL?** 2023. Disponível em: <https://aws.amazon.com/pt/nosql/>. Acesso em: Janeiro, 2023.

AZURE MICROSOFT. **O que é o Java Spring Boot?** 2023. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-java-spring-boot/>. Acesso em: 21 jan. 2023.

BANCO CENTRAL DO BRASIL. **Pix**. 2022. Disponível em: <https://www.bcb.gov.br/estabilidadefinanceira/pix>. Acesso em: 17 dez. 2022.

BANKS, A.; PORCELLO, E. **Learning React: functional web development with React and Redux**. [S.l.]: "O'Reilly Media, Inc.", 2017.

BECK, K. **JUnit Pocket Guide: Quick Look-up and Advice**. [S.l.]: "O'Reilly Media, Inc.", 2004.

BERNARDO, P. C.; KON, F. A importância dos testes automatizados. **Engenharia de Software Magazine**, v. 1, n. 3, p. 54–57, 2008.

CHACON, S.; STRAUB, B. **Pro git**. [S.l.]: Springer Nature, 2014.

CRUZ, F. **Scrum e PMBOK unidos no Gerenciamento de Projetos**. [S.l.]: Brasport, 2013.

DELAMARO, M.; JINO, M.; MALDONADO, J. **Introdução ao teste de software**. [S.l.]: Elsevier Brasil, 2013.

DIGITAL, D. **DTI Digital**. DTI Digital, 2023. Disponível em: <https://www.dtidigital.com.br/o-que-fazemos/>. Acesso em: Janeiro, 2023.

EISENMAN, B. **Learning react native: Building native mobile apps with JavaScript**. [S.l.]: "O'Reilly Media, Inc.", 2015.

IBM. **O que é um serviço da web?** 2023. Disponível em: <https://www.ibm.com/docs/pt-br/integration-bus/10.0?topic=services-what-is-web-service>. Acesso em: Fevereiro, 2023.

JACKSON, J. C. **Web Technologies**. [S.l.]: Pearson India, 2006.

JSON. **Introducing JSON**. 2023. Disponível em: <https://www.json.org/json-pt.html>. Acesso em: Janeiro, 2023.

LASTER, B. **Professional git**. [S.l.]: John Wiley & Sons, 2016.

LECHETA, R. R. **Web Services RESTful: aprenda a criar web services RESTful em Java na nuvem do Google**. [S.l.]: Novatec Editora, 2015.

LOURIDAS, P. Version control. **Ieee Software**, IEEE, v. 23, n. 1, p. 104–107, 2006.

MARTINS. **Grupo Martins**. Martins, 2023. Disponível em: <https://www.tribanco.com.br/institucional/sistema-martins/>. Acesso em: Janeiro, 2023.

META. **React: Uma biblioteca JavaScript para criar interfaces de usuário**. 2023. Disponível em: <https://pt-br.reactjs.org>. Acesso em: Janeiro, 2023.

NETO, A.; CLAUDIO, D. Introdução a teste de software. **Engenharia de Software Magazine**, v. 1, p. 22, 2007.

ORACLE. **O que é um banco de dados relacional (RDBMS)**. 2023. Disponível em: <https://www.oracle.com/br/database/what-is-a-relational-database/>. Acesso em: Janeiro, 2023.

PEREIRA, P.; TORREÃO, P.; MARÇAL, A. S. Entendendo scrum para gerenciar projetos de forma ágil. **Mundo PM**, v. 1, p. 3–11, 2007.

PHP. **PHP**. 2023. Disponível em: <https://www.php.net/manual/pt_BR/preface.php>. Acesso em: Janeiro, 2023.

PONTES, T. B.; ARTHAUD, D. D. B. Metodologias ágeis para o desenvolvimento de softwares. **Ciência e Sustentabilidade**, v. 4, n. 2, p. 173–213, 2018.

RIGA, M. **EXCLUSIVO: 76,2% das empresas já implementaram ou estão desenvolvendo planos de transformação digital**. Forbes Tech, 2021. Disponível em: <https://forbes.com.br/forbes-tech/2021/05/exclusivo-762-das-empresas-ja-implementaram-ou-estao-desenvolvendo-planos-de-transformacao-digital/>. Acesso em: Janeiro, 2023.

SALAS-ZÁRATE, M. del P. et al. Analyzing best practices on web development frameworks: The lift approach. **Science of Computer Programming**, Elsevier, v. 102, p. 1–19, 2015.

SCHWABER, K.; SUTHERLAND, J. The scrum guide. **Scrum Alliance**, v. 21, n. 1, p. 1–38, 2011.

SCHWABER, K.; SUTHERLAND, J. The scrum guide. 2020. **Accessed April**, 2021.

SHAH, D. N. **A Complete Guide To Internet And Web Programming**. [S.l.]: Dreamtech Press, 2009.

STACKOVERFLOW. **Web frameworks**. 2021. Disponível em: <https://insights.stackoverflow.com/survey/2021/#section-most-popular-technologies-web-frameworks>. Acesso em: 21 jan. 2023.

TRIBANCO. **Nossa História: O banco do Martins, do varejo e de Uberlândia.** 2023. Disponível em: <https://www.tribanco.com.br/institucional/o-banco/>. Acesso em: Janeiro, 2023.

TUDOSE, C. **JUnit in action**. [S.l.]: Simon and Schuster, 2020.

UMALI, R. **Learn Git in a Month of Lunches**. [S.l.]: Simon and Schuster, 2015.

VALENTE, M. T. Engenharia de software moderna. **Princípios e Práticas para Desenvolvimento de Software com Produtividade**, v. 1, p. 24, 2020.