PEDRO SILVEIRA LOPES

DESENVOLVIMENTO DE UM SISTEMA WEB E MOBILE PARA A INTEGRAÇÃO DE DOIS SISTEMAS JÁ EXISTENTES:

UM RELATÓRIO DO TRABALHO REALIZADO NA EMPRESA DELTA INOVA

LAVRAS – MG

2020

PEDRO SILVEIRA LOPES

DESENVOLVIMENTO DE UM SISTEMA WEB E MOBILE PARA A INTEGRAÇÃO DE DOIS SISTEMAS JÁ EXISTENTES:
UM RELATÓRIO DO TRABALHO REALIZADO NA EMPRESA DELTA INOVA

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

Profa. Dra. Juliana Galvani Greghi
Orientadora

LAVRAS – MG
2020

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Lopes, Pedro Silveira
Desenvolvimento de um sistema web e mobile para a integração de dois sistemas já existentes : Um relatório do trabalho realizado na empresa Delta Inova / Pedro Silveira Lopes. – Lavras : UFLA, 2020.
72 p. : il.

TCC(graduação)–Universidade Federal de Lavras, 2020.
Orientadora: Profa. Dra. Juliana Galvani Greghi.
Bibliografia.

1. React. 2. React Native. 3. Material Design. I. Greghi, Juliana Galvani. II. Título.

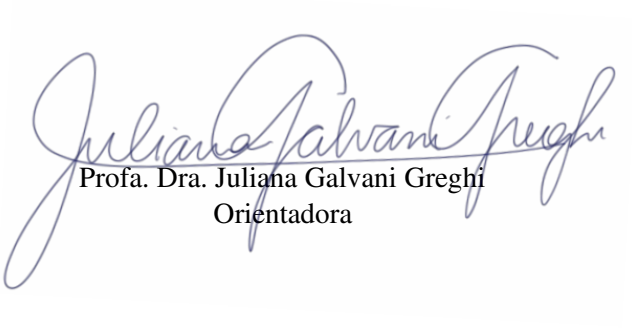
PEDRO SILVEIRA LOPES

DESENVOLVIMENTO DE UM SISTEMA WEB E MOBILE PARA A INTEGRAÇÃO DE DOIS SISTEMAS JÁ EXISTENTES: UM RELATÓRIO DO TRABALHO REALIZADO NA EMPRESA DELTA INOVA
DEVELOPMENT OF A WEB AND MOBILE SYSTEM FOR INTEGRATION OF TWO EXISTING SYSTEMS: A REPORT OF THE WORK DONE AT DELTA INOVA

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

APROVADA em 25 de agosto de 2020.

Profa. Dra. Renata Teles Moreira UFLA
Prof. Dr. Raphael Winckler de Bettio UFLA



Profa. Dra. Juliana Galvani Greghi
Orientadora

LAVRAS – MG
2020

À minha família, minha namorada e meus colegas do grupo Samurai, sem os quais eu não teria chegado até aqui.

AGRADECIMENTOS

Agradeço aos meus professores, que me prepararam para entrar no mundo da programação e à UFLA como um todo, que me proporcionou o melhor ambiente de estudos que eu poderia querer. Também sou muito grato à Delta por tudo. Grande parte do profissional que sou hoje é um reflexo do que aprendi nesse estágio. Uma experiência única que eu não trocaria por nada.

RESUMO

O objetivo do projeto descrito nesse documento foi a implementação de um sistema com plataformas WEB e *mobile* cuja função é integrar dois sistemas já existentes no Grupo Delta, um dos maiores grupos de serviços e tecnologias para o mercado de seguros da América Latina. Dois dos vários serviços oferecidos pelo grupo, e que foram o foco desse projeto de estágio, são o de rastreamento de veículos (chamado de Delta Sat) e o de assistência veicular 24 horas (chamado de Delta Assistance). Como esses dois serviços tinham clientes em comum, esse projeto foi criado com o objetivo de unificá-los em um novo sistema chamado de Delta Fleet, proporcionando aos clientes um acesso mais fácil às informações dos seus veículos, mais dados de *Business Intelligence* e, ao mesmo tempo, atraindo clientes que conheçam apenas um dos sistemas a conhecer o outro. A integração desses dois serviços em um novo sistema foi feita utilizando majoritariamente os frameworks React e React Native, com características do Material Design nas interfaces.

Palavras-chave: Estágio. React. React Native. Material Design.

ABSTRACT

The objective of the project described in this document was to implement a system with WEB and *mobile* platforms whose function is to integrate two existing systems in Grupo Delta, one of the largest groups of services and technologies for the insurance market in Latin America. Two of the various services offered by the group, which were the focus of this internship project, are vehicle tracking (called Delta Sat) and 24-hour vehicle assistance (called Delta Assistance). As these two services had customers in common, this project was created with the aim of unifying them in a new system called Delta Fleet, providing customers with easier access to their vehicle information, more data from *Business Intelligence* and, at the same time, attracting customers who know only one of the systems to know the other. The combination of these two services in a new system was made using mostly the React and React Native frameworks, with Material Design characteristics in the interfaces.

Keywords: Internship. React. React Native. Material Design.

LISTA DE FIGURAS

Figura 2.1 – Modelo Cascata 19
Figura 2.2 – Exemplo de Componentização 27
Figura 2.3 – Ciclo de Vida 29
Figura 3.1 – Desenho da principal do sistema web 32
Figura 3.2 – Desenho da principal do aplicativo móvel 34
Figura 3.3 – Componentes que utilizam dados dos veículos 36
Figura 3.4 – Código ComponenteSimples 39
Figura 3.5 – Código ComponenteSimples em React Native 39
Figura 3.6 – Exemplo de uso do ComponenteSimples 40
Figura 3.7 – ComponenteSimples usando *props* 41
Figura 3.8 – Utilizando o ComponenteSimples com *props* 42
Figura 3.9 – Contador: Exemplo de componente que utiliza *state* 43
Figura 3.10 – Contador em React Native 44
Figura 3.11 – Ações Redux 46
Figura 3.12 – Redux *Reducer* 47
Figura 3.13 – Redux *Store* 48
Figura 3.14 – Redux *Provider* 48
Figura 3.15 – Exemplo de uso do Redux 49
Figura 4.1 – Estande Delta Fleet na 22ª Fenatran 53
Figura 4.2 – Componentização da sessão “Minha Frota” 55
Figura 4.3 – Componentização “ListaVeiculos” 55
Figura 4.4 – Tela inicial do sistema web 57
Figura 4.5 – Tela da frota no sistema web 58
Figura 4.6 – Tela do mapa no sistema web 59
Figura 4.7 – Componente “Autocomplete” do Material UI 59

Figura 4.8 – Componente “Autocomplete” onde as opções foram customizadas e mostram dados sobre um veículo (placa, marca, modelo e imagem da marca) 60
Figura 4.9 – Tela principal do app 62
Figura 4.10 – Tela com lista de todos os veículos do cliente no app 63
Figura 4.11 – Tela aplicativo Uber 64
Figura 4.12 – Tela inspirada no app Uber mostrado na Figura 4.11 65
Figura 4.13 – Tela da Figura 4.12 porém com o mapa à mostra 66
Figura 4.14 – Desenho da tela com todos os veículos 67
Figura 4.15 – Mapa com grupos de marcadores 68

SUMÁRIO

1 INTRODUÇÃO 17
2 REFERENCIAL TEÓRICO 19
2.1 Engenharia de Software 19
2.2 Produto Mínimo Viável 21
2.3 Interação Humano-Computador 22
2.4 Material UI 23
2.5 Arquitetura REST 24
2.6 Desenvolvimento Web 25
2.7 API 26
2.8 Javascript 26
2.9 React 26
2.9.1 Ciclo de vida dos componentes 28
2.10 React Native 29
2.11 Redux 30
2.12 PHP 30
3 DESCRIÇÃO DAS ATIVIDADES 31
3.1 Design 31
3.1.1 Design das interfaces 31
3.1.2 Escolha das Tecnologia e estratégias de desenvolvimento 33
3.1.3 Componentização das telas 35
3.1.4 Redux 36
3.2 Implementação 37
3.2.1 Criação de componentes 38
3.2.2 Utilização de propriedades 40
3.2.3 Utilização de estados 41
3.2.4 Utilização do Redux 45
3.2.5 Implementação da aplicação web 47

3.2.6 Implementação da aplicação para dispositivos móveis 50
3.3 Testes 51
4 RESULTADOS 53
4.1 Componentização das telas 54
4.2 Utilização do Redux 56
4.3 Implementação da Aplicação Web 56
4.3.1 Pesquisa de veículos 57
4.3.2 Lista de veículos com rolagem infinita 60
4.4 Implementação da aplicação para dispositivos móveis 61
4.4.1 Telas com mapa escondido 62
4.4.2 Mapa com todos os veículos 65
5 CONCLUSÃO 69
REFERÊNCIAS 71

1 INTRODUÇÃO

O primeiro contato com o mercado de trabalho geralmente é muito difícil em qualquer profissão, principalmente porque a maioria das empresas contratantes dão preferência para profissionais que já tenham experiência na área. Esse é um dos motivos que faz com que muitos recém-formados em cursos superiores não encontrem emprego na área para a qual estudaram por vários anos e tenham que atuar em outras áreas para obter renda (1, 2).

O estágio abre as portas para que o estudante, mesmo antes de terminar sua graduação, consiga ter essa primeira experiência com o mercado, podendo auxiliar na resolução de problemas reais de uma empresa em funcionamento, aplicando as teorias que aprendeu em seus estudos. Isso pode se tornar o grande diferencial para sua contratação posteriormente.

O estágio descrito nesse trabalho foi realizado, durante o período de 13 de agosto à 20 de dezembro de 2019, na Delta Inova, a fábrica de *softwares* do Grupo Delta. As informações sobre o Grupo Delta e as empresas que o compõe, que serão apresentadas a seguir, foram obtidas de forma verbal [1].

O Grupo Delta é um grupo de empresas que nasceu com a ideia de dois sócios de oferecer assistência 24 horas para o mercado automotivo do Brasil. A partir disso, outros dois sócios foram convidados, um para cuidar da gestão dos negócios e outro para cuidar da gestão tecnológica. Posteriormente outras duas pessoas investiram na empresa e se tornaram sócios investidores. Com sua sede principal em Porto Alegre - RS e uma base tecnológica em Lavras - MG, o Grupo Delta teve uma rápida expansão e conquistou a confiança de muitos clientes em todo o Brasil.

Inicialmente foi criada a Delta Assist, empresa de assistência 24 horas para veículos. A empresa conta com mais de 5000 prestadores de serviço qualificados

[1] Informações fornecidas pelo sócio do Grupo Delta, Marzon Castilho, em 27 de março de 2020

em todo o Brasil para garantir o atendimento de todo tipo de chamado (panes mecânicas, tombamentos, acidentes, etc.) e equipes trabalhando 24 horas por dia.

Enquanto a empresa amadurecia, foram percebidas outras necessidades do mercado veicular como a necessidade de rastrear o posicionamento dos automóveis, tanto por questões de segurança quanto para garantir que não estão sendo utilizados de forma errada. Por isso foi criada a Delta Sat, a segunda empresa do Grupo Delta com um sistema que se comunica com o rastreador instalado nos veículos dos clientes, permitindo monitorar a posição dos veículos em tempo real, gerar relatórios dos locais onde esse veículo esteve e também notificar quando algum evento importante acontecer (como a entrada/saída do veículo em uma área de interesse ou quando a velocidade exceder um limite configurado pelo usuário).

Com o amadurecimento da empresa, notou-se que a Delta Sat e a Delta Assist possuem públicos-alvo e clientela muito parecidos. A partir disso, foi criada a proposta de desenvolver um sistema chamado Delta Fleet que integrasse os serviços das duas empresas para facilitar o acesso às informações pelos clientes e também para que aqueles que só conheçam os serviços de uma das empresas, possam também conhecer os da outra. Esse se tornou o objetivo do estágio: criar um sistema web e um aplicativo *mobile* que consigam integrar os serviços das duas empresas de forma organizada, sem sobrecarregar a interface com informações. Nossa equipe contava com cinco programadores na cidade de Lavras, MG e um designer em Porto Alegre, RS.

Este trabalho está estruturado da seguinte forma: no Capítulo 2 será apresentado o referencial teórico com os conceitos essenciais para o bom entendimento do projeto; no Capítulo 3 serão descritas as atividades desenvolvidas durante projeto; no Capítulo 4 serão mostrados os resultados obtidos no final do estágio; no Capítulo 5 serão apresentadas as conclusões sobre esta experiência.

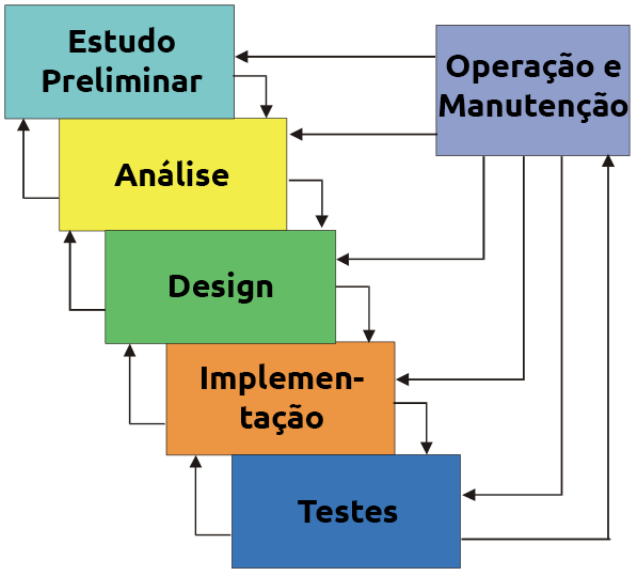
2 REFERENCIAL TEÓRICO

Um único projeto de sistema computacional geralmente abrange uma grande gama de áreas da computação e, como muitas empresas tendem a funcionar com equipes reduzidas, os desenvolvedores também têm que atuar em mais de uma área ao mesmo tempo. Nesse capítulo, serão mostrados os assuntos que foram essenciais para o bom andamento das atividades durante o estágio.

2.1 Engenharia de Software

Engenharia de software é uma área de estudo focada nos fundamentos, métodos, processos e técnicas de desenvolvimento, manutenção e gerência de software e, por isso, está presente em todos os projetos de software (3). Uma visão sistemática simplificada do ciclo de vida de um software é bem representada pelo modelo em cascata, mostrado na Figura 2.1. Ela também mostra que o desenvolvimento do software não é linear, podendo retornar à etapas anteriores.

Figura 2.1 – Modelo Cascata



Fonte: Adaptado de *Fundamentals of Software Engineering* (3)

Porém, o modelo cascata assume que uma etapa tem de ser completamente concluída para dar início à próxima (4), o que não se encaixava nesse projeto já que o produto tinha um prazo curto de entrega em que quanto mais funcionalidades fossem entregues, melhor. Portanto, a equipe não teria tempo para executar todas as etapas completamente antes da entrega.

Por isso, foi utilizado o modelo de desenvolvimento iterativo incremental, que é baseado no modelo cascata mas, ao invés de assumir que as etapas serão concluídas completamente, o projeto é dividido em várias partes (iterações). Cada iteração segue as mesmas etapas do modelo cascata mas apenas para uma pequena parte do sistema (4), dessa forma entregando novas funcionalidades ao sistema ao fim de cada iteração.

Esse modelo de desenvolvimento é o mais adequado ao processo da empresa, que consiste nas seguintes etapas:

1. Levantamento de requisito: Grande parte dos requisitos já haviam sido levantados durante o desenvolvimento dos sistemas Delta Sat e Delta Assist. Os requisitos de novas funcionalidades foram levantados pelos vendedores de Porto Alegre que estavam em contato direto com os clientes. Todos os requisitos são registrados em um kanban no site do Trello [1].

2. Designação das tarefas: O gerente da equipe de desenvolvimento analisa as tarefas pendentes nesse kanban e define quais tarefas cada desenvolvedor realizará até a próxima entrega.

3. Implementação: O desenvolvedor implementa as tarefas que foram designadas a ele.

4. Entrega: Periodicamente é gerada uma nova versão de teste do sistema. Esse período geralmente é de sete dias. Quando essa nova versão é gerada,

[1] https://trello.com/

os desenvolvedores ficam responsáveis por entrar em contato com a equipe de Porto Alegre e para apresentar e testar as novas funcionalidades.

Dessa forma, por exemplo, parte do design das interfaces poderia ser liberado para que os desenvolvedores dessem início à implementação antes do design completo estar concluído. Ao final de cada iteração, o resultado é mostrado aos clientes e, caso seja necessário alterar algo, essa iteração pode voltar algumas etapas.

É necessário, entretanto, que sejam feitas algumas considerações a respeito das etapas realizadas. Por se tratar da integração de sistemas já existentes e, de certa forma, consolidados, grande parte das etapas de estudos preliminares e análise de requisitos foram reaproveitadas dos desenvolvimentos anteriores.

Logo, as etapas principais que foram englobadas por esse projeto de estágio foram as várias iterações de:

1. Design: Estruturação de como será feita a ligação entre os dois sistemas, projeto das telas, escolha das tecnologias e planejamento dos componentes;

2. Implementação: Desenvolvimento do código do sistema;

3. Testes: Testes realizados em um ambiente simulando o de produção.

2.2 Produto Mínimo Viável

O foco do projeto foi a criação de um Produto Mínimo Viável (5), que é uma versão simplificada do produto final de um projeto. Nessa versão simplificada, a empresa oferece o mínimo de funcionalidades com o objetivo de testar a aceitação do produto no mercado. Caso o produto se mostre viável, funcionalidades vão sendo adicionadas até que ele fique completo.

2.3 Interação Humano-Computador

Segundo a Sociedade Brasileira de Computação [2] (SBC), a Interação Humano-Computador (IHC) é uma área da computação que se dedica a estudar os fenômenos de comunicação entre pessoas e sistemas computacionais. Por englobar tanto computadores quanto pessoas, a IHC é multidisciplinar, envolvendo ciências da computação, da informação e ciências sociais e comportamentais (6). As pesquisas realizadas nessa área são muito importantes para explicar e prever fenômenos da interação de usuários com sistemas e, assim, identificar falhas e possíveis melhorias de design.

O design da interface foi um dos desafios do planejamento do Delta Fleet. Como o sistema proposto une os dados e funcionalidades de outros dois sistemas, o design teve de ser muito bem pensado, já que a tendência seria a de ocorrer uma sobrecarga de informações. Sobrecarregar o usuário pode causar incertezas na manipulação do sistema e levar a sensação de frustração, o que pode ser definitivo na aceitação ou não do produto. Pensando nisso, os pilares para o desenvolvimento da interface do projeto foram os 6 princípios de usabilidade de Donald Norman (7):

1. Visibilidade: As partes corretas devem estar visíveis e elas devem transmitir a mensagem correta a respeito de sua funcionalidade. O design do menu que possibilita a navegação no site, por exemplo, deve mostrar com clareza ao usuário onde ele será levado;

2. *Feedback*: É muito importante mostrar ao usuário os efeitos de suas ações. Ao clicar em um botão, por exemplo, deve ficar claro o que este clique causou no sistema;

3. *Affordance*: Este termo vem do inglês e não tem tradução direta, mas refere-se à capacidade de um objeto transmitir sua função e forma de usar sem precisar de instruções. Um botão com uma seta para a esquerda, por

[2] http://www.sbc.org.br

exemplo, nos remete à ação de voltar e um com o símbolo de uma lixeira nos remete à ação de apagar algo, jogar fora;

4. Mapeamento: Um termo que veio da matemática e que representa o relacionamento entre duas coisas. Em um jogo, por exemplo, a botão do teclado com a seta para a direita move o jogador para a direita e não o contrário;

5. Restrições: Nas palavras de Norman (7), "A maneira mais segura de tornar alguma coisa fácil de usar, com poucos erros, é tornar impossível fazê-la de outro modo, restringindo a quantidade de escolhas", portanto, o usuário deve ter liberdade no sistema, mas não deve ter a liberdade de cometer erros, quando for possível;

6. Consistência: Interfaces que tem funções semelhantes devem manter uma consistência no seu design. Por exemplo, manter o mesmo design em todos os botões que enviam formulários no sistema pode evitar que o usuário confunda a função do botão.

Seguindo esses fundamentos, alguns estilos de design foram estudados. O que se destacou dos demais foi o Material Design (8), criado pela Google e utilizado por ela na maioria dos seus aplicativos. Ele reúne um conjunto de boas práticas na criação de interfaces, como padrões de botões com *feedback* visual de clique, campos de texto com *labels* padronizadas, etc., mas também permite customização para deixar a marca da empresa no design.

2.4 Material UI

O Material UI é um *framework* que ajuda na criação de interfaces para se utilizar com o React (9) e foi utilizado no desenvolvimento do *front end* do sistema web. Ele reúne um conjunto de componentes que seguem os padrões definidos pelo Material Design (8), o que permite criar interfaces muito atrativas e

funcionais sem precisar se ater à customização do visual dos componentes. As especificações dos componentes fornecidos por esse *framework* podem ser encontradas na documentação do Material UI (9).

2.5 Arquitetura REST

Representation State Transfer (REST) é um estilo arquitetural para sistemas hipermídia (sistema que permite acesso a textos, imagens estáticas ou em movimento, sons, softwares, etc. a partir de links que acionam outros documentos e assim sucessivamente) distribuídos, criada por Roy Fielding (10) e foi utilizado na criação do *back end* da aplicação.

Em sua tese, ele definiu cinco características para que um sistema se enquadre na arquitetura REST (10):

1. **Cliente-servidor:** Separando a interface do usuário da lógica de armazenamento, organização e processamento dos dados existe um aumento da portabilidade (já que múltiplas plataformas podem acessar o mesmo servidor) e da escalabilidade (por simplificar os componentes do servidor);

2. **Sem estado:** Cada requisição do cliente ao servidor deve conter toda a informação necessária para atender àquela requisição e não deve levar em conta nenhum contexto armazenado no servidor. Uma sessão, por exemplo, deve ficar inteiramente armazenada no lado cliente;

3. **Armazenável em cache:** Os dados de uma resposta a uma requisição devem ser explicita ou implicitamente marcados como armazenável ou não em cache. Se for armazenável, o cliente pode reutilizar aquela resposta sem ter que requisitar novamente ao servidor;

4. **Interface uniforme:** Ao aplicar o princípio de generalidade da Engenharia de Software à interface do componente, a arquitetura geral do sistema é

simplificada e a visibilidade das interações é aprimorada. Para obter uma interface uniforme, são necessárias várias restrições arquiteturais para orientar o comportamento dos componentes. O REST é definido por quatro restrições de interface: identificação de recursos; manipulação de recursos através de representações; mensagens auto-descritivas; e hipermídia como o mecanismo do estado do aplicativo;

5. **Sistema em camadas:** O estilo do sistema em camadas permite que uma arquitetura seja composta de camadas hierárquicas, restringindo o comportamento do componente, de modo que cada componente não possa ter acesso a outra camada que não aquela com a qual está interagindo.

Aos sistemas que se enquadram na arquitetura REST, dá-se o nome de RESTful.

2.6 Desenvolvimento Web

O desenvolvimento web diz respeito ao desenvolvimento de sites para a internet ou um rede privada de computadores. O desenvolvimento web geralmente é dividido em duas áreas que se complementam:

1. *Front end*: A interface do site, onde o usuário pode interagir com o sistema. Em aplicações web, é feito utilizando tecnologias que podem ser interpretadas por browsers, como HTML, CSS e JavaScript (11).

2. *Back end*: A lógica de negócios por trás do funcionamento do sistema. Geralmente situado em servidor, faz o acesso ao banco de dados, processa os dados e retorna uma resposta ao *front end*, o que alivia o processamento da máquina do usuário (11).

O *back end* e *front end* funcionam em conjunto para a formação dos sistemas web. Questões como segurança dos dados, por exemplo, devem ser levados

em conta no desenvolvimento de ambas as áreas. Nesse projeto, o foco será no *front end* já que grande parte do código dos servidores que estavam em funcionamento pôde ser reaproveitado.

2.7 API

Uma API (do inglês *Application Programming Interface*, em tradução direta, Interface de Programação de Aplicativos) é um conjunto de protocolos e regras definidos para especificar como será a interação entre dois softwares (12, 13). Neste projeto, APIs foram utilizadas para a comunicação na internet, especificamente para comunicação entre o *front end* e o *back end* da aplicação.

2.8 Javascript

Javascript, ou apenas JS, é uma linguagem leve e interpretada que se popularizou como uma linguagem de *scripts* para páginas Web. Hoje ela já é utilizada em vários outros ambientes sem *browser* (14), como em servidores utilizando NodeJS e aplicativos *mobile* utilizando React Native. Além disso, atualmente está entre as 7 linguagens mais utilizadas no mundo (15).

Com sua popularização e sua flexibilidade, vários *frameworks* e bibliotecas foram criados para potencializar o uso do Javascript. Dois deles são React e React Native, que foram utilizados neste projeto e serão detalhados nas seções seguintes.

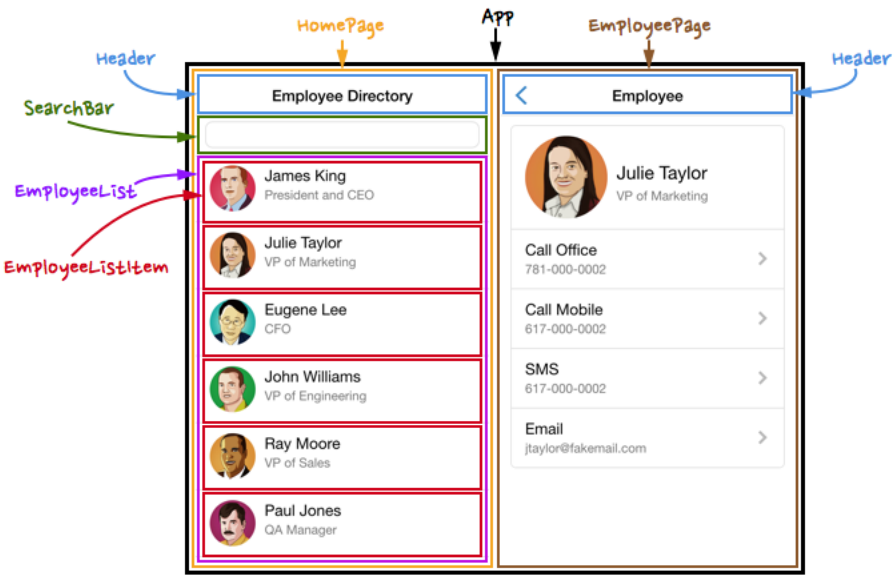
2.9 React

React é uma biblioteca do Javascript usada para a criação de interfaces em páginas Web (16). Ele foi desenvolvido e disponibilizado pelo Facebook e hoje é mantido por uma enorme comunidade de desenvolvedores voluntários em um

repositório aberto no GitHub [3] que já conta com mais de 13000 commits, que são alterações do código fonte do repositório.

A principal característica do React é ser baseado em componentes encapsulados (16), que representam funcionalidades isoladas do sistema, e podem ser utilizados em vários locais diferentes na mesma aplicação. A junção de vários pequenos componentes cria componentes cada vez mais complexos, até a formação da aplicação em si, como ilustrado na Figura 2.2.

Figura 2.2 – Exemplo de Componentização



Fonte:
https://blog.rocketseat.com.br/react-do-zero-componentizacao-propriedades-e-estado/
acessado em 31/08/2020

No React tudo que é renderizado na tela é um componente, inclusive a própria aplicação, mostrada como "App" na Figura 2.2. Cada componente é responsável por gerenciar suas próprias informações, o que faz com que o código fique muito mais organizado (16).

[3] https://github.com/facebook/react

Cada componente recebe as chamadas *props*, que são propriedades para o seu funcionamento (17), característica que veio do HTML. Sempre que as propriedades de um componente forem alteradas, o react sabe que uma nova renderização será necessária para atualizar os dados mostrados na tela (18). Um exemplo de utilização de propriedades pode ser encontrado na subseção 3.2.2.

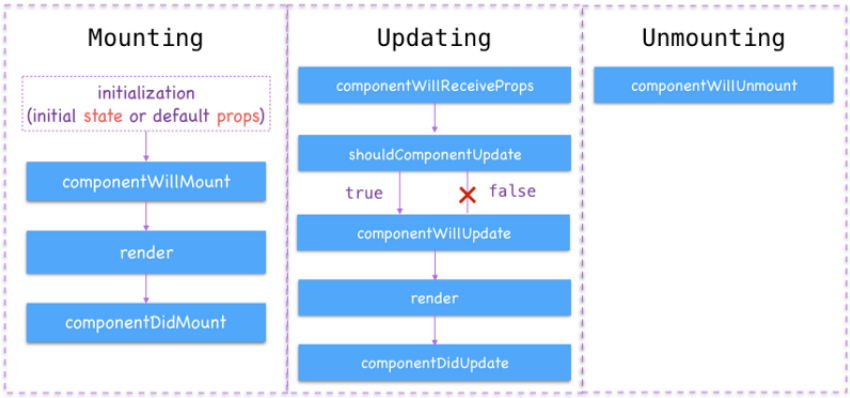
O estado de um componente, ou *state*, é um conceito parecido com o de propriedades, mas ao invés de ser passado ao componente, o estado é configurado dentro da própria classe e ela é responsável por gerenciá-lo (19). Um exemplo de utilização de estados pode ser encontrado na subseção 3.2.3.

2.9.1 Ciclo de vida dos componentes

Todos os componentes no React possuem ciclo de vida (19): primeiro o componente é montado, depois ele pode sofrer alterações (caso haja mudanças no estado ou nas propriedades do componente, por exemplo) e por fim o componente é desmontado quando não estiver mais sendo usado. A cada passo desse ciclo existe uma função correspondente na classe Component que pode ser sobrescrita para interceptar o fluxo de renderização normal do componente. As funções e as etapas do ciclo de vida de um componente são mostradas na figura 2.3. Uma dessas funções, e a única que não vem implementada por padrão na classe Component, é a função “render”, que define o que será mostrado na tela.

Um uso muito comum dessas funções é para fazer a busca de dados no *back end* da aplicação sempre que o componente é criado. Isso pode ser feito implementando a função “componentDidMount” para chamar as funções de busca e armazenar os dados retornados em uma variável de estado. Alterações no estado da aplicação só são permitidas após a montagem do componente, portanto, esse tipo de operação não é permitida no *construtor* da classe, por exemplo. Existem muitas recomendações do que se deve fazer e do que se deve evitar durante a implementação de um componente na documentação oficial do React (20).

Figura 2.3 – Ciclo de Vida



Fonte: http://nitrajka.com/

2.10 React Native

O React Native (RN) também é uma biblioteca Javascript mantida pelo Facebook [4] e segue o mesmo padrão do React, com componentização, propriedades, estado e ciclo de vida. A principal aplicação desse *framework* está na construção dos chamados aplicativos *mobile* híbridos, que são aqueles que funcionam tanto para iOS quanto Android com um único código.

Algumas bibliotecas que prometem criar aplicações híbridas fazem isso utilizando *web views*, que são componentes que agem como um *browser*, renderizando a aplicação escrita em HTML, CSS e JavaScript, mas esse tipo de abordagem tem alguns pontos fracos principalmente em relação à performance (21, 22). Já o React Native funciona de uma forma totalmente diferente.

Seu funcionamento é agir como uma ponte entre os códigos nativos Java (Android) e Swift (iOS), possibilitando a criação de aplicações híbridas com desempenho equiparável à de aplicações nativas (que são aquelas escritas exclusivamente para a plataforma) e com acesso a funções do sistema operacional do celular, como utilização de câmera, microfone, GPS, etc (21, 22, 23).

[4] https://github.com/facebook/react-native/

Por não utilizar *web views*, não há HTML/CSS dentro das aplicações React Native. A biblioteca tem suas próprias *tags* e componentes de estilização, detalhados na documentação oficial (24).

2.11 Redux

O Redux é uma biblioteca JavaScript utilizada para gerenciar estados globais, que são dados disponíveis em toda a aplicação. Um exemplo de uso dessa biblioteca é para armazenar os dados do usuário que está logado, permitindo que esses dados sejam acessados em toda a aplicação (25).

O uso do Redux pode ser muito benéfico para diminuir o número de acessos ao servidor, pois componentes distintos, que utilizam os mesmos dados, podem buscá-los apenas uma vez no servidor e acessá-los simultaneamente pelo Redux. Um exemplo de utilização do Redux será apresentado na subseção 3.2.4.

2.12 PHP

Algumas alterações no *back end* do Delta Sat e Delta Assist tiveram de ser feitas para que fosse possível o seu correto funcionamento em conjunto. O servidor de ambas as aplicações segue a arquitetura REST para fornecer uma API de consulta implementada utilizando PHP.

O PHP é uma linguagem que foi criada em 1994 por Rasmus Lerdorf e que, de forma similar ao Javascript, nasceu e se popularizou como uma linguagem de *scripts* para Web, mas hoje já é capaz de fazer muito mais que isso fora dos *browsers* (26, 27). Apesar de não estar entre as linguagens mais novas, o PHP ainda é muito utilizado, sendo a 8º linguagem mais utilizada no mundo segundo o índice TIOBE (15).

3 DESCRIÇÃO DAS ATIVIDADES

Neste capítulo, serão descritas as etapas de desenvolvimento realizadas para a obtenção do resultado final do projeto. Foi seguido o modelo incremental, que tem as mesmas etapas que o modelo apresentado na Figura 2.1. O prazo de desenvolvimento era entregar um produto mínimo viável no dia 10 de outubro de 2019 para que ele fosse testado antes do início da 22ª edição da Feira Nacional do Transporte (Fenatran), onde seria apresentado ao público.

3.1 Design

Além do design das interfaces do sistema, na etapa de Design foram escolhidas as tecnologias utilizadas e definidas as estratégias de desenvolvimento do projeto.

3.1.1 Design das interfaces

As interfaces do sistemas foram projetadas por um designer profissional contratado pela empresa. Durante todo o processo, o designer esteve em contato com os programadores e outros funcionários para discutir ideias, possíveis dificuldades de implementação e alterações necessárias. A referência utilizada pela equipe durante todo o processo foi o Material Design da Google (8) e os 6 fundamentos de Donald Norman (7).

O programa utilizado para o desenho das telas foi o Adobe XD, uma ferramenta usada para desenvolver interfaces de aplicativos da Web e móveis, disponibilizada gratuitamente pela Adobe (28). O protótipo da tela principal do sistema web está mostrado na Figura 3.1, onde estão destacadas informações sobre o contrato do cliente, alguns veículos de sua frota de veículos e gráficos sobre os veículos. Apenas as partes mais importantes das demais telas serão mostradas neste documento.

Figura 3.1 – Desenho da principal do sistema web

Algumas características dos fundamentos de Norman já podem ser notados no desenho da tela:

- Visibilidade: A tela inicial contém uma visão geral de todo o sistema com uma barra de navegação no topo que permite ao usuário navegar entre as principais telas do sistema. Além disso, botões logo abaixo do título permitem o acesso a funções específicas do programa, como acionar a assistência técnica ou rastrear seus veículos.

- *Affordance*: Os ícones e as cores dos botões ajudam a transmitir a função deles. O botão "Acionar Assistência", por exemplo, é vermelho e contém um símbolo que remete a "atenção", pois deve ser utilizado em casos em que a assistência é necessária (geralmente acidentes na estrada ou panes).

- Consistência: Existe um padrão de cores e estilos que são utilizados em todo o site, ajudando a criar uma navegação mais intuitiva. Um exemplo disso são os componentes com cor vermelha que apresentam dados ou funções que precisam de atenção.

As mesmas características estão presentes no desenho do sistema *mobile*, apresentado na Figura 3.2. A consistência entre o design das duas aplicações também é mantida para que o cliente, familiarizado com uma das plataformas, não tenha problemas em utilizar a outra.

3.1.2 Escolha das Tecnologia e estratégias de desenvolvimento

A escolha de tecnologias certas para um projeto tem impacto na eficiência de implementação, na manutenibilidade do código e também na facilidade de contratar profissionais para trabalhar no sistema. Como o objetivo do projeto Delta Fleet foi unir as funcionalidades dos sistemas Delta Sat e Delta Assist, a escolha das tecnologias foi influenciada pelos sistemas existentes.

O Quadro 3.1 mostra as tecnologias utilizadas nos sistemas do Grupo Delta, bem como as que foram escolhidas para o novo sistema. Observe que ambos os sistemas que já estavam em funcionamento utilizam linguagem PHP no

Figura 3.2 – Desenho da principal do aplicativo móvel

back end e, como a maioria das lógicas de negócios implementadas poderiam ser reaproveitadas, foi decidido que o *back end* do Delta Fleet também usaria PHP. Para facilitar ainda mais a manutenção e o reaproveitamento do código, foi decidido que o Delta Fleet utilizaria o mesmo *back end* que o Delta Sat, de forma que apenas as novas funções referentes à consulta de dados na Delta Assist tiveram de ser implementadas.

Quadro 3.1 – Tecnologias utilizadas no Grupo Delta

	Back End	Front End
Delta Sat	PHP	PHP (sem *framework*)
Delta Assist	PHP	JS (React)
Delta Fleet	PHP	JS (React)

No *front end*, a Delta Assist utiliza PHP sem a utilização de *frameworks* e a Delta Sat utiliza JavaScript com o *framework* React. Por se tratar de uma tecnologia em alta no mercado e que funcionou muito bem no sistema Delta Sat, o React foi escolhido. Para agilizar o desenvolvimento do novo sistema, a equipe decidiu utilizar o *front end* do Delta Sat como base para o desenvolvimento do Delta Fleet, ou seja, esse projeto foi clonado e apenas as atualizações de *layout* necessárias e as funcionalidades referentes à assistência 24h foram adicionadas. Decidiu-se, também, que as telas que fossem consideradas secundárias, mesmo que precisassem de algumas modificações de design, poderiam ser mantidas do sistema já pronto e posteriormente modificadas.

Como o sistema Delta Fleet seguiria a arquitetura REST, onde uma das características é a divisão cliente-servidor (10), a API dessa aplicação também poderia ser utilizada pelo aplicativo *mobile*. Já no *front end* desse aplicativo foi definida a utilização de React Native, pois essa tecnologia é semelhante ao React e isso facilitaria para os programadores que não precisariam aprender dois *frameworks* totalmente diferentes.

Apesar de o React Native não utilizar HTML e CSS e não permitir o reaproveitamento dos componentes da aplicação web, essa tecnologia foi escolhida por preocupação com a performance que seria necessária no app devido a algumas telas complexas.

3.1.3 Componentização das telas

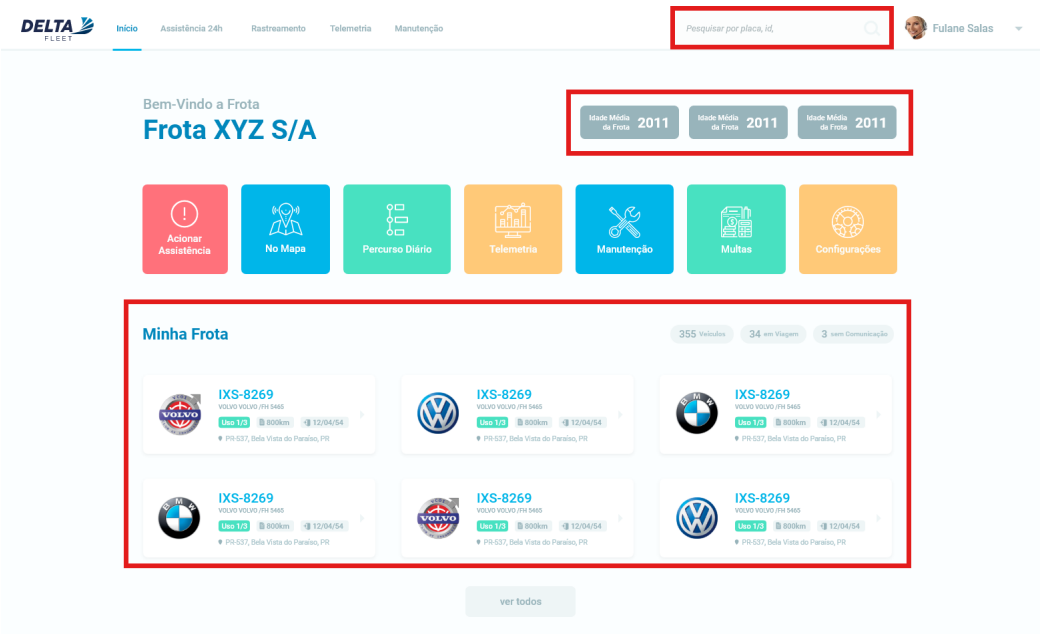
Componentizar uma tela significa dividi-la em componentes do React, algo similar ao mostrado na Figura 2.2. Esse processo de planejamento dos componentes antes da implementação é importante para evitar que as classes fiquem excessivamente complexas e desorganizadas. Também é possível definir qual componente ficará encarregado de buscar as informações do servidor e o que cada componente receberá através de suas propriedades.

O processo de componentização não é determinístico, ou seja, cada desenvolvedor pode enxergar a divisão de componentes de uma forma diferente e, mesmo assim, obter um resultado visualmente parecido depois de implementá-los. O estagiário ficou responsável por iniciar o desenvolvimento tanto da aplicação web quanto da aplicação *mobile*, e também por esse planejamento.

3.1.4 Redux

Após analisar os componentes planejados na etapa anterior, foi notado que alguns dados seriam compartilhados por vários componentes diferentes. Se cada um desses componentes fizesse um acesso ao servidor para buscar esses dados, o *back end* poderia ficar sobrecarregado. A Figura 3.3, por exemplo, destaca 3 componentes que utilizariam dados sobre a lista de veículos do cliente.

Figura 3.3 – Componentes que utilizam dados dos veículos



Em casos onde os componentes estão próximos na hierarquia do website, esse problema pode ser resolvido com o uso de propriedades, como destacado na subseção 3.1.3. Mas em casos onde esses componentes estão distantes uns

dos outros, não é recomendado que uma propriedade seja passada muitos níveis abaixo, pois isso diminui a qualidade do sistema, dificultando a manutenção do código.

Por exemplo, se um componente “A” tem acesso a um dado que é utilizado em um componente “D”, mas eles estão há três níveis de distância, essa propriedade teria de ser passado de “A” para “B”,de “B” para “C” e, finalmente, de “C” para “D”.

Após pesquisar algumas alternativas de como resolver esse problema, a que foi considerada a melhor para o projeto foi a possibilidade de manter alguns dados globalmente disponíveis na aplicação utilizando os chamados gerenciadores de estados globais. Existem várias bibliotecas que permitem que isso seja feito, mas a mais comumente utilizada na comunidade do React é o Redux.

Dentro do React, o Redux funciona passando o estado global através das propriedades dos componentes que “se inscrevem” para recebê-lo. Um exemplo da utilização do Redux será mostrado na subseção 3.2.4.

Inicialmente, o Redux não se mostrou tão necessário na aplicação *mobile* pois nela haviam poucas telas se comparado ao sistema web. Entretanto, algumas telas foram adicionadas à aplicação ao longo do projeto e o uso do Redux voltou a ser discutido e provavelmente será implementado futuramente para evitar acessos ao servidor e telas de carregamento. Isso mostra como um projeto que inicia pequeno pode escalar no crescimento rapidamente e, por isso, deve-se tentar prever os diversos cenários futuros para que possa ser criado projeto robusto desde o início.

3.2 Implementação

Depois que os componentes foram planejados, iniciou-se a fase de implementação do sistema. Antes que a fase de implementação das aplicações web

ou *mobile* seja descrita, serão apresentados exemplos de como são os códigos de componentes React, mostrando também as diferenças em relação ao React Native.

3.2.1 Criação de componentes

Nesta subseção, será exemplificado como é o código de um componente, mostrando o método mais comum de criação de um componente chamado "ComponenteSimples" utilizando React e posteriormente será mostrado o mesmo componente em React Native. Existem outras formas de criar componentes, especificadas na documentação do React (16), mas elas não foram utilizados no projeto.

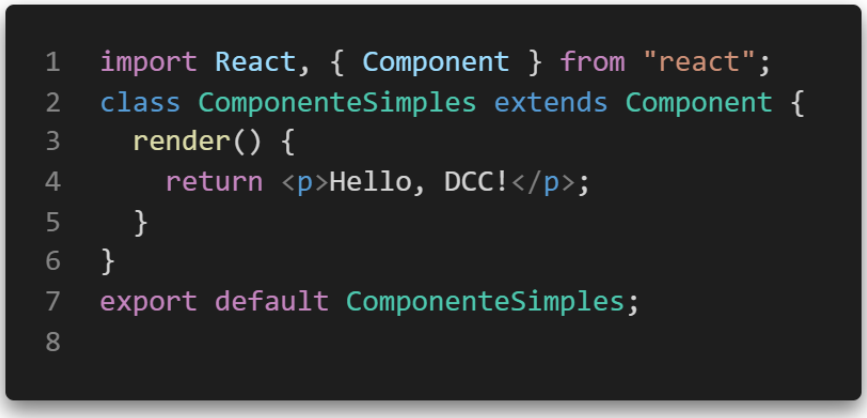
O primeiro passo para a criação de um componente é a criação de um arquivo com a extensão ".js" e a importação do React e da classe Component. Essa é a classe base dos componentes, que contém todo o ciclo de vida padrão do React, outra característica importante dessa biblioteca, detalhada na subseção 2.9.1.

Depois disso deve-se criar uma classe filha da Component e implementar o método "render". Esse método é um dos mais importantes do ciclo de vida do componente (subseção 2.9.1) pois define o que será mostrado na tela, sem o qual o componente não existe. O código para esse componente está na Figura 3.4. Vale ressaltar que esse é apenas um exemplo de componente e não foi utilizado na aplicação.

Esse mesmo componente poderia ser criado em React Native com o código apresentado na Figura 3.5. Repare que a estrutura do componente é a mesma, a única diferença é que a *tag* "p" teve de ser substituída por um componente "Text" do React Native.

Para utilizar esse componente tanto em React queanto em React Native, basta importar a classe "ComponenteSimples" e colocá-la como uma *tag* no método "render" de qualquer outro componente da sua aplicação React, como na Figura 3.6.

Figura 3.4 – Código ComponenteSimples



```
1  import React, { Component } from "react";
2  class ComponenteSimples extends Component {
3    render() {
4      return <p>Hello, DCC!</p>;
5    }
6  }
7  export default ComponenteSimples;
8
```

Figura 3.5 – Código ComponenteSimples em React Native

```
1  import React, { Component } from "react";
2  import { Text } from "react-native";
3
4  class ComponenteSimples extends Component {
5    render() {
6      return <Text>Hello, DCC!</Text>;
7    }
8  }
9  export default ComponenteSimples;
```

Da forma como foi implementado, sempre que esse componente for usado em qualquer lugar da aplicação, ele mostrará o texto “Hello, DCC!”. Mesmo se tratando de um componente exemplo, esse componente não é muito reutilizável, afinal ele mostra apenas um texto fixo e muitas vezes precisamos de um texto diferente para cada contexto.

Figura 3.6 – Exemplo de uso do ComponenteSimples

```
1  import React, { Component } from "react";
2  import ComponenteSimples from
   "./ComponenteSimples";
3
4  class OutroComponente extends Component {
5    render() {
6      return (
7        ...   // outros componentes
8        <ComponenteSimples />
9        ...   // outros componentes
10     );
11   }
12 }
13 export default OutroComponente;
```

Por exemplo, suponha que esse componente fosse utilizado para saudar o usuário após o login na aplicação. Nesse caso ele se tornaria muito mais útil se pudesse escrever “Hello” seguido do nome do usuário. Isso pode ser alcançado utilizando as propriedades do componente, como está detalhado na próxima subseção.

3.2.2 Utilização de propriedades

Pode-se alterar o “ComponenteSimples”, criado na Figura 3.4, para receber o nome que deverá ser usado no título no lugar de “DCC” através de uma propriedade. As propriedades são acessadas através da variável “this.props”. O código do componente alterado está mostrado na Figura 3.7.

Figura 3.7 – ComponenteSimples usando *props*

```
1  import React, { Component } from "react";
2
3  class ComponenteSimples extends Component {
4    render() {
5      return <p>Hello, {this.props.nome}!</p>;
6    }
7  }
8  export default ComponenteSimples;
```

Percebe-se que as variáveis do componente podem ser utilizadas dentro do retorno do método “render” ao serem colocadas entre chaves, como mostrado na linha 5 da Figura 3.7.

A propriedade “nome” é necessária para o funcionamento correto do componente da Figura 3.7, portanto deve ser informada no momento de sua utilização, como mostrado na Figura 3.8, onde o componente vai renderizar um texto “Hello, UFLA!”. Esse tipo de alteração faz com que um mesmo componente possa ser utilizado em diferentes locais da aplicação e isso é uma das características principais do React: Reaproveitamento de código.

Esse mesmo componente seria implementado em React Native apenas trocando a *tag* “p” pelo componente “Text”, assim como foi feito anteriormente no código mostrado na Figura 3.5 e sua utilização é feita da mesma forma que no React (mostrado na Figura 3.8).

3.2.3 Utilização de estados

Um exemplo simples de componente que lida com manipulação de estado está apresentado no código da Figura 3.9, onde é implementado um componente

Figura 3.8 – Utilizando o ComponenteSimples com *props*

```
1  import React, { Component } from "react";
2  import ComponenteSimples from
   "./ComponenteSimples";
3
4  class OutroComponente extends Component {
5    render() {
6      return (
7        ...   // outros componentes
8        <ComponenteSimples nome={"UFLA"}/>
9        ...   // outros componentes
10     );
11   }
12 }
13 export default OutroComponente;
```

chamado “Contador”, composto de um botão e um texto que indica quantas vezes o botão foi clicado.

No construtor da classe deve-se chamar o construtor da classe pai passando como parâmetro as propriedades para que o componente seja criado corretamente (linha 5 da Figura 3.9). Também pode ser definido o objeto “this.state” com os valores iniciais do estado, por exemplo, “cliques”, que iniciou com o valor igual a zero (linha 7). A partir de então, em qualquer método desse componente, o estado pode ser acessado através da variável “this.state”.

Sempre que for preciso alterar um valor armazenado no estado do componente, a função “setState”, herdada da classe Component, deve ser chamada, passando como parâmetro um objeto com as propriedade que foram alteradas, como mostrado na linha 13 da Figura 3.8. Assim como as propriedades, sempre que o

estado é alterado o componente sabe que uma nova renderização será necessária e é isto que difere as varáveis de estado de outras variáveis da classe.

Figura 3.9 – Contador: Exemplo de componente que utiliza *state*

```
1  import React, { Component } from "react";
2
3  class Contador extends Component {
4    constructor(props) {
5      super(props);
6      this.state = {
7        cliques: 0,
8      };
9    }
10
11   incrementar() {
12     let quantidade = this.state.cliques + 1;
13     this.setState({ cliques: quantidade });
14   }
15
16   render() {
17     return (
18       <div>
19         <button
20           onClick={() => this.incrementar()}
21         >
22           INCREMENTAR
23         </button>
24         <p>{this.state.cliques}</p>
25       </div>
26     );
27   }
28 }
29 export default Contador;
```

Para que esse componente funcionasse no React Native, seria necessário trocar as *tags* “div” pelo componente “View”, “button” pelo componente “Button”, “p” pelo componente “Text” como mostrado na Figura 3.10. Outra diferença é que o método “onClick” foi substituído pelo método “onPress”, seu equivalente no RN. O restante do componente continua o mesmo.

Figura 3.10 – Contador em React Native

```
1  import React, { Component } from 'react';
2  import { Text, View, Button } from
   'react-native';
3
4  class Contador extends Component {
5    ...
6    render(){
7      return (
8        <View>
9          <Button onPress={this.incrementar}/>
10         <Text>
11           {this.state.cliques}
12         </Text>
13       </View>
14     );
15   }
16 }
17 export default Contador;
```

Através desses exemplos, pode-se perceber o quanto o React e React Native têm um funcionamento parecido, permitindo que se mantenha a mesma estrutura do componente, mas alterando as *tags* que no React Native são um ponte entre os componentes nativos Android e iOS. Isso proporcionou uma aceleração no de-

senvolvimento, já que as habilidades aprendidas no desenvolvimento do sistema web puderam ser utilizadas nos aplicativos móveis.

3.2.4 Utilização do Redux

Para exemplificar o funcionamento do Redux, será mostrada uma implementação de um componente que pode incrementar ou decrementar uma variável que está armazenada no estado global da aplicação. Existem 3 conceitos principais do Redux que são essenciais:

1. *Store*: A estrutura que armazena o estado global. Deve existir apenas uma *store* em cada aplicação.

2. *Actions*: As ações que podem ser feitas na *store*, já que ela não pode ser diretamente alterada.

3. *Reducers*: Definem o que será feito na *store* para cada *action* disparada.

Primeiramente, define-se as *actions* que poderão ser realizadas no estado global. No exemplo da Figura 3.11 foram definidos 3 tipos de ações: incrementar, decrementar ou alterar o valor da variável para um valor recebido por parâmetro. Pode ser percebido que a estrutura de uma *action* é simples, consistindo de um *type*, que é uma string que identifica aquela ação e, se necessário, um *payload* que contêm os valores necessários para aquela ação. As duas primeiras ações definidas não precisaram de *payloads*, pois apenas vão aumentar ou diminuir em 1 o valor que estiver na variável.

A próxima etapa é a definição do *reducer* para essas ações. O *reducer* nada mais é que uma função que recebe o estado global atual e a ação que foi disparada e retorna como o estado global deve ficar depois da ação. É uma boa prática definí-lo com um *switch* que define o que será feito de acordo com o *type* da ação recebida, como na Figura 3.12. Para cada tipo de ação definida anteriormente, o *contadorReducer* manipulará o estado global de uma forma diferente.

Figura 3.11 – Ações Redux

```
1  export const incrementarAction = () => ({
2    type: "INCREMENTAR",
3  });
4
5  export const decrementarAction = () => ({
6    type: "DECREMENTAR",
7  });
8
9  export const mudarValorAction = (valor) => ({
10   type: "MUDAR_VALOR",
11   payload: valor,
12 });
13
```

Finalmente, cria-se a *store* dessa aplicação (Figura 3.13). Foi utilizada a função "combineReducers", onde seriam colocados os demais *reducers* da aplicação, caso existissem. O próximo passo é envolver o componente raiz da aplicação com o componente "Provider" do pacote "react-redux" (Figura 3.14).

Após esse processo, pode-se utilizar o Redux nos componentes da aplicação. Para isso, utiliza-se a função *connect*, do react-redux, que recebe dois parâmetros:

1. *mapStateToProps*: Função onde pode ser selecionado o que será passado do estado global para as *props* do componente.

2. *mapDispatchToProps*: Função onde são determinadas quais *actions* serão passadas para o componente.

Ao utilizar a função *connect*, os elementos selecionados nas funções *mapStateToProps* e *mapDispatchToProps* são passados para as propriedades do compo-

Figura 3.12 – Redux *Reducer*

```
const estadoInicial = 0;

export function contadorReducer(
  state = estadoInicial,
  action
) {
  switch (action.type) {
    case "INCREMENTAR":
      return state + 1;
    case "DECREMENTAR":
      return state - 1;
    case "MUDAR_VALOR":
      return action.payload;
    default:
      return state;
  }
}
```

nente. Um exemplo de uso da função *connect* pode ser encontrada na Figura 3.15, onde o componente "Contador" se conectou ao Redux e recebeu o valor de "contador" que estava armazenado no estado global, bem como as funções "incrementar", "decrementar" e "reset" que disparam *actions*.

3.2.5 Implementação da aplicação web

Nessa etapa foram implementados os componentes planejados para a aplicação web na etapa de componentização (subseção 3.1.3). Como a aplicação contém muitos componentes, serão apresentados na seção de resultados apenas os componentes que mais impactaram na experiência do estagiário.

Figura 3.13 – Redux *Store*

```
import {
  combineReducers,
  createStore
} from "redux";
import contadorReducer from "./contadorReducer";


const rootReducer = combineReducers({
  contador: contadorReducer,
});

const store = createStore(rootReducer);
export default store;

```

Figura 3.14 – Redux *Provider*

```
import React from 'react';
import { Provider } from 'react-redux';
import store from './redux/store';

function App() {
  return (
    <Provider store={store}>
      // restante do app
    </Provider>
  );
}
export default App;
```

Figura 3.15 – Exemplo de uso do Redux

```
import React, { Component } from "react";
import { connect } from 'react-redux';

class Contador extends Component {
  render() {
    ...
  }
}

const mapStateToProps = (globalState) => {
  return { contador: globalState.contador }
}

const mapDispatchToProps = (dispatch) => {
  return {
    incrementar: () =>
      dispatch(incrementarAction()),
    decrementar: () =>
      dispatch(decrementarAction()),
    reset: () =>
      dispatch(mudarValorAction(0)),
  };
};

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(Contador);

```

Como parte das atividades relacionadas ao desenvolvimento da aplicação web podem ser citados a otimização de componentes para a economia de tempo

de usuários e programadores. Além disso, também foram feitas buscas por alternativas à apresentação de grande volume de conteúdo de modo a torná-la menos massante para o usuário, mas ainda operacionalmente viável.

3.2.6 Implementação da aplicação para dispositivos móveis

O design e implementação de aplicações para dispositivos móveis exigem processos muito diferentes dos utilizados nas aplicações para *web/desktop*, pois geralmente são dispositivos com uma tela menor, menos poder de processamento, menos memória e bateria limitada. Deve-se ficar atento a isso durante o desenvolvimento desse tipo de aplicação para que o usuário não tenha uma experiência ruim, como travamentos, alto consumo de bateria e excesso de informação na tela.

Nessa etapa, foram implementados os componentes planejados para a aplicação móvel na etapa de componentização (subseção 3.1.3). Como a aplicação contém muitos componentes, serão apresentados na seção de resultados os componentes que mais impactaram experiência do estagiário.

Como tarefas desempenhadas durante o estágio relacionadas ao desenvolvimento da aplicação móvel, destacam-se aquelas relacionadas à busca por soluções para a apresentação de mapas com informações importantes para o usuário, de modo legível e sem sobrecarregar a aplicação. Cogitou-se até na exclusão de algumas dessas telas que continham mapas, mas, no final, a solução encontrada mostrou-se satisfatória e foi mantida.

3.3 Testes

Como o prazo para conclusão do projeto estava próximo e o foco era o desenvolvimento de um produto mínimo viável, o estagiário não presenciou a fase de desenvolvimento de testes de forma completa, tendo participado apenas dos testes do sistema que eram feitos antes da liberação de cada nova funcionalidade.

Para a realização desses testes, os ambientes de produção dos sistemas em funcionamento foram clonados e apenas os dados sigilosos foram alterados, criando um ambiente que simulasse o uso real das aplicações. Esse ambiente de testes foi utilizado durante toda a implementação e foi nele que algumas falhas de performance e design foram notadas.

Após a liberação de uma nova funcionalidade, o desenvolvedor é responsável por contatar os funcionários responsáveis pelo sistema na sede da empresa para explicar seu funcionamento e testá-las em conjunto no ambiente de testes onde era avaliado o funcionamento das novas funcionalidades em vários casos de teste. Caso a funcionalidade fosse aprovada, ela seria liberada na próxima atualização do sistemas, que geralmente ocorria semanalmente.

Além desse ambiente de testes, foi disponibilizado um usuário no ambiente de produção para que a equipe pudesse se certificar que tudo tinha ocorrido como planejado após a liberação das atualizações.

4 RESULTADOS

Como resultado global das atividades desenvolvidas durante o estágio pode-se destacar que o projeto foi entregue no prazo estabelecido e pôde ser apresentado pela empresa na 22ª edição da Fenatran.

Figura 4.1 – Estande Delta Fleet na 22ª Fenatran

A apresentação na feira foi feita em um estande com televisões com tecnologia *touch screen*, onde os vendedores puderam navegar pelo sistema web em tempo real para apresentá-lo a possíveis clientes. Os testes realizados anteriormente foram essenciais para evitar imprevistos durante a apresentação.

A aplicação *mobile* também foi apresentada na feira em vários *tablets* espalhados pelo estande, permitindo que os feirantes conhecessem o sistema navegando livremente por ele. Isso exige mais cuidado que uma apresentação guiada, já que não se pode prever o caminho que o cliente irá tomar. O sistema se mostrou robusto e não apresentou falhas.

Pode-se dizer que, tanto o projeto quanto sua apresentação, obtiveram sucesso, o que culminou no aumento do número de veículos cadastrados em três vezes a quantidade anteriormente cadastrada. Essa nova carga de dados no servidor fez com que a empresa tivesse que realizar mudanças em seu funcionamento interno, com a contratação de funcionários e a contratação de servidores mais robustos com maior confiabilidade.

Após a feira, mais funcionalidades foram desenvolvidas para o Delta Fleet. No final do projeto cogitou-se a possibilidade de o novo sistema substituir completamente o Delta Sat (que é um sistema menor, se comparado ao Delta Assist).

O resultado individual das atividades desempenhadas pelo estagiário será apresentado nas próximas subseções.

4.1 Componentização das telas

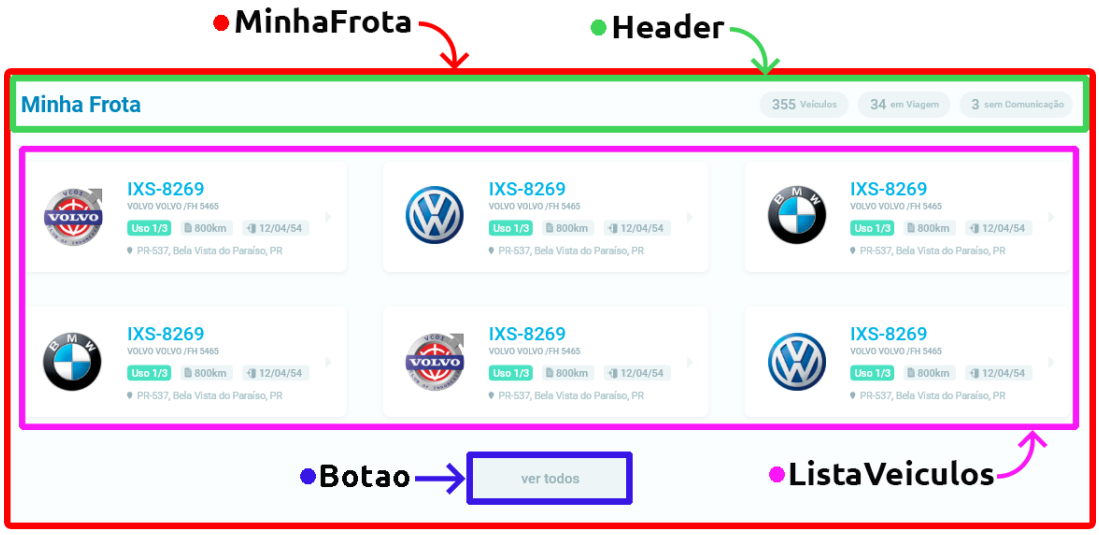
A etapa de componentização se mostrou trabalhosa, mas também muito recompensadora. O planejamento dos componentes antes de iniciar sua implementação possibilitou uma melhor organização dos dados e, também, mostrou algumas necessidades, como o uso de um gerenciador de estados globais como o Redux.

A seção “Minha Frota”, por exemplo, mostrada na Figura 3.1 foi dividida em vários componentes, como mostrado na Figura 4.2. O componente “MinhaFrota” é considerado o componente-pai dos demais pois ele os contém dentro de si.

Nessa divisão já é possível definir algumas características dos estados e propriedades desses componentes. O “ListaVeiculos” deve ter acesso a uma lista de todos os veículos e o “Header” deve ter acesso a alguns dados sobre essa mesma lista, como quantos veículos existem nela e quantos estão em viagem. Como ambos acessam dados da mesma lista, seria um gasto desnecessário de rede, processamento e memória buscá-la duas vezes no servidor, então, fazemos com que o

componente-pai “MinhaFrota” busque a lista e envie os dados necessários para “ListaVeiculos” e “Header” através de *props*.

Figura 4.2 – Componentização da sessão “Minha Frota”



Os componentes definidos também foram divididos em mais componentes. “ListaVeiculos”, por exemplo, é composto de vários itens, cada um representando um veículo do cliente, e por isso, foi dividido como mostra a Figura 4.3. Dessa forma, cada componente fica encarregado de uma função específica, abstraindo-se do restante da aplicação, e isso simplifica a implementação.

Figura 4.3 – Componentização “ListaVeiculos”

4.2 Utilização do Redux

Como justificado na subseção 3.2.4, o Redux ajudou a evitar que propriedades fossem passadas muitos níveis abaixo na hierarquia do sistema e também ajudou a diminuir a quantidade de acessos ao servidor. Seu funcionamento será descrito a seguir.

Quando a aplicação é iniciada, os componentes verificam se a lista de veículos existe no estado global e, caso não exista, é enviada uma requisição ao servidor e uma variável, indicando que a frota de veículos está sendo carregada, é armazenada com o valor *true* no Redux. Enquanto a aplicação não receber a resposta do servidor, todos os componentes que precisarem da frota de veículos vão verificar o valor dessa variável e, caso seja *true*, vão aguardar ao invés de enviar a requisição novamente. Quando a resposta do servidor chegar, o valor da variável que indica o carregamento é passado para *false* e a lista de veículos é disponibilizada. Desse forma, os dados sobre a frota de veículos do cliente é buscada apenas uma vez no servidor e utilizada por todos os componentes simultaneamente.

Vale ressaltar que não é recomendada a utilização do Redux para todos os dados da aplicação, já que isso pode dificultar a manutenção do código e, também, faz com que os dados fiquem carregados na memória o tempo todo, o que pode sobrecarregar o dispositivo do usuário. Por isso, ele foi utilizado apenas para armazenar a lista de veículos e alguns dados da sessão do usuário nesse projeto.

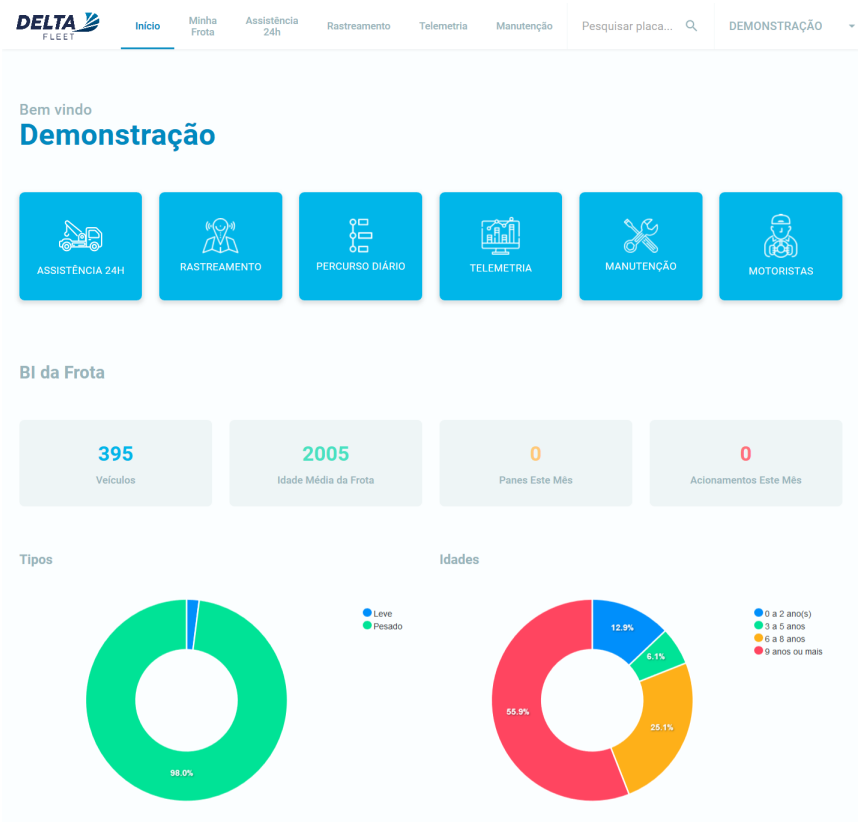
4.3 Implementação da Aplicação Web

As principais telas dessa aplicação estão destacadas nas Figuras 4.4, 4.5 e 4.6, que são imagens reais do sistema em funcionamento, utilizando o usuário de demonstração.

Pode-se notar, na Figura 4.4, a alteração do desenho original apresentado na Figura 3.1. Isso se deve ao fato de o desenvolvimento do sistema não ser linear,

já que várias vezes o design teve de ser repensado para atender a alguma exigência ou resolver um problema que só foi percebido durante a implementação.

Figura 4.4 – Tela inicial do sistema web



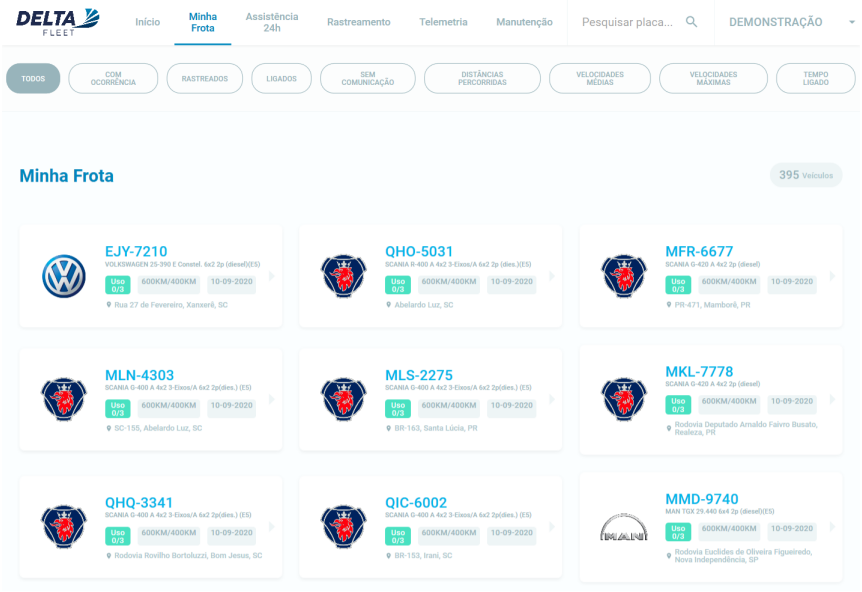
A Figura 4.6 é um exemplo de tela que foi reaproveitada do sistema antigo e passou apenas por uma mudança de design. Esse reaproveitamento de telas foi essencial para garantir a entrega no prazo estipulado.

A seguir serão detalhados alguns componentes que tiveram impacto na experiência do estagiário durante o desenvolvimento da aplicação web.

4.3.1 Pesquisa de veículos

No desenho da aplicação existe um campo de texto na barra superior do *website* para a pesquisa de veículo, um dos elementos mostrados na Figura 3.3.

Figura 4.5 – Tela da frota no sistema web



O funcionamento planejado para esse componente era que o usuário digitaria a placa de um veículo (parcialmente ou completamente) e, ao clicar “enter”, seria apresentada uma nova tela com o resultado da pesquisa.

Porém, o estagiário sugeriu fazer o componente ser mais dinâmico e poupar tempo do usuário e dos programadores. Poderíamos criar um componente que filtrasse a lista de veículos em tempo real enquanto o usuário está digitando e mostrasse as opções logo abaixo do campo de pesquisa, eliminando a necessidade de uma nova página para mostrar os resultados da busca. O componente “Autocomplete”, que é parecido com o que foi idealizado, foi encontrado na biblioteca Material UI (9). Ele é mostrado na Figura 4.7.

A sugestão foi aprovada, desde que fosse possível personalizar as opções mostradas para ficarem com um design mais sofisticado, seguindo os padrões da empresa. O componente nativamente permite a customização das opções, então um protótipo foi criado onde as opções eram baseadas no componente “ItemLista-

Figura 4.6 – Tela do mapa no sistema web

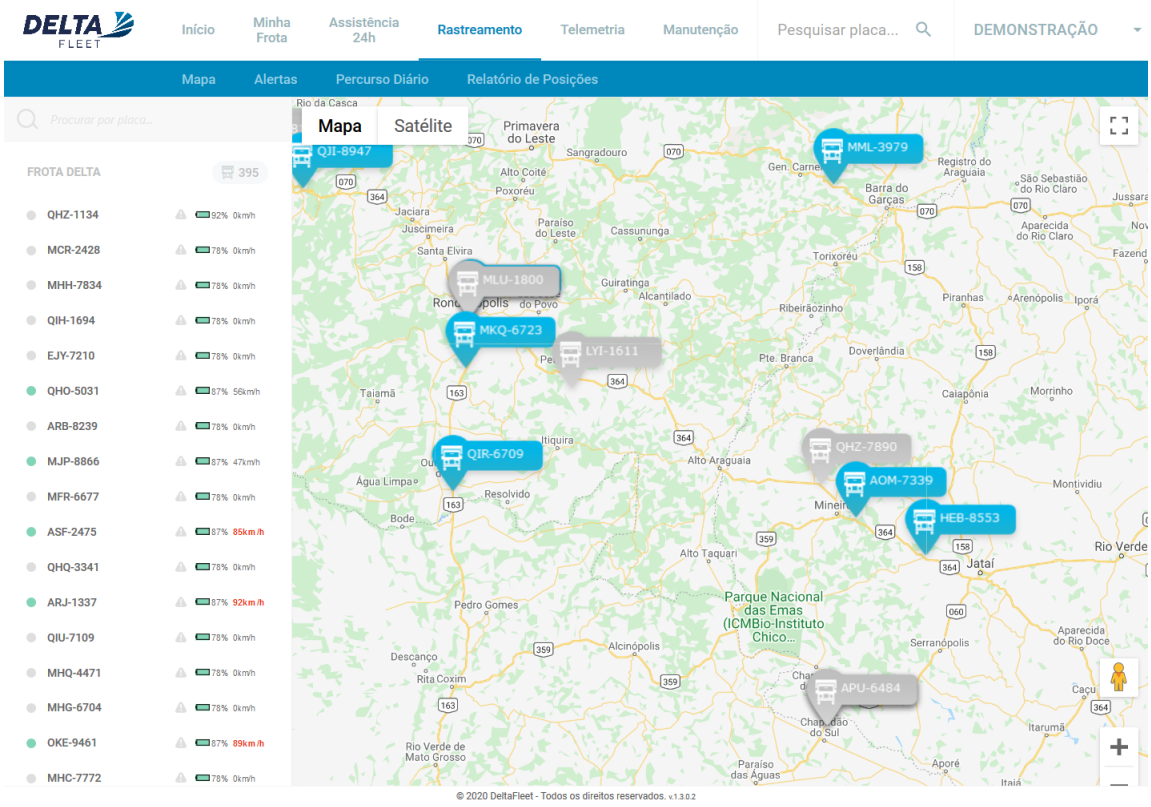
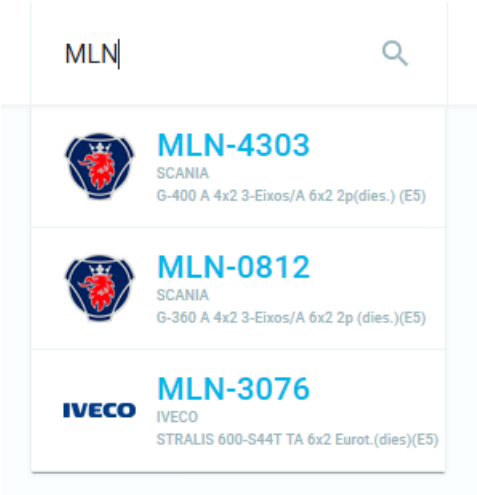


Figura 4.7 – Componente “Autocomplete” do Material UI

Veiculos” (Figura 4.3), resultando no componente mostrado na Figura 4.8, que foi aprovado pela equipe e passou a ser utilizado na aplicação.

Figura 4.8 – Componente “Autocomplete” onde as opções foram customizadas e mostram dados sobre um veículo (placa, marca, modelo e imagem da marca)



4.3.2 Lista de veículos com rolagem infinita

Uma das telas desafiadoras da aplicação foi a tela que mostra todos os veículos do cliente (Figura 4.5), com algumas opções de filtragem dessa lista, como mostrar apenas os veículos que estão ligados no momento ou os que estão sem comunicação.

Inicialmente, pareceu um componente simples e de fácil implementação, porém, quando esse componente foi criado, notou-se um grande consumo de memória e processamento, principalmente quando o cliente possuía uma grande quantidade de veículos cadastrados (alguns possuem mais de 500 veículos). Depois de uma investigação, foi constatado que a causa desse alto consumo era a renderização de muitos componentes ao mesmo tempo na tela. Foram estudadas duas soluções para esse problema:

- Dividir a lista em várias páginas

- Utilizar a técnica de rolagem infinita, onde apenas parte dos dados é mostrado na tela e quando o usuário rolar até o fim da página mais dados serão

carregados. Essa técnica é muito utilizada em redes sociais, como Facebook e Instagram.

Pensando no conforto do usuário, a equipe concluiu que a rolagem infinita seria mais eficiente (pois poupa o usuário de ter que ficar clicando para passar páginas) e elegante. A implementação dessa técnica foi desafiadora, pois não é disponibilizada de forma nativa no React. Para alcançar o resultado desejado, foi implementado um *Event Listener* para a rolagem, que é uma função que será chamada toda vez que um evento específico acontecer (nesse caso, sempre que houver a rolagem da página). Sempre que a posição Y da rolagem for igual ao tamanho da página, mais dados serão carregados na tela.

Embora essa abordagem ainda apresente a desvantagem de consumo excessivo de memória caso o usuário chegue até o fim da lista, a equipe concluiu que isso acontecerá muito raramente, já que se o usuário quiser ver os dados de um veículo específico ele utilizará o campo de pesquisa disponível no topo da aplicação.

4.4 Implementação da aplicação para dispositivos móveis

O aplicativo se tornou um grande diferencial do novo sistema e atraiu a atenção de muitas pessoas durante a exposição na Fenatran tanto pelo seu design quanto por permitir analisar dados da frota de veículos em tempo real em um dispositivo móvel. As Figuras 4.9, 4.10, 4.12, 4.13 mostram as principais telas da aplicação móvel.

A Figura 4.9 mostra a tela principal do aplicativo móvel, onde o cliente tem acesso às principais funcionalidade do aplicativo e um resumo sobre sua frota de veículos. A Figura 4.10 mostra a tela onde o cliente pode pesquisar pela placa de algum veículo de sua frota e, caso ele pressione em algum veículo, será redirecionado para um tela que mostra dados específicos sobre esse veículo.

Figura 4.9 – Tela principal do app

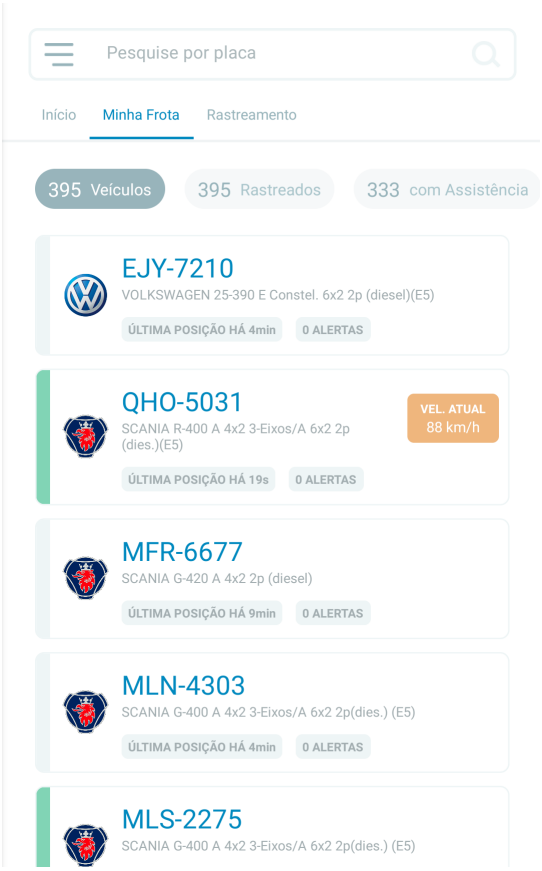
A seguir serão detalhados os componentes que mais impactaram o desenvolvimento da aplicação para dispositivos móveis.

4.4.1 Telas com mapa escondido

A tela com mapa escondido é um design que foi inspirado em uma das telas do aplicativo Uber[1], mostrada na Figura 4.11, que consiste em um mapa mostrando posições de interesse do usuário e um conteúdo na parte inferior. Caso o usuário clique no conteúdo da parte inferior, esse conteúdo sobe e esconde o mapa.

[1] https://www.uber.com/br/pt-br/

Figura 4.10 – Tela com lista de todos os veículos do cliente no app



Os requisitos para o funcionamento desejado dessa tela no projeto foram: inicialmente a tela deveria mostrar os dados de interesse do usuário, como na Figura 4.12 onde são mostrados dados sobre um veículo fictício no conteúdo principal da tela. Caso o usuário clique no botão “No mapa” ou na pequena seta que está no topo da tela, o conteúdo deveria descer e revelar um mapa com as posições referentes àquela tela, como mostrado na Figura 4.13 onde é mostrada a posição atual do veículo, o secundário da tela.

Esse movimento no conteúdo principal da tela foi alcançado utilizando o sistema de animação de componentes nativo do React Native (29), uma experiência totalmente nova e desafiadora para o estagiário, pois a animação deveria controlar

Figura 4.11 – Tela aplicativo Uber

Fonte: Aplicativo Uber

mais de um componente. Na animação que esconde o mapa, por exemplo, o conteúdo principal inteiro desliza para cima, o botão com a seta gira 180º para que a seta fique apontada para baixo e o *background* branco expande verticalmente para cobrir toda a tela.

Esse design foi utilizado em várias telas do aplicativo e se tornou uma das características principais do app Delta Fleet. As informações principais sobre um determinado assunto e um mapa na mesma tela conseguem ser mostrados sem que haja sobrecarga da interface de informações. Além disso, foi um design que impressionou os clientes.

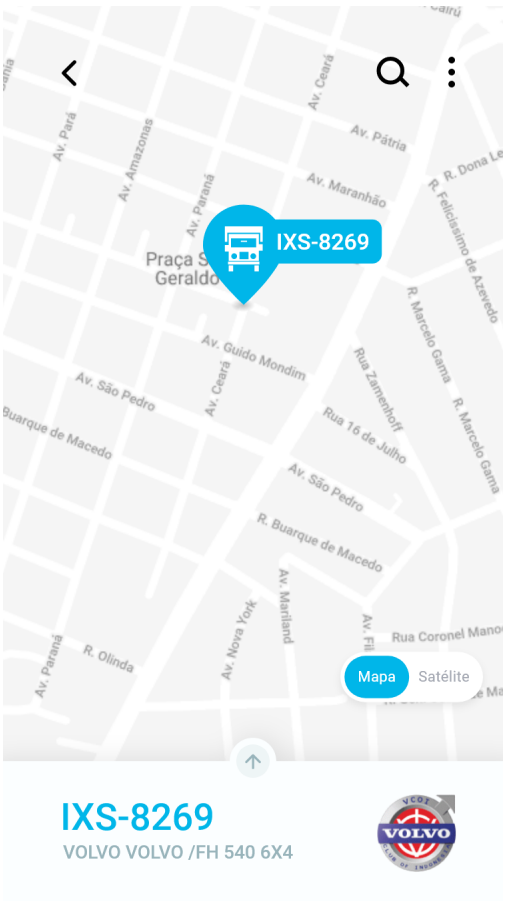
Figura 4.12 – Tela inspirada no app Uber mostrado na Figura 4.11

4.4.2 Mapa com todos os veículos

A tela mostrada na Figura 4.14 foi uma das telas onde as limitações dos dispositivos móveis fizeram toda a diferença. Trata-se de uma tela simples que mostra apenas um mapa com as posições de todos os veículos rastreados do cliente, mas depois de implementada, notou-se um extrema lentidão e travamentos devido à grande quantidade de marcadores no mapa.

A equipe cogitou remover essa tela do aplicativo, mas depois de pesquisar sobre esse problema, foram encontradas duas formas de melhorar a performance e a visualização do mapa:

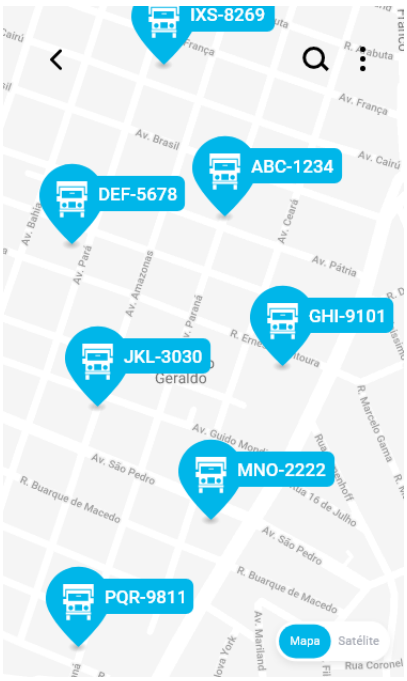
Figura 4.13 – Tela da Figura 4.12 porém com o mapa à mostra



- Os marcadores contém uma propriedade chamada “tracksViewChanges” que, enquanto seu valor for *true*, faz o marcador monitorar alterações nos seus dados (como suas coordenadas). No caso, o componente do marcador só sofreria alterações nos seus dados caso fossem buscadas novas posições no servidor. O restante do tempo, essa propriedade poderia ficar com o valor *false*, poupando muito processamento.

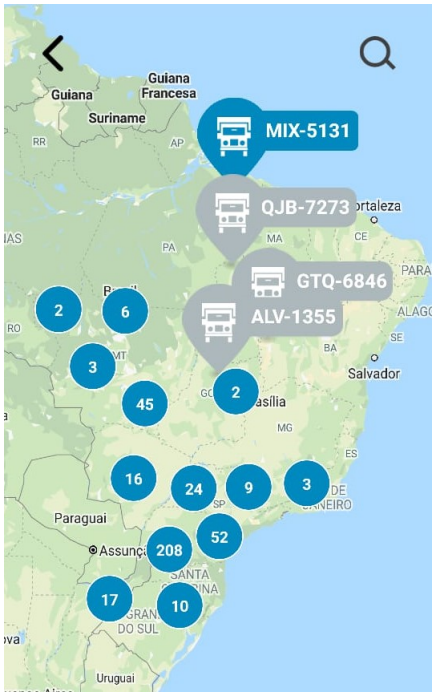
- Foi encontrado um pacote para o React Native chamado “react-native-cluster-map”, que cria grupos de marcadores quando estes estão muito próximos, de forma a reduzir a quantidade de marcadores mostrados ao mesmo tempo na

Figura 4.14 – Desenho da tela com todos os veículos



tela, como mostrado na Figura 4.15. Isso resultou em uma melhoria na performance e no design da tela. Ao clicar em um grupo de marcadores, o mapa é ampliado e os marcadores são mostrados individualmente.

Figura 4.15 – Mapa com grupos de marcadores



5 CONCLUSÃO

O estágio foi uma peça muito importante na minha formação, onde pude realmente pôr em prática tudo que aprendi na faculdade. Até mesmo assuntos que muitas vezes consideramos secundários durante as aulas fazem toda a diferença em um ambiente de trabalho real.

Um grande exemplo disso são as matérias onde estudamos grandes estruturas de dados. O desenvolvimento dessas estruturas do zero não é muito comum atualmente no mercado de trabalho, já que existem implementações otimizadas delas disponíveis na internet, mas essas matérias nos dão uma aula gigante sobre a importância de um código eficiente e sem gasto desnecessário de processamento, mesmo não sendo esse o foco da disciplina. Na Delta, trabalhamos com milhares de veículos e milhões de registros das suas posições, onde todo aumento na complexidade do tratamento desses dados pode impactar muito na experiência do usuário.

Além de aplicar a teoria aprendida na faculdade, o estagiário teve a oportunidade de trabalhar com pessoas muito experientes, como seu mentor Marzon Castilho que tem *background* em grandes empresas como a Xerox e pôde me passar muito do conhecimento adquirido durante sua carreira como programador e empreendedor.

Outra marca importante que a Delta deixou foram os novos amigos que o estagiário conseguiu. Mesmo conhecendo grande parte da equipe apenas nas diversas chamadas do Skype ou conversas do Slack, o estagiário fez novas amizades que espera levar para a vida toda. Por isso, foi uma felicidade enorme o estagiário ter tido a oportunidade de conhecer a equipe pessoalmente em Porto Alegre, uma viagem que vai ficar para sempre em sua memória (também foi a primeira vez que o estagiário andou de avião).

REFERÊNCIAS

1 DUNDER, K. *Recém-formados sofrem para conquistar o primeiro emprego*. 2020. Disponível em: <https://noticias.r7.com/educacao/recem-formados-sofrem-para-conquistar-o-primeiro-emprego-30012020>.

2 DALL'AGNOL, L. *Mais jovens recém-formados ficam sem emprego no país*. 2020. Disponível em: <https://agora.folha.uol.com.br/grana/2019/09/mais-jovens-recem-formados-ficam-sem-emprego-no-pais.shtml>.

3 GHEZZI, C.; JAZAYERI, M.; MANDRIOLI, D. Fundamentals of software engineering. Prentice Hall Englewood Cliffs, 1991.

4 ALSHAMRANI, A.; BAHATTAB, A. A comparison between three sdlc models waterfall model, spiral model, and incremental/iterative model. *International Journal of Computer Science Issues (IJCSI)*, International Journal of Computer Science Issues (IJCSI), v. 12, n. 1, p. 106, 2015.

5 RIES, E. *A startup enxuta*. LEYA BRASIL, 2012. ISBN 9788581780139. Disponível em: <https://books.google.com.br/books?id=vLiUj1h5fhkC>.

6 COMPUTAçãO, S. B. de. *Interação Humano-Computados*. 2020. Disponível em: <https://www.sbc.org.br/14-comissoes/390-interacao-humano-computador>.

7 NORMAN, D. A. *The Design of Everyday Things*. 2. ed. New York: Basic Books, 2013. ISBN 978-0-465-05065-9.

8 GOOGLE. *Material Design*. 2020. Disponível em: <https://design.google/library/visit-materialio/>.

9 UI, M. *Documentação Material UI*. 2020. Disponível em: <https://material-ui.com/pt/>.

10 FIELDING, R. T. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doctoral dissertation) — University of California, Irvine, 2000. Disponível em: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

11 ALURA. *O que é front-end e back-end?* 2020. Disponível em: <https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end>.

12 WEBOPEDIA. *What is API?* Visitado em 2020-05-04. Disponível em: <https://www.webopedia.com/TERM/A/API.html>.

13 COMPUTERWORLD. *Application Programming Interface*. 2020. Disponível em: <https://www.webopedia.com/TERM/A/API.html>.

14 NETWORK, M. D. *Javascript*. 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>.

15 TIOBE. *TIOBE Index*. 2020. Disponível em: <https://www.tiobe.com/tiobe-index/>.

16 FACEBOOK. *React Documentation*. 2020. Disponível em: <https://pt-br.reactjs.org/docs/react-api.html>.

17 FACEBOOK. *Componentes e Props*. 2020. Disponível em: <https://pt-br.reactjs.org/docs/components-and-props.html>.

18 FACEBOOK. *State dos Componentes*. 2020. Disponível em: <https://pt-br.reactjs.org/docs/faq-state.html#what-does-setstate-do>.

19 FACEBOOK. *Estado e Ciclo de Vida*. 2020. Disponível em: <https://pt-br.reactjs.org/docs/state-and-lifecycle.html>.

20 FACEBOOK. *React Component*. 2020. Disponível em: <https://pt-br.reactjs.org/docs/react-component.html>.

21 SIMFORM, R. S. *React Native vs Ionic: Comparing performance, User Experience and much more!* 2020. Disponível em: <https://www.simform.com/react-native-vs-ionic/>.

22 LIFEWIRE. *Native Apps vs. Web Apps*. 2020. Disponível em: <https://www.lifewire.com/native-apps-vs-web-apps-2373133>.

23 FACEBOOK. *Core Components and Native Components*. 2020. Disponível em: <https://reactnative.dev/docs/intro-react-native-components>.

24 FACEBOOK. *React Native*. 2020. Disponível em: <https://reactnative.dev/>.

25 CLARK, D. A. e A. *Redux*. 2020. Disponível em: <https://redux.js.org/>.

26 PHP. *History of PHP*. 2020. Disponível em: <https://www.php.net/manual/en/history.php.php>.

27 PHP. *What can PHP do?* 2020. Disponível em: <https://www.php.net/manual/en/intro-whatcando.php>.

28 ADOBE. *Adobe XD*. 2020. Disponível em: <https://www.adobe.com/br/products/xd.html>.

29 FACEBOOK. *React Native Animations*. 2020. Disponível em: <https://reactnative.dev/docs/animations>.