



**Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"**

JOÃO OTÁVIO DA SILVA

SISTEMA PARA CONTROLE FINANCEIRO DE VEÍCULOS FRETEIROS

**Assis/SP
2022**



**Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"**

JOÃO OTÁVIO DA SILVA

SISTEMA PARA CONTROLE FINANCEIRO DE VEÍCULOS FRETEIROS

Projeto de pesquisa apresentado ao curso de Análise de Sistemas do Instituto Municipal de Ensino Superior de Assis – IMESA e a Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando(a): João Otávio da Silva

Orientador(a): Dr. Almir Rogério Camolesi

**Assis/SP
2022**

Dados Internacionais de Catalogação na Publicação (CIP)

S586s Silva, João Otávio da.

Sistema para controle financeiro de veículos freteiros / João Otávio da Silva – Assis, SP: FEMA, 2022.

43 f.

Trabalho de Conclusão de Curso (Graduação) – Fundação Educacional do Município de Assis – FEMA, curso de Análise e Desenvolvimento de Sistema, Assis, 2022.

Orientador: Prof. Dr. Almir Rogério Camolesi.

1. Motorista. 2. Transportadora. 3. Aplicativo Móvel. 4. Back-End. 5. Web Service. 6. Web Site. I. Título.

CDD 001.61

Biblioteca da FEMA

SISTEMA PARA CONTROLE FINANCEIRO DE VEÍCULOS FRETEIROS

JOÃO OTÁVIO DA SILVA

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, avaliado pela seguinte comissão examinadora:

Orientador: _____
Dr. Almir Rogério Camolesi

Examinador: _____
Me. Guilherme de Cleve Farto

DEDICATÓRIA

Dedico este trabalho primeiramente a Deus, a minha família que me apoiou e apoia nas minhas decisões, e a todos meus amigos que me incentivam a nunca desistir dos meus objetivos.

AGRADECIMENTOS

Agradeço mais uma vez a Deus por sempre estar comigo, dando forças para continuar e me ajudando sempre tudo no melhor momento, para que eu consiga realizar todos meus objetivos. Ao Professor, Almir Camolesi, pela orientação e ensinamentos. Aos meus amigos, Hudyson, Leonardo, Haniel, Fernando, e os demais colegas de sala e a todos que colaboraram de forma direta ou indiretamente, na execução deste trabalho. Agradeço também aos meus pais, Maycon e Milene e a minha namorada Larissa que me deram forças e sempre me apoiaram dando motivações, para a execução do projeto.

RESUMO

O objetivo principal deste trabalho é criar um aplicativo móvel e um web site para o controle de fretes realizados pela frota das transportadoras, garantindo um melhor controle dos lucros e despesas durante as viagens. A proposta do aplicativo móvel é que os motoristas cadastrados possam estar abrindo viagens e lançando as despesas, valores dos fretes, quilometragem rodada, origem e destino da viagem. Já o web site será usado pela transportadora que terá as funcionalidades de cadastro de veículos, cadastro de motoristas e visualizar relatórios das viagens realizadas.

Para o desenvolvimento do aplicativo móvel será utilizado o framework Ionic com Angular, que com um único código fonte traduz e compila para múltiplas plataformas como o Android e IOS. No desenvolvimento do web site será utilizado o framework Angular, e para o desenvolvimento do back-end será criado uma web service com Java e Spring Boot que ficará centralizado toda a regra do negócio do sistema, para que o aplicativo móvel e o web site possam consumir a API.

Palavras-chave: Fretes, Motorista, Transportadora, Aplicativo Móvel, Back-End, Web Service, Web Site.

ABSTRACT

The main part of this work is a mobile application and a web site for the control of freight carried out by the carriers' fleets, having during the trips the best control of profits and expenses. The proposal of the mobile application is that registered professionals can be opening trips, freight values, mileage, origin and destination of the trip. The site will be used by the operator, which will have the functions of registering vehicles, registering drivers and viewing travel reports.

For the development of the mobile application, the Ionic framework with Angular will be used, which can translate and compile a single code to facilitate the use of Android and IOS. The web site will be used in the Angular framework, and for the development of the back-end, a web with Java and Spring Boot will be created that will be centralized for the business rule of the system application, so that the mobile service and the web site can consume in developing an API.

Keywords: Freight, Driver, Carrier, Mobile, Back-End, Web Service, Web Site.

LISTA DE ILUSTRAÇÕES

Figura 1: Estrutura de camadas.....	19
Figura 2: Arquitetura Cliente Servidor.....	20
Figura 3: Diagrama de caso de uso ator do tipo motorista	22
Figura 4: Diagrama de caso de uso ator do tipo transportadora.....	23
Figura 5: Diagrama de Entidade e Relacionamento	24
Figura 6: Diagrama de Classes	25
Figura 7: Diagrama de Classes Enumeradas	26
Figura 8: Protótipo da página inicial do aplicativo móvel	26
Figura 9: Protótipo da tela de viagem em andamento	27
Figura 10: Protótipos das telas de lançamentos	28
Figura 11: Protótipos da tela de relatório da viagem	28
Figura 12: Business Model Canvas	29
Figura 13: Estrutura de arquivos do projeto da API.....	30
Figura 14: Classe de Modelo Transportadora.java.....	31
Figura 15: Classe TransportadoraRepository.java.....	32
Figura 16: Classe TipoVeiculoService.java.....	32
Figura 17: Classe TipoVeiculoController.java.....	33
Figura 18: Interface do Swagger-ui.....	34
Figura 19: Estrutura do projeto Ionic.....	35
Figura 20: Serviço de Viagens.....	36
Figura 21: Serviço responsável por gerenciar o token armazenado.....	37
Figura 22: Função responsável por fazer o login.....	38
Figura 23: Tela de Login	38
Figura 24: Painel da Aplicação na Heroku.....	39

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
APK	Android Application Pack
CLI	Command Line Interface
CSS	Cascading Style Sheet
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
MVC	Model View Controller
REST	Representational State Transfer
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	Structured Query Language
WEB	World Wide Web

SUMÁRIO

1. INTRODUÇÃO	12
1.1. OBJETIVOS	12
1.2. JUSTIFICATIVA	13
1.3. MOTIVAÇÃO	13
1.4. PERSPECTIVA DE CONTRIBUIÇÃO	13
1.5. METODOLOGIA	13
1.6. RECURSOS NECESSÁRIOS	14
1.7. ESTRUTURA DO TRABALHO	14
2. REVISÃO BIBLIOGRÁFICA	16
2.1 DESENVOLVIMENTO MÓVEL	16
2.1.1 Android	16
2.1.2 IOS	16
2.1.3 Tecnologias Para Desenvolvimento Móvel	17
2.2 DESENVOLVIMENTO WEB	18
2.3 DESENVOLVIMENTO DE WEB SERVICE	18
2.3.1 Java	19
2.3.2 Spring	20
2.4 COMUNICAÇÃO ENTRE OS SERVIÇOS	20
3. PROPOSTA DO TRABALHO	21
3.1 INTRODUÇÃO À APLICAÇÃO	21
3.2 DIAGRAMA DE CASO DE USO	21
3.3 MODELAGEM DO BANCO DE DADOS	23
3.4 DIAGRAMA DE CLASSES	24
3.5 PROPOSTA DE INTERFACE GRÁFICA	26
3.6 QUADRO DE MODELO DE NEGÓCIOS	29
4. DESENVOLVIMENTO	30
4.1 DESENVOLVIMENTO DA API	30
4.2 DESENVOLVIMENTO DO APLICATIVO MOVÉL	34
4.3 DEPLOY DA API NO HEROKU	39
5. CONCLUSÃO	40
5.1 TRABALHOS FUTUROS	40

REFERÊNCIAS..... 41

1. INTRODUÇÃO

O transporte rodoviário de carga é responsável por cerca de 60% da movimentação nacional de cargas, possuindo uma frota com mais de 1,5 milhões de veículos (ANTT, 2011). Segundo Navarro (2018) destaca que 60% das mercadorias no Brasil é realizada por transportadores autônomos, e um dos seus maiores desafios é manter sua vida financeira organizada.

No entanto, uns dos maiores problemas dos transportadores é a falta de registro das operações realizadas, não sendo possível por exemplo, identificar as entradas e saídas de caixa para análise de lucros, a separação das despesas pessoais e profissionais e a falta de relatórios financeiros tomada de decisões futuras, conforme Martins (2017).

Apesar de existir grandes sistemas para a gestão financeira de transportadoras, para o transportador autônomo acaba sendo inviável contratar esses softwares, pois o custo fica alto, tornando interessante a utilização de softwares que atendam sua necessidade e com um custo menor, além de ter acesso mais fácil. O sistema Driveyou, por exemplo, é uma aplicação web com a ideia mais simples para gerir as despesas e lucros de serviços de entregas e serviço de taxis, mostrando o total de viagens realizadas, quantidade de horas trabalhadas, quilometragem rodada e outras funções.

Tendo em vista a dificuldade no controle financeiro dos motoristas o presente trabalho de conclusão de curso, tem por objetivo apresentar o desenvolvimento de uma aplicação para auxiliar os motoristas e transportadora com um melhor controle financeiro das viagens realizadas, podendo lançar os fretes realizados, despesas durante a viagem, abastecimentos e controlar as manutenções realizadas no veículo.

1.1. OBJETIVOS

O objetivo desse projeto é desenvolver um sistema que ajude os motoristas com um melhor controle financeiro dos fretes realizados, trazendo relatórios para análise de custos. O sistema também terá a função de controle de manutenção do veículo, podendo ser cadastradas todas as manutenções realizadas, e geração de relatórios para análise de peças ou reparos realizados.

1.2. JUSTIFICATIVA

O presente trabalho se justifica, pois conforme analisado a dificuldade para os controles financeiros e de manutenções, o software irá ajudar a controlar as finanças e manutenções do veículo, deixando o trabalho dos motoristas mais proativos e com melhor controle.

1.3. MOTIVAÇÃO

O presente pré-projeto de pesquisa tem por motivação ajudar os motoristas com um melhor auxílio no controle financeiro e nas manutenções realizadas. Desta maneira, espera-se contribuir com o tema na implementação a opção de relatório para análise dos lucros obtidos, histórico das manutenções realizada, facilitando o serviço.

1.4. PERSPECTIVA DE CONTRIBUIÇÃO

O conceito do Sistema de Gestão Financeira para Veículos Freteiros ainda está em construção, mas visa a melhoria na qualidade das informações necessárias para análise dos gastos e lucros com os fretes realizados, trazendo cálculos para ter uma média aproximadamente do veículo, também ajudará para ter um histórico das manutenções realizadas no veículo, assim quando for necessário uma consulta facilitará para o motorista encontrar. O software gerado poderá contribuir com a redução do uso de papéis, além de melhorar a forma de localizar informações, deixando mais eficiente o serviço.

1.5. METODOLOGIA

Para atender aos objetivos determinados, serão conduzidos estudos teóricos para dar suporte ao presente trabalho. No desenvolvimento do software, será utilizada a metodologia UML como forma de apresentar o projeto, sendo desenvolvidos os seguintes mecanismos: diagrama de caso de uso, diagrama de classe e o diagrama de entidade e relacionamento. Para a fase de implementação da API REST responsável por processar as requisições HTTP, planejou-se utilizar a linguagem de programação Java com o framework SpringBoot e nos desenvolvimentos gráficos será utilizado o framework Ionic e Angular que são escritos

em HTML, CSS, JavaScript e TypeScript, facilitando a construção de interfaces de usuário modernas e de alta qualidade, além de possibilitar a geração de uma aplicação para ambas as plataformas Android e IOS. Em relação à base de dados, devido às suas características de robustez e confiabilidade, optou-se pelo MySQL.

1.6. RECURSOS NECESSÁRIOS

Para atender os objetivos propostos neste trabalho, serão necessários os seguintes recursos de hardware e software:

- Hardware
 - o Notebook
 - Processador Intel Core i7 2.8 GHz.
 - Disco SSD 512 GB.
 - Memória RAM 16,0 GB.
- Software
 - o Eclipse – Ambiente de desenvolvimento do back-end da aplicação.
 - o Visual Studio Code – Ambiente de desenvolvimento do front-end da aplicação.
 - o GIT – Sistema para controle de versões distribuídas.
 - o Ionic CLI – Para trabalhar com o framework Ionic.

1.7. ESTRUTURA DO TRABALHO

- Capítulo 1 – Introdução: contextualização, hipótese, objetivos, justificativa, estado da arte e perspectivas de contribuição.
- Capítulo 2 – Revisão bibliográfica: Revisão bibliográfica dos recursos de software, conceitos aplicados e uma abordagem detalhada das tecnologias a serem utilizadas.
- Capítulo 3 – Planejamento do software e Implementação: Levantamento de requisitos mínimos para o desenvolvimento do produto. Posteriormente, o desenvolvimento dos

componentes do software, prototipação das telas e o quadro de modelo de negócios (Business Model Canvas).

- Capítulo 4 – Desenvolvimento: Abordagem da estrutura utilizada, para organizar os pacotes do projeto.
- Capítulo 5 – Conclusão: Conclusões finais do projeto, e abordagem sobre propostas futuras.
- Referências – Fontes de artigos e publicações referenciadas durante o trabalho.

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo é abordado os conceitos de desenvolvimento móvel nas plataformas Android e iOS, Desenvolvimento Web, as opções de frameworks para o desenvolvimento híbrido e tecnologias usadas para a codificação do Web Services.

2.1 DESENVOLVIMENTO MÓVEL

No Brasil, 91,7% das pessoas com mais de 10 anos têm um telefone celular, de acordo com levantamento recente do IBGE (Instituto Brasileiro de Geografia e Estatística). Com isso, vem crescendo cada vez mais o desenvolvimento móvel com intuito de facilitar cada vez mais o acesso para o usuário. O Android e iOS são os sistemas operacionais para celular mais populares do mundo, segundo relatório publicado pela International Data Corporation (IDC).

2.1.1 Android

Segundo Cidral (2011), Android é um sistema operacional Open Source lançado pela Google, para uma gama de aparelhos móveis. Com ferramentas e recursos, o S.O ajuda a criar apps para smartphones, tablets, TVs e carros que são usados por bilhões de pessoas todos os dias. Em sua documentação, para criar um aplicativo é recomendável utilizar a IDE Android Studio que também disponibiliza um gerenciador de dispositivos virtuais, assim é possível emular o aplicativo para testes sem exportá-lo. O código final em Java então é compilado para um arquivo APK que pode ser instalado em um aparelho Android.

2.1.2 IOS

O IOS é um sistema operacional Open Source lançado pela Apple, de início para smartphones e agora para outros aparelhos móveis. Sua linguagem de programação é o Swift que oferece binários para macOS e Linux que podem compilar código para iOS, macOS, watchOS, tvOS e Linux.

2.1.3 Tecnologias Para Desenvolvimento Móvel

Em desenvolvimento móvel podemos desenvolver a aplicação nativa ou híbrida. O desenvolvimento nativo é voltado apenas para o sistema operacional que é solicitado, ou seja, ele vai atender apenas um público, Android ou IOS(Apple). Sendo assim quando o aplicativo é pensado para ser desenvolvido de forma nativa, as diferentes linguagens para seu desenvolvimento precisam priorizar a melhor experiência do usuário.

No desenvolvimento de aplicativos híbridos, são compostos com linguagens e ferramentas WEB, possibilitando a utilização do mesmo código, para sistemas Android e IOS. Eles têm a capacidade de acessar recursos nativos do dispositivo através do Cordova e Phonegap, que servem basicamente para criar um app nativo, capaz de abrir uma webview executando o HTML/CSS e JavaScript, e mostrar ao usuário, conforme ABRANCHES (2018).

Os custos para desenvolver um aplicativo híbrido fica mais reduzido, assim como a demanda de prazo no desenvolvimento, tendo em vista que um único código é capaz de gerar app para ambas plataformas. Já os custos para desenvolver um aplicativo nativo fica mais elevado devido a ser um código para cada plataforma, com isso, aumentando o prazo de entrega do aplicativo.

Para este projeto foi escolhido trabalhar com o framework Ionic pela sua galeria de componentes acompanhado do Angular, por ter mais produtividade e um menor custo. O Ionic foi desenvolvido por Max Lynch, Ben Sperry e Adam Bradley da Drifty Co, em 2013. muito comparado ao React Native por conta de também ser codificado em HTML, CSS e JavaScript (IONIC, 2020). O diferencial do Ionic é gerado uma WebView que envolve toda a aplicação, ao invés de gerar código nativo, porém seus componentes e elementos seguem um layout de UI responsivo com estilo nativo. Quando é feito o build da aplicação é possível executar o projeto normalmente no navegador ou gerar sua versão para as plataformas móveis com os frameworks Cordova ou Capacitor, ambos oferecem sistemas de plugins que permitem a comunicação entre código JavaScript e funcionalidades da plataforma nativa. A diferença dessas opções é que o Cordova foi desenvolvido pela Apache e o Ionic lançou seu próprio serviço, Capacitor.

2.2 DESENVOLVIMENTO WEB

No desenvolvimento web foi utilizado as tecnologias HTML, CSS e JavaScript para a codificação. O HTML (*HyperText Markup Language*) é uma linguagem de marcação responsável pela criação das páginas web. Através dessa linguagem, o desenvolvedor especifica atributos para um texto, com por exemplo, fontes, tamanho e cor, ou ações que permite o vínculo com outra página, que é apresentado após o clique do usuário. Um fator que distingue o HTML é que ele não é uma linguagem de programação propriamente dita, mas sim classificada como uma linguagem de formatações de textos ou definição da estrutura de uma página, conforme Alves (2015).

O CSS (Cascading Style Sheets) é utilizado para definir aparência nas páginas web, definindo como serão exibidos os elementos contidos no código, tendo a vantagem de efetuar a separação entre o formato e o conteúdo de um documento, conforme Pereira (2009).

Segundo Oliveira (2020), o JavaScript é uma linguagem de programação orientada a objetos, interpretada e executado pelo navegador web. A linguagem apresenta uma sintaxe similar à linguagem Java, e seu objetivo é oferecer uma melhor interatividade das páginas. Uma característica do JavaScript é que ele não apresenta tipos de dados, qualquer variável definida é do tipo variante, sendo assim, o tipo do dado da variável é definido conforme a informação armazenada naquele momento.

Criado pelo Google em 2016, o framework Angular é uma estrutura de design de aplicativos e plataforma para criação aplicativos de página única eficientes e tem módulos para comunicação com Web Services (ANGULAR, 2021). Para o desenvolvimento web foi escolhido o Angular pela sua galeria de componentes, para aproveitar de seus módulos para WebService.

2.3 DESENVOLVIMENTO DE WEB SERVICE

O conceito de Web Service é utilizado para transferir dados através de protocolos de comunicação para diferentes plataformas, independentemente das linguagens de programação utilizadas nessas plataformas, para isso, é necessária uma linguagem intermédia que garanta a comunicação entre a linguagem do Web Service e o sistema que

faz o pedido ao Web Service. Para tal, existem protocolos de comunicação como o SOAP (Simple Object Access Protocol) e o REST (Representational State Transfer).

O SOAP utiliza o XML para enviar mensagens e, serve-se do protocolo HTTP para transportar os dados. O REST é um protocolo mais recente que surgiu com o objetivo de simplificar o acesso aos Web Services, baseando no protocolo HTTP e permitindo utilizar vários formatos para representação de dados, como JSON, XML, RSS, entre outros (OPENSOFTE, 2016).

Neste capítulo serão descritas as tecnologias a serem usadas na construção da API dos softwares propostos neste trabalho.

2.3.1 Java

Java é uma linguagem de programação orientada a objetos que foi lançado pela primeira vez em 1995, na Sun Microsystems. Diferente de outras linguagens o Java usa um conceito chamado máquina virtual, que entre o sistema operacional e a aplicação há uma camada que é responsável por interpretar os comandos escritos da aplicação e traduzir para o sistema operacional correspondente (CAELUM, 2016). Devido suas vantagens de portabilidade, segurança, linguagem simples e de alta performance, foi escolhido para a implementação da web service, que terá a seguinte estrutura de camadas conforme a figura 1.

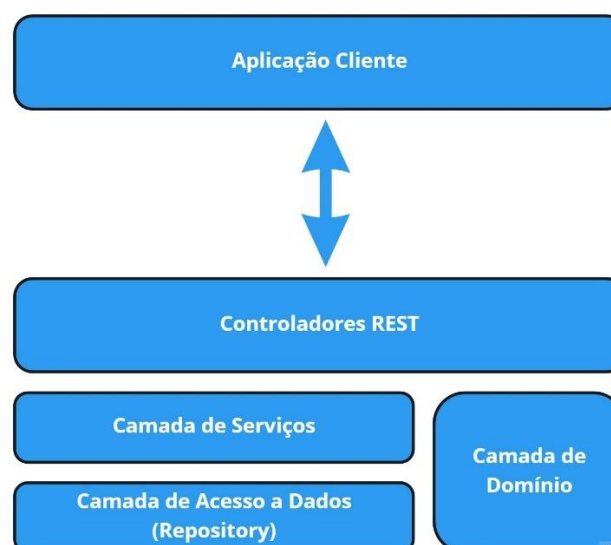


Figura 1: Estrutura de camadas

Fonte: Curso: Spring Boot com Ionic - Estudo de Caso Completo, Nélio Alves

2.3.2 Spring

O Spring é um *framework open source*, criado por Rod Johnson, por volta de 2002. O framework foi criado com intuito de simplificar a programação em Java, trazendo a possibilidade construir aplicações que antes só era possível utilizando Enterprise JavaBeans. O Spring possui vários módulos com Spring Security, Spring Data, entre outros, cabendo ao programador a dizer ao Spring quais módulos usar (GENTIL, 2012).

Um dos elementos principais do Spring é o suporte de infraestrutura no nível do aplicativo, o framework se concentra no encanamento dos aplicativos corporativos para que as equipes possam se concentrar na lógica de negócios no nível do aplicativo, sem vínculos desnecessários com ambientes de implantação específicos (SPRING, 2022).

2.4 COMUNICAÇÃO ENTRE OS SERVIÇOS

Para a comunicação e integração dos serviços foi utilizada a modelo cliente servidor, que é uma arquitetura de aplicação distribuída, ou seja, temos o serviço back-end, que é responsável por fornecer recursos conforme nosso cliente, que trata se da camada de front-end faça uma requisição de algum serviço. As vantagens de utilizar a arquitetura cliente servidor é que os recursos e o banco de dados ficam centralizados, além de oferecer mais segurança, a fim de evitar os problemas. Conforme na figura 2 é possível analisar como funciona está arquitetura, onde o servidor fica rodando a aplicação back-end, e o cliente fica hospedado para que os usuários possam acessar os softwares.

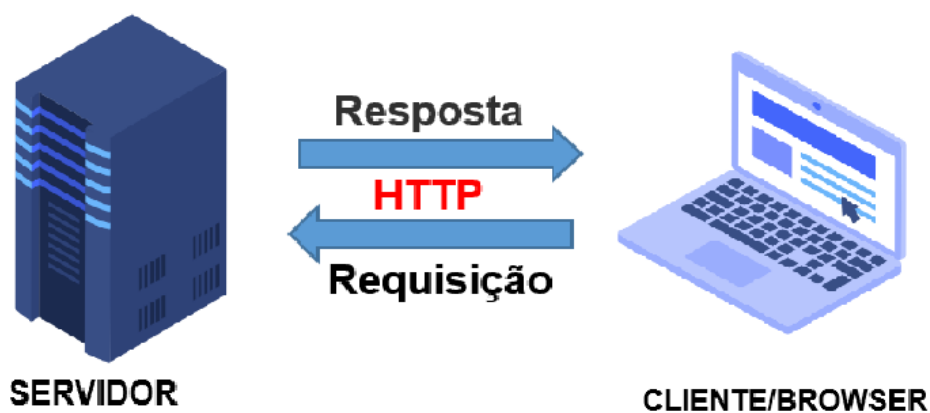


Figura 2: Arquitetura Cliente Servidor.
Fonte: Autoria própria.

3. PROPOSTA DO TRABALHO

Este trabalho tem como proposta um aplicativo e um website para o controle financeiro de fretes realizados da transportadora, desenvolvido com as tecnologias citadas nos capítulos anteriores.

3.1 INTRODUÇÃO À APLICAÇÃO

A aplicação consiste em duas plataformas, uma delas é um sistema web, onde os usuários administradores da transportadora terão acesso para cadastros de motoristas, visualizações de relatórios de viagens, no qual terá as despesas e os lucros lançado pelos motoristas.

Na aplicação mobile os motoristas terão acesso por meio de um login, e terá as funções para adicionar novos fretes/viagens realizados, e para cada viagem irão lançar os abastecimentos, despesas e os pedágios, quando finalizarem a viagem a transportadora terá o acesso as informações decorrentes da viagem para fazer o acerto de contas com os motoristas. Os motoristas por sua vez terão acesso aos dados da viagem finalizada, também no aplicativo terão relatórios com todos os documentos necessários que outras transportadoras solicitam para cadastro e efetuação de carregamento.

3.2 DIAGRAMA DE CASO DE USO

Segundo Fowler (2005), UML (Unified Modeling Language), é uma linguagem de notação gráfica, apoiada por um modelo único, que ajuda na descrição e no projeto de sistemas de software, normalmente aqueles que utilizam o estilo orientado a objetos.

Na UML, os diagramas de caso de uso modelam o comportamento de um sistema e ajudam a capturar os requisitos do sistema. Sua função principal é descrever as interações entre o sistema e os usuários.

O diagrama de caso de uso da Figura 3, representa as ações disponíveis na visão do usuário do tipo “Motorista”.

As rotinas dos motoristas são cadastrar viagens, cadastrar despesas, cadastrar abastecimento, cadastrar pedágios e gerar guia de documentos para cadastro em transportadoras ou indústrias que prestarão o serviço.

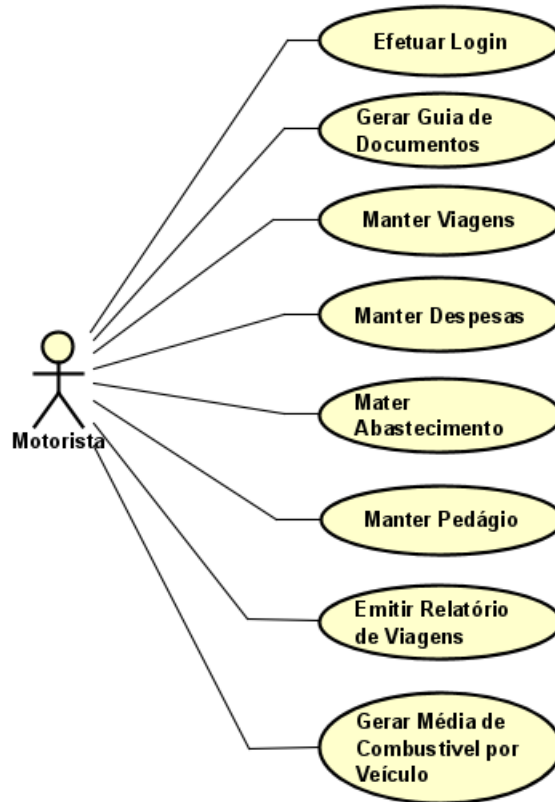


Figura 3: Diagrama de caso de uso ator do tipo motorista

O diagrama de caso de uso da Figura 4, representa as ações disponíveis na visão do usuário de tipo "Transportadora".

Já as rotinas dos usuários administradores que são as transportadoras será visualizar os ganhos de sua frota, cadastrar motoristas, visualizar viagens realizada por determinado motorista para a conferencia do caixa do veículo e visualizar relatório de média de combustível por veículo.

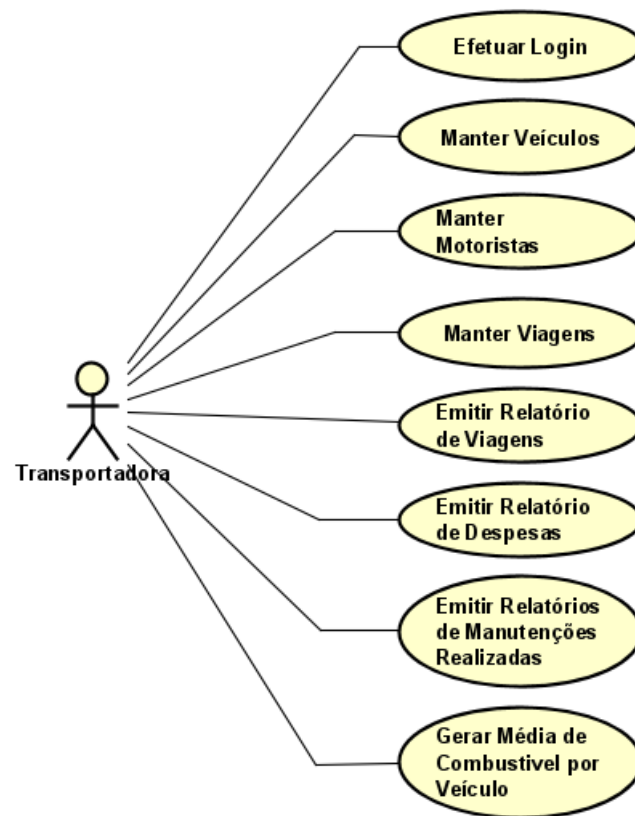


Figura 4: Diagrama de caso de uso ator do tipo transportadora

3.3 MODELAGEM DO BANCO DE DADOS

O SGBD escolhido foi o PostgreSQL 14, um banco de dados do tipo relacional e com sua vantagem principal de oferecer todas as funcionalidades de graça. A Figura 5 apresenta o diagrama de entidade e relacionamento, construído através do aplicativo DB Designer Fork.

As tabelas do banco de dados foram criadas com base no diagrama de classe que foi possível obter quais seriam as tabelas necessárias para armazenamento dos dados.

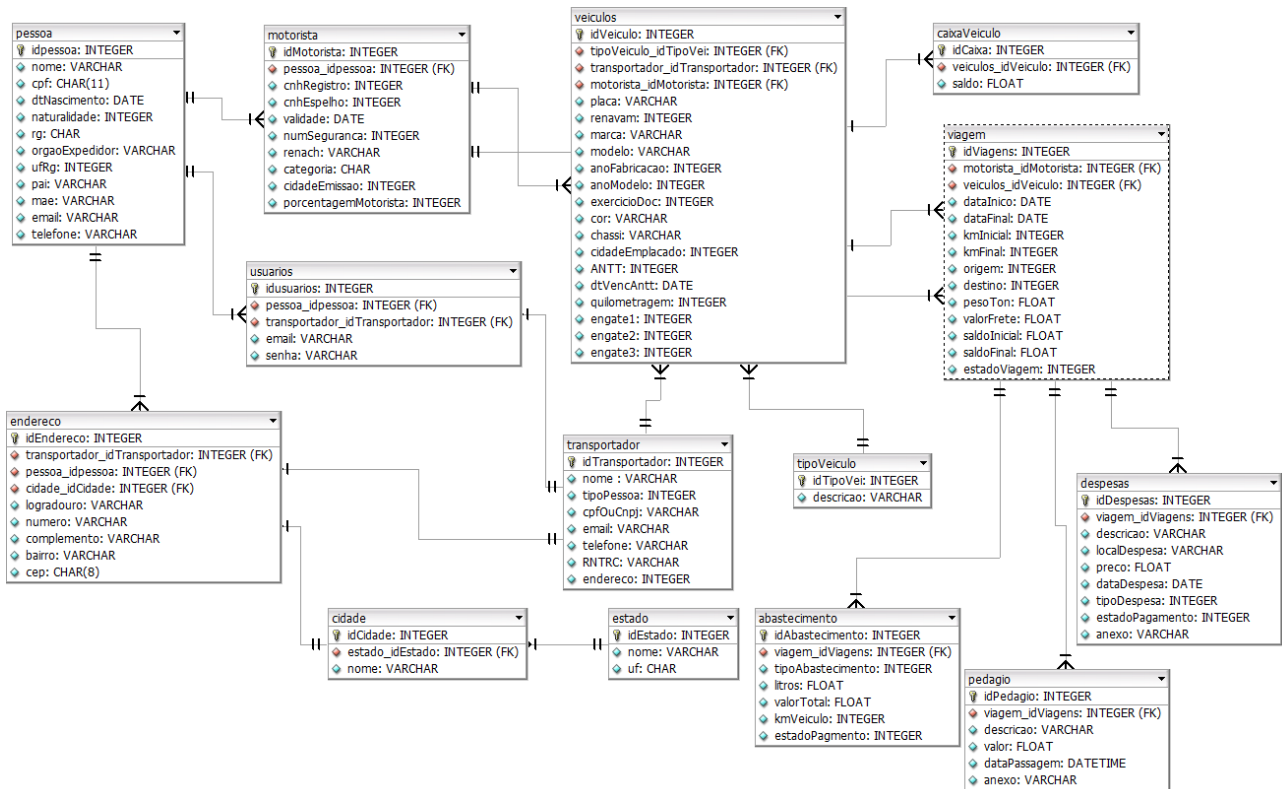


Figura 5: Diagrama de Entidade e Relacionamento

3.4 DIAGRAMA DE CLASSES

Segundo Fowler (2005), o diagrama de classes é uma representação estática para descrever a estrutura de um sistema apresentando suas classes, atributos, operações e as relações entre os objetos. A figura 6 apresenta o diagrama de classes das entidades mapeadas do banco de dados para o Java de forma orientada a objetos. Estão presentes apenas as classes que mapeiam entidades, seus métodos funcionais fazem parte das classes de serviço e controle seguindo o modelo MVC.

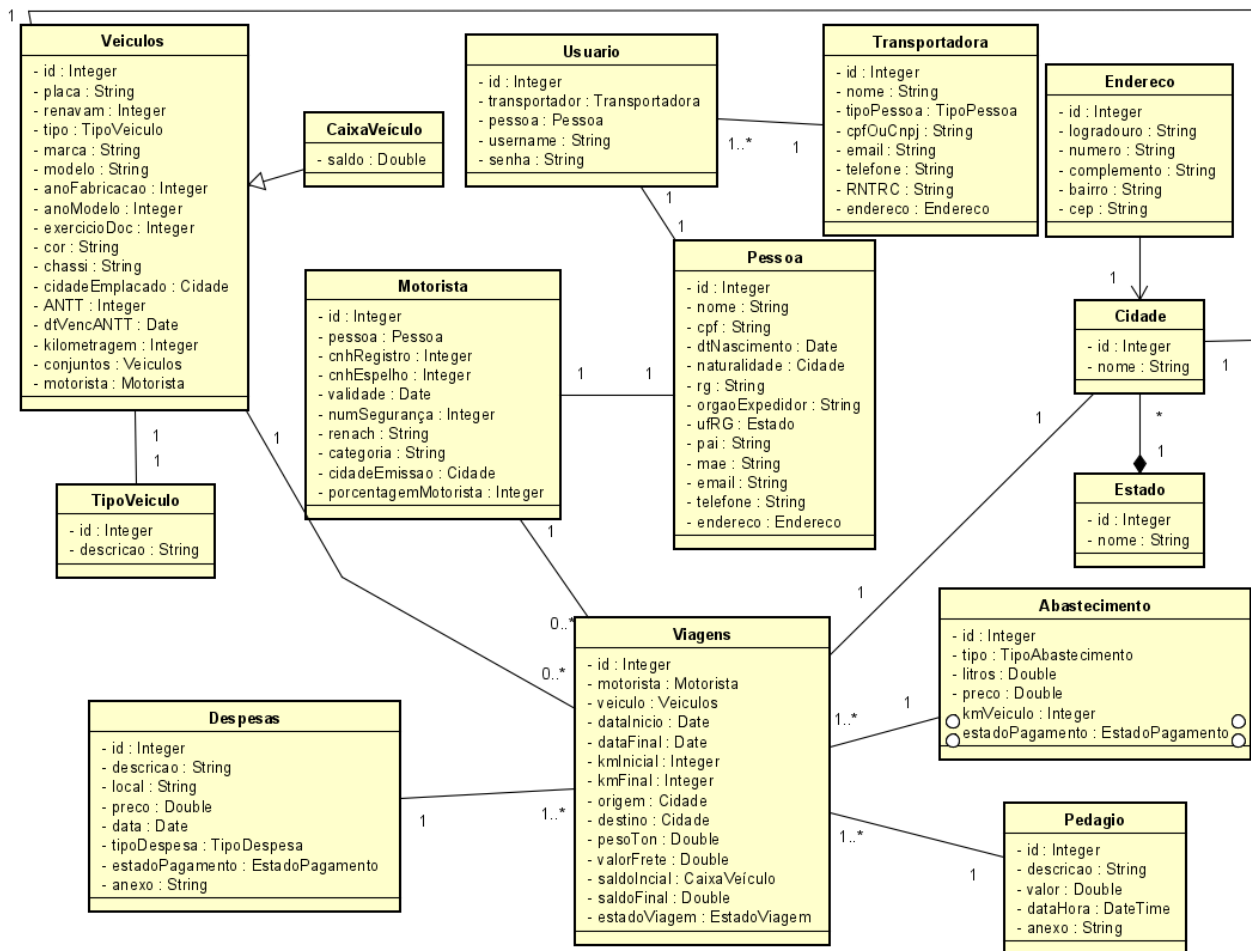


Figura 6: Diagrama de Classes

O tipo de classe *enumeration* é um recurso utilizado em linguagens orientadas a objeto para a representação de um conjunto fixo de constantes. Para o desenvolvimento deste trabalho foi levantado a necessidade de cinco classes de enumeração, para representar tipo de despesa, tipo de pessoa, tipo do abastecimento, o estado da viagem e o estado de pagamento, conforme apresentado na figura 7.

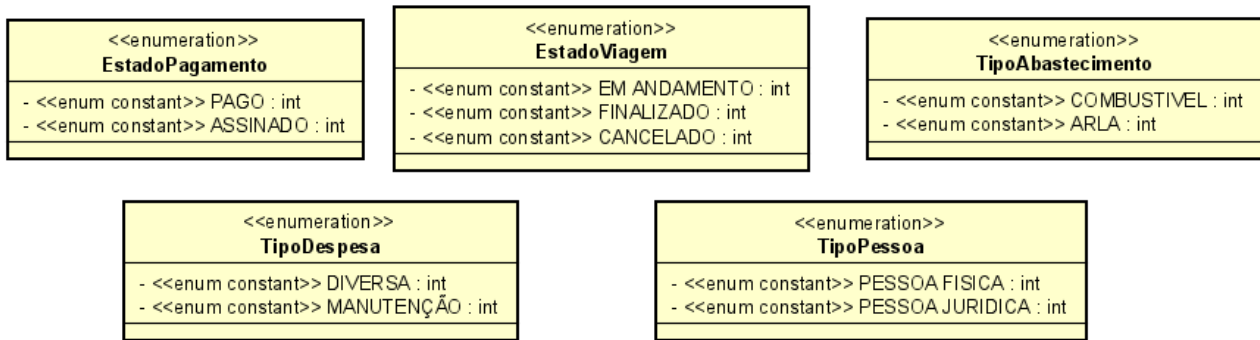


Figura 7: Diagrama de Classes Enumeradas

3.5 PROPOSTA DE INTERFACE GRÁFICA

Nesse subcapítulo temos os designs de algumas das telas principais que foram implementadas. Estes protótipos foram criados com a ferramenta de prototipação e planejamento de projetos, Axure RP 10. Uma ferramenta com vários componentes prontos facilitando o planejamento das telas.

No aplicativo que o motorista tem acesso foi projetado uma interface simples para que o usuário consiga utilizar o aplicativo sem dúvidas. Na tela principal do aplicativo tem o cabeçalho que é composto por um botão que representa o sidebar e um ícone do avatar para fazer logout e redirecionamento do perfil do usuário. O conteúdo principal será uma lista de viagens que o motorista já registrou e que está em andamento, podendo ser pesquisada por nome, ou através da data, conforme a figura 8.

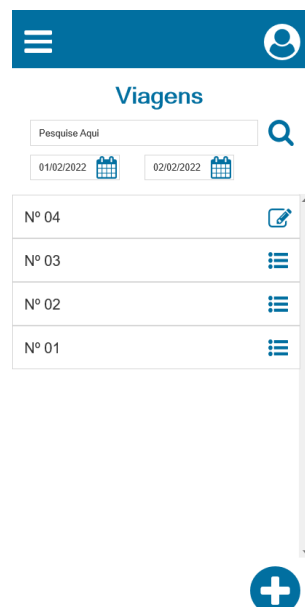


Figura 8: Protótipo da página inicial do aplicativo móvel

Quando o usuário abrir uma viagem, caso ela esteja em andamento será aberto outra tela que terá as informações da viagem, e onde será possível fazer o lançamento de despesas, abastecimentos e pedágios, figura 9.



O protótipo da tela de viagem em andamento apresenta um cabeçalho azul com ícones de menu e perfil. O título "Viagem" está centralizado. Abaixo, há campos de entrada para "Veículos:", "Motorista:", "KM Inicial:", "KM Final:", "Data Inicial:", "Data Final:", "Origem:" e "Destino:". O campo "Peso Ton:" também está presente. Para o frete, há opções de "Valor por Tonelada" e "Valor Total".

As despesas são gerenciadas em uma seção com um campo de busca "Pesquise Aqui" e uma tabela com duas linhas de exemplo:

Descrição	Valor	Ações
Descrição 02/02/2022	R\$ 250,00	[Editar] [Excluir]
Descrição 02/02/2022	R\$ 250,00	[Editar] [Excluir]

Os abastecimentos seguem a mesma estrutura, com uma tabela de exemplo:

Descrição	Quantidade	Valor	Ações
Óleo Diesel 02/02/2022	500Lts	R\$ 2.500,00	[Editar] [Excluir]
Aria 32 02/02/2022	40Lts	R\$ 160,00	[Editar] [Excluir]

Os pedágios também são gerenciados em uma tabela com uma linha de exemplo:

Descrição	Valor	Ações
Palmital Km 413.490 02/02/2022 00:00	R\$ 67,20	[Editar] [Excluir]

Na base da tela, há um botão "PRÉ VISUALIZAÇÃO" e dois botões "SALVAR" e "FINALIZAR VIAGEM".

Figura 9: Protótipo da tela de viagem em andamento

As telas de lançamento serão bem simples, serão uma modal com os campos necessários e com a possibilidade de anexos de documentos, figura 10.

The figure shows three wireframe screens for data entry, each with a blue header bar containing a white 'X' icon.

Lançar Despesa: Includes fields for 'Descrição:', 'Local:', 'Preço:', and 'Data:'. Below these are radio buttons for 'Tipo de Despesa:' (Diversa, Manutenções do Veiculo) and 'Estado Pagamento:' (Pago, Assinado). An 'Anexos:' section contains a file upload icon. A blue 'SALVAR' button is at the bottom.

Lançar Pedágio: Includes fields for 'Descrição:', 'Preço:', and 'Data:'. An 'Anexos:' section contains a file upload icon. A blue 'SALVAR' button is at the bottom.

Lançar Abastecimento: Includes fields for 'Data:', 'Tipo:' (with a dropdown menu showing 'Óleo Diesel'), 'Litros:', and 'Preço:'. Below these are radio buttons for 'Estado Pagamento:' (Pago, Assinado) and a 'Km Veículo:' field. An 'Anexos:' section contains a file upload icon. A blue 'SALVAR' button is at the bottom.

Figura 10: Protótipos das telas de lançamentos

Por fim, quando a viagem já estiver encerrada o motorista irá somente visualizar o relatório da viagem, conforme apresentado na figura 11.

The wireframe shows a 'Relatório da Viagem' screen with a blue header bar containing a menu icon and a user profile icon.

Relatório da Viagem

Veículos: AAA-1234 / AAA-4321
 Motorista: ANTOINO DA SILVA
 KM Inicial: 0 KM Final: 50
 Data Inicial: 02/02/2022 Data Final: 02/02/2022
 Origem: PALMITAL - SP Destino: OURINHOS -SP
 Peso Ton: 32,0

Despesas: Total: R\$ 40,00

Descrição	Pago	R\$
02/02/2022	Pago	R\$ 20,00
02/02/2022	Pago	R\$ 20,00

Abastecimentos: Total: R\$255,00

Óleo Diesel	35Lts	Pago	R\$
02/02/2022	35Lts	Pago	R\$ 175,00
Arla 32	20Lts	Pago	R\$ 80,00

Pedágios: Total: R\$ 67,20

Palmital Km 413.490	R\$
02/02/2022 00:00	R\$ 67,20

Saldo Inicial: 1.000,00
 Valor Frete: 800,00
 Retirada Motorista: 80,00
 Estadia: 0,00
 Estadia Motorista: 0,00
 Receita Líquida: 357,80
 Saldo Final: 1.357,80

Figura 11: Protótipos da tela de relatório da viagem

3.6 QUADRO DE MODELO DE NEGÓCIOS

O *Business Model Canvas* ou simplesmente Canvas, foi criado pelo pesquisador Alexandre Osterwalder. Sua proposta é acabar com a burocracia das estratégias de gestão de projetos tradicionais, conforme Donato (2021).

Uma das principais características do método Canvas é seu formato visual, que é possível visualizar todas as funções e atividades do negócio de maneira simples, além de facilitar a implantação de mudanças na infraestrutura, ajudar na visualização de áreas críticas e que precisam de inovação, reduz retrabalhos entre outros obstáculos.

Para o desenvolvimento deste trabalho foi desenvolvido um modelo de negócio com o método canvas, conforme apresentado na figura 12.

Canvas - Sistema p/ Gestão Financeira de Fretes



Figura 12: Business Model Canvas

4. DESENVOLVIMENTO

O desenvolvimento dos softwares foi iniciado com as configurações das IDE's necessárias para o desenvolvimento do projeto, após realizar as configurações foi inicializado o back-end com o Spring, posteriormente o projeto em Ionic e Angular que são responsáveis pela parte visual do projeto front-end.

4.1 DESENVOLVIMENTO DA API

A estrutura da aplicação back-end foi organizada de forma que cada pacote tenha sua responsabilidade, por exemplo temos o pacote de serviços que é onde se encontra todos os serviços da aplicação, pacote dos controladores que é responsável pela chamada de serviços, temos o pacote responsável pelos repositórios do projeto o pacote de modelo que é onde fica todas as classes de modelo do sistema, temos outro que fica responsável pela camada de transferência de dados, outro responsável pela segurança e outro para configurações gerais do projeto, conforme apresentado na figura 13.

```

  v src/main/java
    v br.com.mytravels.apimytravels
      > config
      > controller
      v data
        > dto
        > model
      > exception
      > repository
      > security
      > services
      > ApiMytravelsApplication.java

```

Figura 13: Estrutura de arquivos do projeto da API.

Após a configuração do banco de dados e das dependências que estão sendo utilizadas no projeto, foi desenvolvido as classes de modelos que servem para definir os atributos de cada objeto e as operações que cada objeto realiza. Para a criação das classes de modelos foi seguido o diagrama de classes apresentado no capítulo 3, como por exemplo a classe transportadora apresentada na figura 14.

```

1 package br.com.mytravels.apimytravels.data.model;
2
3 import java.io.Serializable;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31 @Entity
32 @Table(name="transportadora")
33 @JsonPropertyOrder({"id", "nome", "tipoPessoa", "cpfOuCnpj", "email", "telefones", "enderecos"})
34 public class Transportadora extends ResourceSupport implements Serializable {
35     private static final long serialVersionUID = 1L;
36
37     @Id
38     @GeneratedValue(strategy = GenerationType.IDENTITY)
39     @Column(name = "id")
40     @JsonProperty("id")
41     private Long key;
42
43     @Column(name = "nome", nullable = false, length = 255)
44     private String nome;
45
46     @Column(name = "tipo", nullable = false)
47     @Enumerated(EnumType.STRING)
48     private TipoPessoa tipoPessoa;
49
50     @Column(name = "cpf_cnpj", nullable = false, length = 19)
51     private String cpfOuCnpj;
52
53     @Column(unique=true)
54     private String email;
55
56     @ElementCollection
57     @CollectionTable(name="TELEFONE")
58     private Set<String> telefones = new HashSet<>();
59
60     @Column(unique=true)
61     private String RNTRC;
62
63     @Column(name = "vencimento_rntrc")
64     private LocalDateTime vencimentoRNTRC;
65
66     @OneToMany(mappedBy="transportadora", cascade=CascadeType.ALL)
67     private List<Endereco> enderecos = new ArrayList<>();
68

```

Figura 14: Classe de Modelo Transportadora.java.

Para facilitar o trabalho com persistência de dados, foi utilizado o Spring Data JPA, que é responsável pela implementação dos repositórios, ele oferece as funcionalidades que são mais utilizadas para o acesso ao banco de dados. Para a utilização desse recurso é preciso criar uma interface específica para cada classe de entidade/modelo, e nelas, estender a interface `JpaRepository`. Na figura 15 temos o exemplo da classe `TransportadoraRepository.java`.


```

1 package br.com.mytravels.apimytravels.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7
8
9 @Repository
10 public interface TransportadoraRepository extends JpaRepository<Transportadora, Long>{
11
12     @Transactional(readOnly=true)
13     Transportadora findByEmail(String email);
14
15 }
16

```

Figura 15: Classe TransportadoraRepository.java.

Na camada de serviços foi instanciado a classe de repositórios e criado os métodos de buscas, criação, alteração e exclusão do objeto, além de ser a camada que é definida as regras de negócio do sistema, conforme apresentado na figura 16.

```

1 package br.com.mytravels.apimytravels.services;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13 @Service
14 public class TipoVeiculoService {
15
16     @Autowired
17     private TipoVeiculoRepository repository;
18
19     public List<TipoVeiculo> findAll() {
20         return repository.findAll();
21     }
22
23     public Optional<TipoVeiculo> findById(Long id) {
24         return repository.findById(id);
25     }
26
27     public TipoVeiculo create(TipoVeiculo tipoVeiculo) {
28         return repository.save(tipoVeiculo);
29     }
30
31     public TipoVeiculo update(TipoVeiculo tipo) {
32         var entity = repository.findById(tipo.getKey())
33             .orElseThrow(() -> new ResourceNotFoundException("Nenhum registro encontrado para este ID"));
34         entity.setDescricao(tipo.getDescricao());
35         repository.save(entity);
36         return entity;
37     }
38
39     public void delete(Long id) {
40         TipoVeiculo entity = repository.findById(id)
41             .orElseThrow(() -> new ResourceNotFoundException("Nenhum registro encontrado para este ID"));
42         repository.delete(entity);
43     }
44
45 }
46

```

Figura 16: Classe TipoVeiculoService.java.

Nas classes do pacote *controller*, são onde fica as responsabilidades de controlar as requisições da aplicação, indicando quem deve receber e para quem deve recebe-las, conforme na figura 17. Quando é realizado uma requisição pela URL, por exemplo, `http://localhost:8080/api/tipoVeiculo` sem passar nenhum corpo na requisição é chamado o método que foi criado no serviço para retornar todos os tipos de veículos cadastrados no banco, porém se o usuário faz uma requisição no método POST passando um objeto do tipo `TipoVeiculo.java`, o controlador chama o método responsável por criar um novo tipo de veículo, e assim como nos outros métodos de PUT e DELETE.

```

1 package br.com.mytravels.apimytravels.controller;
2
3 import static org.springframework.hateoas.mvc.ControllerLinkBuilder.LinkTo;
4
25 @Api(tags = "Tipo Veiculo")
26 @RestController
27 @RequestMapping("/api/tipoVeiculo")
28 public class TipoVeiculoController {
29
30     @Autowired
31     private TipoVeiculoService service;
32
33     @ApiOperation(value = "Listar todas os tipos de veiculos" )
34     @GetMapping(produces = { "application/json", "application/xml" })
35     public ResponseEntity<List<TipoVeiculo>> findAll() {
36         List<TipoVeiculo> list = service.findAll();
37         list.stream().forEach(p -> p.add(
38             LinkTo(methodOn(TipoVeiculoController.class).findById(p.getKey())).withSelfRel()
39         )
40     );
41     return ResponseEntity.ok().body(list);
42 }
43
44     @ApiOperation(value = "Buscar uma tipo de veiculo específico através do ID" )
45     @GetMapping(value =("/{id}", produces = { "application/json", "application/xml" })
46     public ResponseEntity<Optional<TipoVeiculo>> findById(@PathVariable("id") Long id) {
47         Optional<TipoVeiculo> tipo = service.findById(id);
48         return ResponseEntity.ok().body(tipo);
49     }
50
51     @ApiOperation(value = "Cadastrar novo tipo de Veiculo")
52     @PostMapping(produces = { "application/json", "application/xml"},
53         consumes = { "application/json", "application/xml"})
54     public TipoVeiculo create(@RequestBody TipoVeiculo tipoVeiculo) {
55         TipoVeiculo tipo = service.create(tipoVeiculo);
56         tipo.add(LinkTo(methodOn(TipoVeiculoController.class).findById(tipo.getKey())).withSelfRel());
57         return tipo;
58     }

```

Figura 17: Classe TipoVeiculoController.java.

Durante o desenvolvimento da aplicação foi utilizado o Swagger para auxiliar na descrição, visualização e consumo dos recursos disponíveis na API REST, garantindo a padronização das interfaces de integração. Para que o mesmo funcionasse, foi importado as dependências *springfox-swagger2* e *springfox-swagger-ui*, após a importação das

dependências foi criado uma classe de configuração para ativar o serviço através da anotação `@EnableSwagger2`, além de outras configurações disponíveis como informações que são adicionadas no cabeçalho do documento, na figura 18 temos a sua representação.

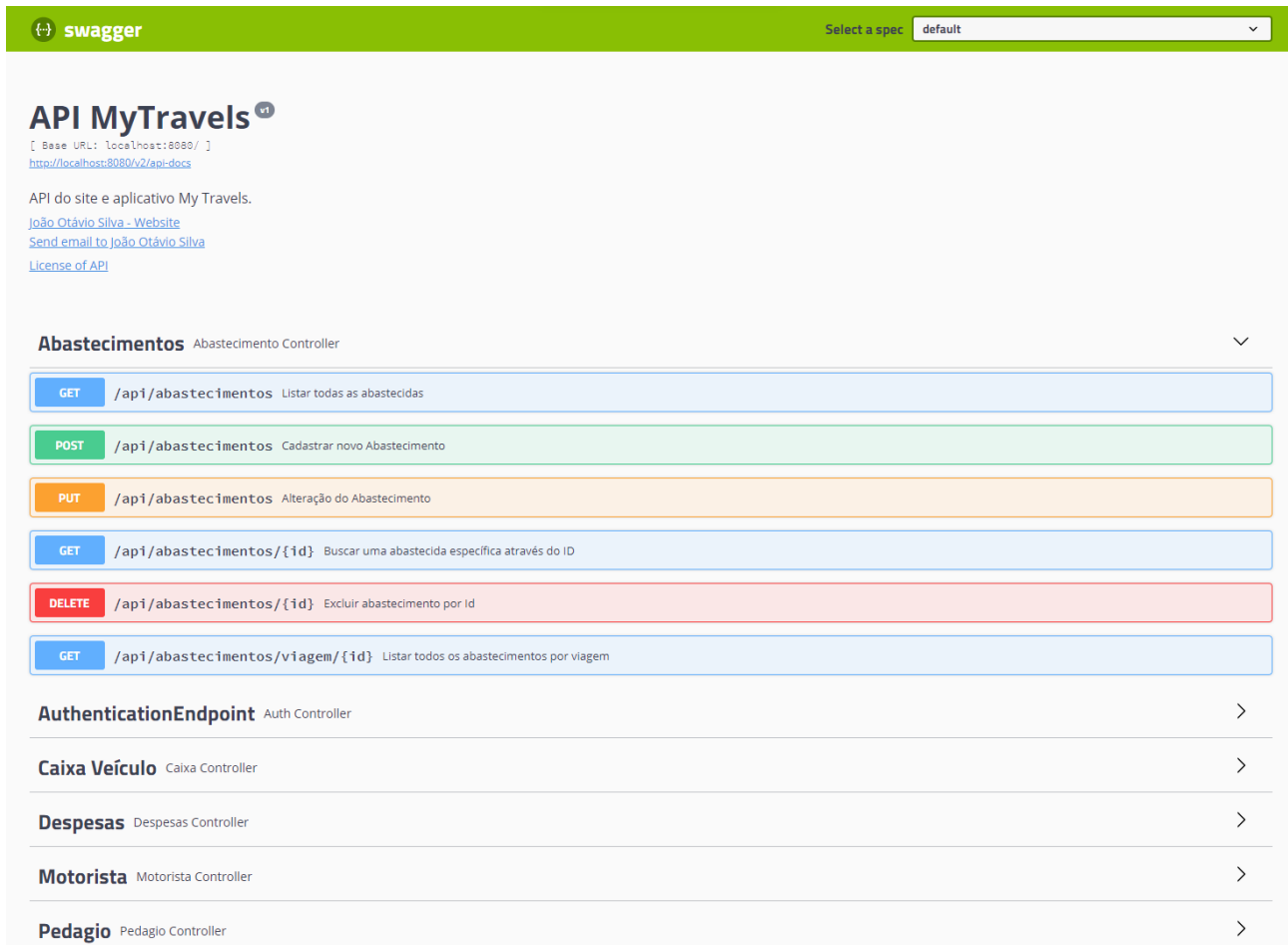


Figura 18: Interface do Swagger-ui.

4.2 DESENVOLVIMENTO DO APLICATIVO MOVÉL

O aplicativo móvel foi desenvolvido com o Ionic e Angular, foi iniciado a partir do comando “ionic start” e dado o nome para o projeto, após dar o seu nome o framework disponibiliza alguns modelos de projeto, como o *tabs* que é um layout baseado em guias, o *blank* que é um projeto vazio com uma única pagina e o escolhido para o projeto que é o *sidemenu* que é disponibilizado um menu lateral como padrão. Por fim é mostrado os possíveis frameworks para integração como por exemplo, React, Vue e o Angular que foi o escolhido para o desenvolvimento do app. A estrutura do projeto foi dividida em pastas dentro do diretório *app* como por exemplo, uma pasta para criar os serviços, outra para páginas, outra

par os componentes que serão utilizados nas páginas, outra para criar um método de verificação para ver se o usuário está logado, e um para configurações em geral.

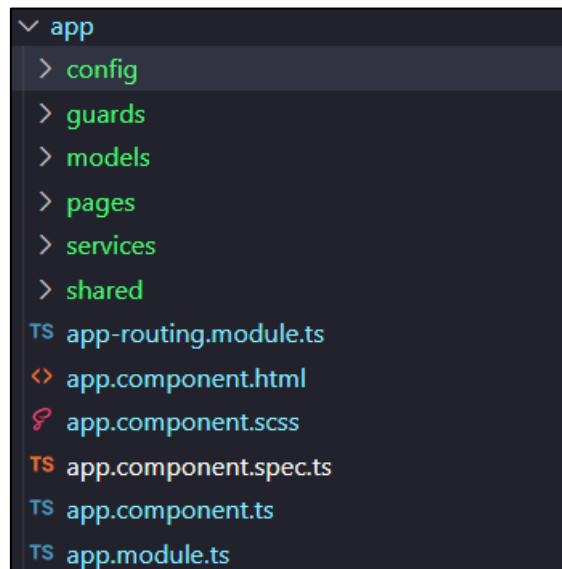


Figura 19: Estrutura do projeto Ionic.

Na comunicação das aplicações foi utilizado o módulo de cliente HTTP oferecido pelo framework Angular. Esse mecanismo oferece uma interface simplificada para realizar essa tarefa, mas que internamente utiliza o XMLHttpRequest. Para utilização é necessário realizar a importação do HttpClientModule no módulo do projeto, depois instanciamos ele no construtor para utilizar os recursos que ele oferece, conforme no exemplo abaixo.

```
src > app > services > TS viagens.service.ts > ViagensService > getAllMotorista
1  import { StorageService } from './storage.service';
2  import { HttpHeaders, HttpClient } from '@angular/common/http';
3  import { Injectable } from '@angular/core';
4  import { Observable } from 'rxjs';
5  import { API_CONFIG } from '../config/api.config';
6
7  @Injectable({
8    providedIn: 'root'
9  })
10 export class ViagensService {
11
12     constructor(
13         private http: HttpClient,
14         private storage: StorageService
15     ) { }
16
17     getAllMotorista(): Observable<any> {
18         const id = 1;
19         return this.http.get(`${API_CONFIG.baseUrl}/api/viagens/motorista/${id}`, {
20             headers: new HttpHeaders().set('Authorization', `Bearer ${this.storage.getToken()}`)
21         });
22     }
23
24 }
25
```

Figura 20: Serviço de Viagens.

Como apresentado na figura 20, na função `getAllMotorista()` fazemos uma requisição na aplicação back-end passando que é uma requisição do tipo GET passamos a URL para o acesso da informação e no cabeçalho da requisição é informado o token para que o usuário consiga acessar o método. Quando o usuário faz o login, a aplicação retorna o token e o username do usuário e é armazenado no Local Storage da aplicação, e para que o usuário acesse outras paginas do sistema é realizado uma verificação se existe um token armazenado, caso contrario o usuário só pode ter acesso a tela de login.

```
src > app > services > TS storage.service.ts > ...
1  import { STORAGE_KEYS } from '../config/storage_keys.config';
2  import { LocalUser } from '../models/local_user';
3  import { Injectable } from '@angular/core';
4
5  @Injectable({
6    providedIn: 'root'
7  })
8  export class StorageService {
9
10     public getLocalUser() {
11         const user = localStorage.getItem(STORAGE_KEYS.localUser);
12         if (user == null) {
13             return null;
14         } else {
15             return JSON.parse(user);
16         }
17     }
18
19     public getToken() {
20         const user = this.getLocalUser();
21         return user.token;
22     }
23
24     public setLocalUser(obj: LocalUser) {
25         if (obj == null) {
26             localStorage.removeItem(STORAGE_KEYS.localUser);
27         } else {
28             localStorage.setItem(STORAGE_KEYS.localUser, JSON.stringify(obj));
29         }
30     }
31 }
```

Figura 21: Serviço responsável por gerenciar o token armazenado.

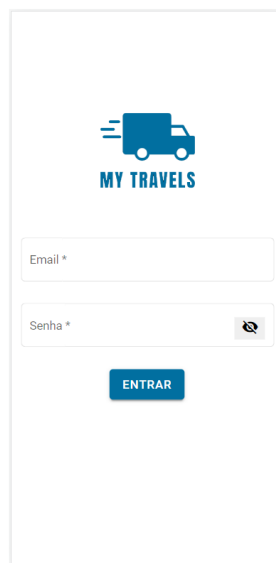
Na figura 21, é apresentado as funções responsáveis por retornar o token, que é utilizado quando é feita uma requisição e esta requisição é necessária a autenticação na API, tem o método que é responsável por setar as informações armazenadas no storage, por exemplo quando é feito o logout é apagado as informações e atribuído nulo para a variável localUser. O método de login é realizado passando as credenciais que são o email e senha do usuário, após passado essas informações o usuário clica no botão para entrar que é chamado a função passando as informações do formulário e chama na classe de serviço a função que

faz a requisição ao servidor que retorna o token e é armazenado no storage da aplicação e redireciona o usuário para a página home.

```
async login() {
  this.dados = false;
  const cred: CredenciaisDTO = this.loginModel.value;
  await this.authService.authenticate(cred).subscribe(
    data => {
      this.response = data;
      this.authService.successfulLogin(this.response.token, this.response.username);
      this.router.navigate(['/home']);
    },
    error => {
      console.log(error);
      this.loginModel.reset();
      this.dados = true;
      this.onError('Não foi possível realizar o login, tente novamente!');
    }
  );
}
```

Figura 22: Função responsável por fazer o login.

A tela de login foi feita padrão tendo a mesma interface tanto para o aplicativo móvel quanto para o sistema web, assim como a implementação do armazenamento do token no storage da aplicação devido a utilização das mesmas tecnologias.



A imagem mostra a interface de login de um aplicativo. No topo, há um ícone de um caminhão azul e o texto "MY TRAVELS" em azul. Abaixo, há dois campos de entrada: "Email *" e "Senha *". O campo de senha possui um ícone de olho para alternar a visibilidade. Abaixo dos campos, há um botão azul com o texto "ENTRAR".

Figura 23: Tela de Login

4.3 DEPLOY DA API NO HEROKU

A Heroku é uma plataforma que foi utilizada para a realização do deploy da aplicação backend na nuvem, para que fosse possível realizar as requisições no aplicativo móvel. A plataforma oferece os serviços de hospedagem para ambas tecnologias como por exemplo, Java, PHP, Ruby, Node e Python. Também oferece um serviço de hospedagem integrado com o Git Hub para atualizar o serviço conforme alterado o repositório no git e feito o push da aplicação. Além do serviço hospedado na Heroku foi configurado a base de dados postgres para que o banco também ficasse centralizado no servidor junto com a aplicação facilitando manutenção.

Para o controle da aplicação no site da Heroku é disponível um painel de controle onde são apresentadas as atividades feitas na aplicação como por exemplos as configurações da base de dados, os últimos *deploys* realizados ou se teve alguma falha durante a execução, e os menus de configurações e de como realizar o *deploy*, conforme apresentado na figura 24.

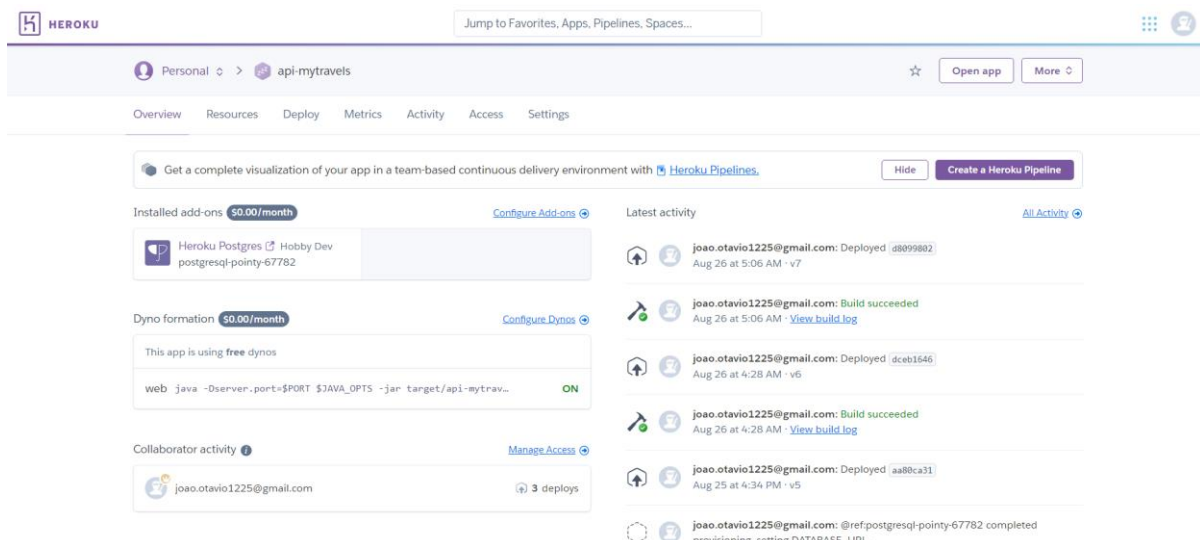


Figura 24: Painel da Aplicação na Heroku.

5. CONCLUSÃO

No início do desenvolvimento do projeto foram abordados no capítulo de revisão bibliográfica um pouco sobre os conceitos das tecnologias que seriam usadas para o seguimento do projeto. Com isso teve o início da modelagem dos diagramas que foram apresentados no capítulo 3, e juntamente algumas imagens para proposta visual do aplicativo. Já no capítulo 4, foi abordado algumas partes dos códigos desenvolvidos na aplicação back-end e códigos do aplicativo.

Grandes partes dos objetivos foram alcançados, um ponto bem importante é que apesar de já ter um conhecimento básico em algumas das tecnologias utilizada no projeto, houve algumas dificuldades, mas nada muito difícil e foram encontrados conteúdos na internet facilitando o entendimento ou muitas vezes auxílio de como tratar erros encontrados nos códigos. Outra dificuldade encontrada, foi que devido ao tempo, não foi possível implementar as funções para armazenamento de anexos e a função para que o aplicativo conseguisse trabalhar de forma offline, mas são funções que estão como trabalhos futuros, além de adicionar outras tecnologias para melhorar as funcionalidades do sistema e facilitar futuramente a hospedagem dos serviços.

O projeto ficará disponível no GitHub e no LinkedIn, para que outras pessoas ou desenvolvedores podem ter acesso aos códigos para darem contribuições, ou até mesmo aproveitarem recursos disponíveis que auxiliam em outros projetos. E com base na finalização desse trabalho, chegou-se à conclusão de que a aplicação desenvolvida terá um valor enorme para a área de logística, com auxílio maior aos motoristas e gerentes de frotas.

5.1 TRABALHOS FUTUROS

Conforme conclusão do projeto foi analisado que há grandes chances do software se tornar um produto para o mercado, e para isso, como propostas futuras foi seria integrado uma API que é responsável por cálculos de pedágios e combustíveis gastos antes da realização da viagem para uma análise básica dos custos. Outra proposta é implementar no aplicativo móvel a funcionalidade offline para que os motoristas consigam realizar cadastros e acessar sem estar conectado há uma rede de internet.

REFERÊNCIAS

ABRANCHES, Junior. Aplicativos e desenvolvimento mobile híbrido x nativo. Disponível em < <https://imasters.com.br/desenvolvimento/aplicativos-e-desenvolvimento-mobile-hibrido-x-nativo/>>. Acesso em: 13 de mar. de 2022.

ALVES, Nélcio. Curso: Spring Boot com Ionic – Udemy. Disponível em < https://www.udemy.com/share/101sie3@yoOtrX1SdgtEQFC0VN7QHm9HUSRZ_GdOfJC EzXkqWaY39dyv1ufjBx6CLx5acoUedg==/>. Acesso em: 22 de abr. de 2022.

ALVES, William Pereira. Java para Web - Desenvolvimento de Aplicações, 1º Edição. São Paulo: Érica, 2015.

ANGULAR, Angular. 2021. Disponível em < <https://angular.io/guide/what-is-angular/>>. Acesso em: 13 de mar. de 2022.

ANTT/RNTRC (2011) Dados do Registro Nacional de Transportadores Rodoviários de Carga. Disponível em Agência Nacional de Transportes Terrestres, Brasília. Acesso em: 16 nov. 2021.

AXURE, Axure RP. 2022. Disponível em < <https://www.axure.com/> >. Acesso em: 15 de mar. de 2022.

CAELUM. Java e Orientação a Objetos. Disponível em < <https://www.caelum.com.br/apostila-java-orientacao-objetos/>>. Acesso em: 13 de mar. de 2022.

CIDRAL, Beline. Afinal, o que é Android?. Disponível em < <https://www.techtudo.com.br/noticias/2011/01/afinal-o-que-e-android.ghtml/>>. Acesso em: 13 de mar. de 2022.

DONATO, Lilian. Modelo Canvas. Disponível em < <https://blog.aevo.com.br/modelo-canvas/>>. Acesso em 19 mar. 2022.

DRIVEYOU, Driveyou. Disponível em < <https://driveyou.com.br/>>. Acesso em: 16 nov. 2021.

FOWLER, Martin. UML Essencial, 3º Edição. trad. João Tortello, Porto Alegre: Bookman, 2005.

GENTIL, Efraim. Introdução ao Spring Framework. Disponível em < <https://www.devmedia.com.br/introducao-ao-spring-framework/26212/>>. Acesso em: 13 mar. 2022.

HEROKU, heroku. Disponível em < <https://driveyou.com.br/>>. Acesso em: 01 set. 2022.

IBGE (2019). Pesquisa Nacional por Amostra de Domicílios Contínua Anual – 4º trimestre. Disponível em < <https://sidra.ibge.gov.br/tabela/7369#resultado/>>. Acesso em: 13 mar. 2022.

IDC (2021), International Data Corporation. Smartphone Market Share. Disponível em < <https://www.idc.com/promo/smartphone-market-share/>>. Acesso em: 13 mar. 2022.

IONIC, Ionic Framework. 2020. Disponível em < <https://ionicframework.com/docs/>>. Acesso em: 29 de jul. de 2021.

MARTINS, Fernando. As 6 melhores dicas de finanças pessoais para caminhoneiros. Disponível em < <https://blogwlmscania.itaipumg.com.br/as-6-melhores-dicas-de-financas-pessoais-para-caminhoneiros/>>. Acesso em: 16 out. 2021.

NAVARRO, Conrado. Caminhoneiros e sua importância para a economia (e crescimento) do Brasil. Disponível em < <https://autovideos.com.br/caminhoneiros-economia-brasil/>>. Acesso em: 16 out. 2021.

OLIVEIRA, Cláudio Luís Vieira. JavaScript descomplicado : programação para a Web, IOT e dispositivos móveis, 1º Edição. São Paulo: Érica, 2020.

OPENSOFTECH, OpenSoft. 2016. Disponível em < <https://www.opensoft.pt/web-service/>>. Acesso em: 13 de mar. de 2022.

PEREIRA, Ana Paula. O que é CSS?. Disponível em < <https://www.tecmundo.com.br/programacao/2705-o-que-e-css-.htm/>>. Acesso em: 13 out. 2021.

SPRING. Spring Framework. Disponível em < <https://spring.io/projects/spring-framework#overview/>>. 2022. Acesso em: 13 de mar. de 2022.