UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

WIGLAN ELLI PATRIEL TOSCAN MARIANI

SOLUÇÃO DE GESTÃO WEB E MOBILE PARA MICROEMPRESAS E AGENTES AUTÔNOMOS NA ÁREA DE PRESTAÇÃO DE SERVIÇOS DE FRETE

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2017

WIGLAN ELLI PATRIEL TOSCAN MARIANI

SOLUÇÃO DE GESTÃO WEB E MOBILE PARA MICROEMPRESAS E AGENTES AUTÔNOMOS NA ÁREA DE PRESTAÇÃO DE SERVIÇOS DE FRETE

Monografia de especialização apresentado ao IV Curso de Especialização em tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Robison Cris Brito

PATO BRANCO
2017

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

UTFPR

TERMO DE APROVAÇÃO

SOLUÇÃO DE GESTÃO WEB E MOBILE PARA MICROEMPRESAS E AGENTES AUTÔNOMOS NA ÁREA DE PRESTAÇÃO DE SERVIÇOS DE FRETE

por

WIGLAN ELLI PATRIEL TOSCAN MARIANI

Este trabalho de conclusão de curso foi apresentado em 08 de dezembro de 2017, como requisito parcial para a obtenção do título de Especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Beatriz Terezinha Borsoi e Vinicius Pegorini. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Robison Cris Brito
Prof. Orientador (UTFPR)

_____
Beatriz Terezinha Borsoi
Banca (UTFPR)

_____
Vinicius Pegorini
Banca (UTFPR)

_____
Robison Cris Brito
Coordenador da IV Especialização em Tecnologia Java

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

MARIANI, Wiglan Elli Patriel Toscan. Solução de gestão *web* e *mobile* para microempresas e agentes autônomos na área de prestação de serviço de frete. 58f. 2017. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2017.

Este trabalho apresenta o desenvolvimento de uma solução de gestão para microempresas e agentes autônomos na área de prestação de serviços de frete. Esse tipo de gestão representa uma das principais procuras por esse nicho de mercado, uma vez que facilita o entendimento, a organização e a gerência dos processos, gerando, assim, resultados mais satisfatórios. A solução proposta consiste em um aplicativo para a plataforma Android, que fazendo uso de uma estrutura Cliente-Servidor realizadas chamadas para um Servidor em nuvem sincronizando os dados lançados pelos usuários. O aplicativo permite a gestão de fretes, bem como os lançamentos de receitas e despesas que ocorrem durante a prestação do serviço. Esses dados ficarão disponíveis para consulta em tempo real pelos administradores em um portal *online*. Para facilitar a troca de mensagem do aplicativo com o servidor e para criar leiautes mais estilizados foram utilizados *frameworks* gratuitos.

Palavras-chave: Android. Cliente-Servidor. Gestão de fretes.

ABSTRACT

MARIANI, Wiglan Elli Patriel Toscan. Web and Mobile Management Solution for micro-enterprises and autonomous agents in the area of freight service delivery. 58f. 2017. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Campus Pato Branco. Pato Branco, 2017.

This work presents the development of a management solution for microenterprises and freelancers in the area of freight services. This type of management represents one of the main demands for this niche market, since it facilitates the understanding, organization and management of the processes, thus generating more satisfactory results. The proposed solution consists of an application for the Android platform, which making use of a Client-Server structure made calls to a Cloud Server synchronizing the data released by the users. The application allows the management of freights as well as the postings of revenues and expenses that occur during the rendering of the service. This data will be available for real-time consultation by administrators in an online portal. To facilitate the exchange of the application message with the server and to create more stylish layouts were used free frameworks.

Keywords: Android. Client-Server. Freight Management.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	*Application Programming Interface*
CSS	*Cascading Style Sheet*
CTC	*Cooperativas de Transporte Rodoviário de Cargas*
CTe	*Conhecimento de Transporte Eletrônico*
DAO	*Data Access Object*
ETC	*Empresas de Transporte Rodoviário de Cargas*
HTML	*Linguagem de marcação de hipertexto*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
MVC	*Model-View-Controller*
REST	*Transferência de Estado Representacional*
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais.
RNTRC	*Registro Nacional de Transportadores Rodoviários de Carga*
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language*
TAC	*Transportadores Autônomos de Cargas*
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

No Brasil, o transporte na modalidade rodoviário representa 61,1% de todo o transporte de cargas realizado, seguido pela modalidade ferroviário com 20,7% e aquaviário com 13,6% (BOLETIN CNT, 2017).

Segundo informações do sistema de Registro Nacional de Transportadores Rodoviários de Carga (RNTRC), o número de veículos vigentes em circulação, que atuam na atividade de transporte de cargas ultrapassa 1,6 milhões, sendo desses 600 mil veículos operadores por Transportadores Autônomos de Cargas (TAC), 1,1 Milhão sob Empresas de Transporte Rodoviário de Cargas (ETC) e 22 mil sob Cooperativas de Transporte Rodoviário de Cargas (CTC). Ao todo, são 555 mil entidades atuando no setor de transporte de cargas, dessas, os motoristas autônomos representam 76% com 424 mil operadores, seguido das CTCs e ETCs com 130 mil entidades (ESTATÍSTICAS ANTT, 2017).

Frente a toda essa movimentação do setor de transporte, surge a necessidade de aprimorar os controles e os processos das empresas, de registrar de forma detalhada todas as operações realizadas, bem como os gastos e receitas, para que, assim, tenha em mãos dados que ajudem na tomada de decisão por parte de seus administradores.

Microempresas que prestam serviços de fretes geralmente terceirizam seus fretes para motoristas que, na grande maioria, atuam de forma autônoma. Assim, a empresa disponibiliza os recursos necessários como os veículos e terceiriza o serviço para motoristas subcontratados. Outra forma de frete é os próprios motoristas que possuem seus veículos e procuram por fretes, apenas repassando as despesas para as empresas e pessoas que o contratam. Em ambas as situações, durante a prestação do serviço, muitas ocorrências precisam ser tratadas diretamente pelos motoristas, como por exemplo, a manutenção dos veículos, especialmente durante a viagem, gastos com alimentação, hospedagem, combustível, pedágios, entre outros. Estes gastos são arcados pelos motoristas e posteriormente descontados do valor que eles teriam a receber pelo trabalho prestado. Frente a todos esses gastos, surge-se então a necessidade de um controle de gestão sobre esses registros.

Para os administradores, é fundamental saber de forma antecipada tudo o que está sendo movimentado durante a prestação do serviço de frete, para que assim ele possa realizar um planejamento financeiro, muitas vezes burocrático, com antecedência, referente a toda movimentação ocorrida. Na ausência dessa informação, quando os motoristas encerram os fretes, eles retornam as empresas com diversos comprovantes, notas fiscais de gastos,

registros de pedágios, etc. Registros esses que precisam ser apurados e reembolsados, porém, como esses fechamentos costumam ser realizados de forma mensal, ocorre que um grande volume de informações chega de uma forma não muito confiável nem organizada, colocando em risco assim, a integridade das informações.

Para propor uma solução que aproxime de forma antecipada a gestão da empresa com esse grande volume de informação gerado, utiliza-se um recurso que já está na mão da maioria dos brasileiros, o *smartphone*. Em 4 anos, o número de pessoas que usam smartphones no Brasil subiu de 3,5 vezes passando de 14% para 62% em 2016. No segundo trimestre de 2015, 72 milhões de brasileiros utilizavam o aparelho (EXAME, 2016).

A solução proposta consiste no desenvolvimento de um aplicativo que seja compatível com a maioria dos smartphones disponíveis, e para isso, optou-se pelo desenvolvimento na plataforma Android.

O Android é um sistema operacional para *smartphones*, desenvolvido por um grupo de empresas e atualmente é liderado pela Google. É um sistema robusto que permite a utilização de todos os recursos oferecidos pelos aparelhos. Em comparação aos demais sistemas operacionais, o Android é o mais vendido no mundo, com 70,01%, seguidos do iOS com 21% e 8,9% para os demais sistemas (CANALTECH, 2016).

No Brasil, em 2013 o Android estava em 88,7% dos *smartphones* vendidos, seguido pelo Windows Phone (6%) e iOS (4,7%). No ano seguinte, subiu para 91%. No início de 2016, a fatia do Android já era de 93% (TECNOBLOG, 2016). Outro ponto a se considerar é que o desenvolvimento de aplicativos para a plataforma Android é gratuito, o que fomenta ainda mais o número de pessoas que a utilizam.

O aplicativo desenvolvido neste projeto, tem como objetivo permitir aos motoristas que registrem todas as operações no ato de seu acontecimento, ou seja, qualquer lançamento de receita ou despesa pode ser registrado no momento da sua ocorrência. Para facilitar o entendimento, o aplicativo permitirá a classificação dos lançamentos por frete e categorias, visando levar para a empresa a informação mais organizada. Essas informações registradas serão sincronizadas em um servidor de dados de forma toda vez que o aparelho encontrar conexão com Internet. Utilizando como base os *smartphones*, o aplicativo desenvolvido proporcionará maior mobilidade e facilidade na inclusão dos registros. Para a gestão da empresa, tendo essas informações de forma antecipada, além de tornar o processo organizado, eliminamos o problema do grande volume de informações que antes era apresentado de forma manual somente nos fechamentos mensais.

2 FERRAMENTAS, TECNOLOGIAS E MÉTODOS

Neste capítulo são apresentadas as ferramentas e tecnologias utilizadas no desenvolvimento do projeto, bem como a forma que se estruturou desde a fase de estudo até a concepção do software.

2.1 FERRAMENTAS E TECNOLOGIAS

Para o desenvolvimento deste projeto, buscou-se a utilização de ferramentas livres e tecnologias gratuitas, as quais já estão presentes no mercado por um tempo necessário para que as defina como estáveis.

O Quadro 1 apresenta as ferramentas e tecnologias utilizadas.

Ferramenta\Tecnologia	**Versão**	**Link**	**Finalidade**	**Ambiente**
Eclipse	Neon.3 Release (4.6.3)	http://www.eclipse.org/	*Integrated Development Environment* (IDE) para desenvolvimento do servidor de requisições e aplicação *web*	Servidor
SpringBoot	1.5.1	https://projects.spring.io/spring-boot/	Configuração e publicação das requisições do Servidor	Servidor
PostgreSQL	10.1	https://www.postgresql.org/	Sistema gerenciador de banco de dados	Servidor
Android Studio	3.0.1	https://developer.android.com/studio/	IDE para desenvolvimento *mobile*	Cliente
Dispositivo Android	Nougat 7.0	https://developer.android.com/studio/	Testes do Aplicativo	Cliente
SqLite	Nativa Android	https://www.sqlite.org/	Banco de Dados para o aplicativo *mobile*	Cliente
RetroFit	2.2.0	http://square.github.io/retrofit/	*Framework* para requisições *client-server*	Cliente
GitHUb	2.15.0	https://git-scm.com/about	Controle de versão para os códigos-fonte	Cliente e Servidor
Bizagi	3.1.0.011	https://www.bizagi.com/	Modelagem de processos	Para o projeto
Atom	1.22.1	https://atom.io	Desenvolvimento *web*	Portal *web*
Angular	4.1	http://angularjs.org	Desenvolvimento *web*	Portal *web*

Quadro 1 – Tecnologias e ferramentas utilizadas para o desenvolvimento do projeto

A seguir, é apresentada um pouco das características de cada ferramenta.

2.1.1 Eclipse

O Eclipse é uma comunidade de código aberto cujos projetos são concentrados na criação de uma plataforma de desenvolvimento aberta composta de estruturas, ferramentas e tempos de execução extensíveis para desenvolvimento, implementação e gerenciamento de software por todo o ciclo de vida. A Eclipse Foundation é uma empresa sem fins lucrativos mantida pelos seus associados que hospeda os projetos do Eclipse e ajuda a cultivar uma comunidade de software livre e um ecossistema de produtos e serviços complementares. Em sua instalação padrão, o Eclipse vem com um conjunto padrão de *plugins*, incluindo as amplamente conhecidas Ferramentas de Desenvolvimento Java (ANISZCZYK, 2012).

Por ser uma comunidade de código aberto, o Eclipse oferece suporte a *plugins* para desenvolvimento em várias linguagens, como C++, Python, Android. Para o desenvolvimento deste projeto, o Eclipse foi utilizado na implementação do servidor de requisições, utilizando a estrutura padrão de um projeto Maven, que torna o uso e a organização das bibliotecas mais eficientes.

2.1.2 SpringBoot

Quando se inicia um novo projeto, umas das principais dificuldades é criar todo o ambiente necessário, por exemplo, incluir dependências que serão utilizadas, definições do servidor da aplicação, configurações de bibliotecas de terceiros, etc. Para facilitar esse trabalho surgiu o Spring Boot.

O Spring Boot é um projeto da Spring que facilita o processo de configuração e publicação das aplicações desenvolvidas. O objetivo do *framework* é construir as soluções da forma mais fácil e rápida possível, aumentando assim a produtividade. Com o uso de anotações, é possível criar o projeto com mais facilidade. Por exemplo, quando o projeto é anotado com *@SpringBootApplication*, automaticamente já é iniciado com a definição do servidor e são providenciadas algumas configurações que sem o uso do Spring teria que ser configurado manualmente. Outras anotações utilizadas foram a @RestController, o que diz

para o *framework* que a classe disponibiliza métodos para troca de informações *Transferência de Estado Representacional* (REST).

2.1.3 PostgreSQL

O PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional, desenvolvido como projeto de código aberto. Atualmente é um projeto coordenado pelo PostgreSQL Global Development Group.

Para o projeto desenvolvido foi utilizado o PostgreSQL para armazenar todas as informações que chegam ao servidor. Informações como motoristas, veículos, fretes, lançamentos de receitas e despesas.

O PostgreSQL é otimizado para aplicações complexas, que envolvem grandes volumes de dados ou que tratam de informações críticas. Assim, para um sistema de comércio eletrônico de porte médio/alto, por exemplo, o PostGreSQL é recomendado, já que esse Sistema Gerenciador de Banco de Dados (SGBD) é capaz de lidar de maneira satisfatória com o volume de dados gerado pelas operações de consulta e venda (ALECRIM, 2013).

Algumas características técnicas do PostGreSQL:

- Compatibilidade multi-plataforma, ou seja, executa em vários sistema operacionais, como Windows, Mac OS X, Linux e outras variantes de Unix;
- Compatibilidade com várias linguagens, entre elas, Java, PHP, Python, Ruby, e C/C++;
- Base de dados de tamanho ilimitado;
- Tabelas com tamanho de até 32 terabytes;
- Quantidade de linhas de até 1.6 terabytes ilimitada;
- Campos de até 1 gigabytes;
- Suporte a recursos como *triggers, views, stored procedures*, schemas, transactions, savepoints, referential integrity e expressões regulares;
- Instruções em *Structured Query Language* (SQL);
- Chaves estrangeiras, views, controle de transações.

2.1.4 Android Studio

O Android Studio é o ambiente de desenvolvimento integrado (IDE) oficial para o desenvolvimento de aplicativos Android e é baseado no IntelliJ IDEA. Além do editor de código e das ferramentas de desenvolvedor avançados do IntelliJ, o Android Studio oferece mais recursos para aumentar a produtividade na criação de aplicativos Android (ANDROID STUDIO, 2017).

Alguns recursos que o Android Studio oferece:

- Um sistema de compilação flexível baseado no Gradle;
- Um emulador rápido com muitos recursos;
- Um ambiente unificado para desenvolver para todos os dispositivos Android;
- Instant Run para aplicar alterações a aplicativos em execução sem precisar compilar um novo aplicativo;
- Modelos de códigos e integração com GitHub para ajudar a criar recursos comuns dos aplicativos e importar exemplos de código;
- Ferramentas e estruturas de teste com diversas possibilidades;
- Ferramentas de verificação de código suspeito para detectar problemas de desempenho, usabilidade, compatibilidade com versões e outros;
- Compatibilidade com C++ e NDK;
- Compatibilidade embutida com o Google Cloud Platform, facilitando a integração do Google Cloud Messaging e do App Engine.

2.1.5 Dispositivo Android

Para realizar os testes do aplicativo, foi utilizado um dispositivo físico, a fim de garantir ao máximo o seu funcionamento. O modelo utilizado foi um Zenfone 3, executando o Android na sua versão 7.0 (Nougat). Testes no emulador que a própria plataforma de desenvolvimento oferece, também foram realizados. No emulador, foi utilizada a versão 6.0 do Android.

2.1.6 SQLite

O SQLITE é um banco de dados nativo que executa nos *smartphones* que fazem uso do Sistema Operacional Android. É uma biblioteca em linguagem C que implementa um banco de dados SQL. Apesar de não ser robusto ao nível de um grande SGBD, ele é suficiente para armazenar dados mais simples.

Algumas características do SQLITE:

- É software Livre/domínio público e multiplataforma;
- É um mecanismo de armazenamento seguro com transações com atomicidade, consistência, isolamento e durabilidade (ACID);
- Não necessita de instalação, configuração ou administração;
- Permite guardar o banco de dados em um único arquivo;
- Suporta bases de dados abaixo de 2 terabytes;
- Não tem dependências externas;

2.1.7 Retrofit

Retrofit é uma das bibliotecas mais conhecidas por parte dos desenvolvedores para se trabalhar com troca de mensagens via requisições *Hypertext Transfer Protocol* (HTTP). É também um projeto open *source* produzido pela Square Inc.

No aplicativo proposto neste trabalho foi utilizado o Retrofit para efetuar a comunicação com o servidor e sincronizar os dados que são informados pelo usuário.

2.1.8 GitHub

Git é um sistema de controle de versão de arquivos. Ele possibilita desenvolver projetos na qual diversas pessoas podem contribuir simultaneamente, editando e criando novos arquivos e permitindo que eles possam existir sem o risco de suas alterações serem sobrescritas (SCHMITZ, 2015).

O Github foi utilizado no projeto como forma de *backup online* dos fontes e também com objetivo de manter versões estáveis do software. Fazer o uso de um controlador de versões é fundamental para o desenvolvimento, uma vez que facilita a localização rápida de

erros de desenvolvimento com o recurso de comparação entre versões dos arquivos e matem as versões estáveis intactas.

2.1.5 Bizagi

Bizagi é um software para modelagem de processos de negócios. Pela simplicidade e capacidade, o Bizagi Modeler é uma ferramenta que auxilia os profissionais no desenvolvimento de fluxogramas para entender como é o processo de uma organização ou setor. Dentre estes profissionais, destacam-se desenvolvedores de softwares, administradores e empresários (LIMA, 2016).

2.1.5 Atom

Atom é um editor de texto de código aberto compatível com diversas plataformas. É um software muito versátil que permite a instalação de diversos *plugins*, que o tornam ainda mais produtivo. É um software muito semelhante a outros programas para edição de texto, como o Sublime e o Visual Studio. Para este projeto, o Atom foi utilizado para o desenvolvimento do *front-end*.

2.1.5 Angular

Angular é um *framework* de desenvolvimento *front-end* para aplicações *web*, desta forma, as aplicações são executadas dentro do navegador do cliente. O Angular tem substituído o iQuery, biblioteca JavaScript muito utilizada pelos desenvolvedores *web*, no entanto, o Angular apresenta diversas funcionalidades e melhorias as quais são inexistentes no jQuery, como, por exemplo, o Data Binding utilizado que relaciona os componentes visual da página com os objetos internos que armazenam as informações.

Algumas características do Angular são: criação de diretivas customizadas, *framework* pronto para trabalhar com *Application Programming Interface* (API) REST, validações de *form client-side*, injeção de dependências e *data binding* de mão dupla

2.2 MÉTODO

As principais atividades realizadas para o desenvolvimento deste trabalho estão listadas a seguir.

2.2.1 Levantamento de Requisitos

Os requisitos foram levantados e definidos tendo como base a necessidade de desenvolver um sistema para a gestão de fretes de microempresas e motoristas autônomos.

O levantamento de requisitos foi realizado pelo autor deste trabalho a partir de entrevistas informais realizadas com profissionais que atuam na área de prestação de serviços (fretes). Por meio dessas entrevistas foi possível verificar as principais dificuldades e falhas no sistema de gestão que a maioria pratica. Quase sempre, os registros das receitas e despesas ocorrem em anotações manuais e, em alguns casos, nada é registrado.

Em entrevista com administradores de microempresas, o principal ponto falho encontrado foi o grande volume de documentos que chegam até o escritório somente no final das viagens realizados pelos motoristas, tornando o processo de conferência oneroso e inseguro. Com base nessas dificuldades, montou-se um plano de ideias visando resolver esses problemas. Este plano resultou na solução proposta neste trabalho.

2.2.2 Planejamento

a) *Levantamento das tecnologias a ser utilizadas*

Tendo como base o escopo do projeto, fez-se uma pesquisa para definir as tecnologias que seriam necessárias para atender aos propósitos. Neste estudo, definiu-se que seria necessário:

- Um aplicativo para a plataforma Android;
- Um servidor escrito em Java que disponibilize requisições REST;
- Um *framework* para fazer chamadas ao servidor por parte do aplicativo (Retrofit);
- Componentes de leiaute para o aplicativo afim de melhor a interface com o usuário (RecyclerView);
- Sistema de controle de versão para os códigos fontes.

b) *Levantamento das IDEs de Desenvolvimento e ferramentas*

Tendo definido as tecnologias utilizadas, buscou-se IDEs que permitissem o uso destas tecnologias. Para isso, definiu-se:

- O Eclipse para construção do Servidor;
- Android Studio para construção do Aplicativo;
- GitHub para controle de versão.

2.2.3 Desenvolvimento

a) *Modelagem das classes (estrutura dos bancos de dados)*

Definiu-se a estrutura do banco de dados, tanto do aplicativo quanto do servidor, como seriam armazenadas as informações, em quais tabelas, que tipos de campos, etc. Neste passo, também foram implementadas as classes “model” dos sistemas, as quais são um espelho do banco de dados.

b) *Leiaute das telas*

Desenvolvido as interfaces (telas) de interação com o usuário, as quais permitissem as operações definidas nos requisitos.

c) *Lógica de funcionamento das telas*

As regras de negócio foram implementadas no aplicativo, juntamente com a persistência dos dados no banco local do dispositivo.

d) *Servidor de Requisições e armazenando no banco de dados*

Desenvolvido a lógica do servidor de requisições para tratar das chamadas realizadas pelo aplicativo, bem como a persistência dos dados no banco de dados do servidor.

e) *Chamadas WebService do aplicativo para o servidor*

Desenvolvido no aplicativo as chamadas para integração dos dados coletados para que estes fiquem disponíveis no servidor. Para esta comunicação foi utilizado o *framework* Retrofit, definido na fase de planejamento.

2.2.4 Testes

Os testes de funcionalidades foram realizados pelo autor do trabalho. A ideia é que a solução seja disponibilizada para demais pessoas para avaliar a aderência do produto.

3 RESULTADO

Neste capitulo é apresentado o resultado da realização deste trabalho, que é a modelagem e a implementação de um sistema como forma de gestão para microempresas e autônomos na área de prestação de serviço de frete.

3.1 ESCOPO DO SISTEMA

A solução proposta visa prover maior gestão sobre os eventos que acontecem no decorrer da prestação de serviço de frete e disponibilizar esses eventos para os seus administradores de uma forma rápida e simples. Desta forma, todos os lançamentos efetuados pelos usuários no aplicativo ficarão disponíveis em um portal de consulta.

O sistema se divide em três principais partes: o aplicativo *mobile* para a entrada de dados, o servidor de requisições que mantém as informações no banco de dados e uma página de consulta das informações registradas.

3.1.1 Aplicativo Mobile

O aplicativo concentra a principal parte do sistema, nele são registrados os fretes. Para cada frete lançado é identificado o motorista, o veículo e dados que identificam o frete, como número do CTe(Conhecimento de Transporte Eletrônico), origem e destino. Para cada frete registrado é possível fazer “n” lançamentos. Cada lançamento representa uma ocorrência do frete, por exemplo, um abastecimento de combustível, uma despesa com hospedagem, e até mesmo um depósito efetuado pela empresa que neste caso será registrado como uma receita. Quando um lançamento é registrado, o usuário informa o valor, a data e a categoria que representa uma classificação do lançamento. Essas categorias podem ser alteradas conforme a necessidade.

O aplicativo permite a inclusão e a manutenção de categorias, veículos, motoristas, fretes e lançamentos. Todas as informações registradas no aplicativo são automaticamente enviadas para o servidor de dados sem a interferência do usuário. Sempre que essas informações forem sincronizadas com o servidor, será apresentada a data e hora em que o

referido evento ocorreu. Também existe uma tela para sincronização manual dos dados caso seja necessário.

3.1.2 Servidor de Requisições

O servidor de requisições tem como objetivo responder as chamadas de sincronização de dados provenientes do aplicativo *mobile* e persistir estes dados no banco. Para isso, foi utilizado o *framework* Spring Boot, que facilita o desenvolvimento destas funcionalidades.

O servidor também é responsável por retornar as chamadas provenientes do portal de consulta. Nessas chamadas, ele retornada uma listagem do que foi requisitado. Por exemplo, o gestor precisa ver quais os lançamentos foram feitos para o frete de código 10, neste caso o servidor recebe uma chamada para listar todos os lançamentos para o frete 10 e é retornada a lista dos lançamentos.

3.1.3 Portal de Consulta

O portal de consulta é responsável por apresentar os dados coletados no aplicativo. Ele pode ser utilizado em qualquer momento e apresentará de forma *online* todas as informações que estejam disponíveis no servidor.

3.2 MODELAGEM DO SISTEMA

Os requisitos levantados para o desenvolvimento do aplicativo classificam-se em duas categorias: funcionais e não funcionais. Os requisitos funcionais representam as funcionalidades consideradas necessárias para o funcionamento do sistema. Os requisitos não funcionais explicitam regras de negócio, restrições ao sistema de acesso, como, por exemplo, requisitos de qualidade, desempenho, segurança e outros.

Por convenção, a referência a requisitos pode ser feita por meio do identificador do requisito, usando a sigla RF para Requisitos Funcionais e RNF para Requisitos Não Funcionais.

O Quadro 2 apresenta a listagem dos requisitos funcionais do sistema.

ID	Descrição
RF 1	O aplicativo deve manter categorias, motoristas e veículos possibilitando sua edição/exclusão.
RF 2	O Aplicativo deve manter o frete com informações como motorista, veiculo, data de inicio, data de termino, origem, destino, número de CTE, valores do frete.
RF 3	No aplicativo, para cada frete cadastrado, deve ser possível manter “n” lançamentos. (incluir, alterar, excluir).
RF 4	No aplicativo, para cada lançamento, deve ser informado valor, data, hora, categoria e deixar um campo aberto para observação.
RF 5	O servidor de requisições deve manter todas os dados enviados pelo aplicativo em um banco de dados para posterior consulta.
RF 6	O servidor deve disponibilizar as informações registradas no banco de dados para o portal de consulta.
RF 7	O portal de consulta deve apresentar todos os fretes e seus lançamentos.
RF 8	Quando utilizado o recurso de sincronização de dados, o aplicativo deve apresentar a data e hora que a informação foi enviada para o servidor.

Quadro 2 – Requisitos Funcionais

O Quadro 3 apresenta a listagem dos requisitos não funcionais do sistema.

ID	Descrição
RNF 1	O aplicativo deve utilizar o *framework* Retrofit para executar chamadas HTTP ao servidor.
RNF 2	Deve-se utilizar o PostgreSQL como banco de dados para armazenagem das informações no servidor.
RNF 3	Tanto as requisições de sincronizações quanto as de consultas de dados, devem retornar um objeto no formato *JavaScript Object Notation* (JSON) para os chamadores.

Quadro 3 – Requisitos Não Funcionais

Com base nos requisitos funcionais e não funcionais, criou-se o diagrama de caso de uso, conforme apresenta a Figura 1.

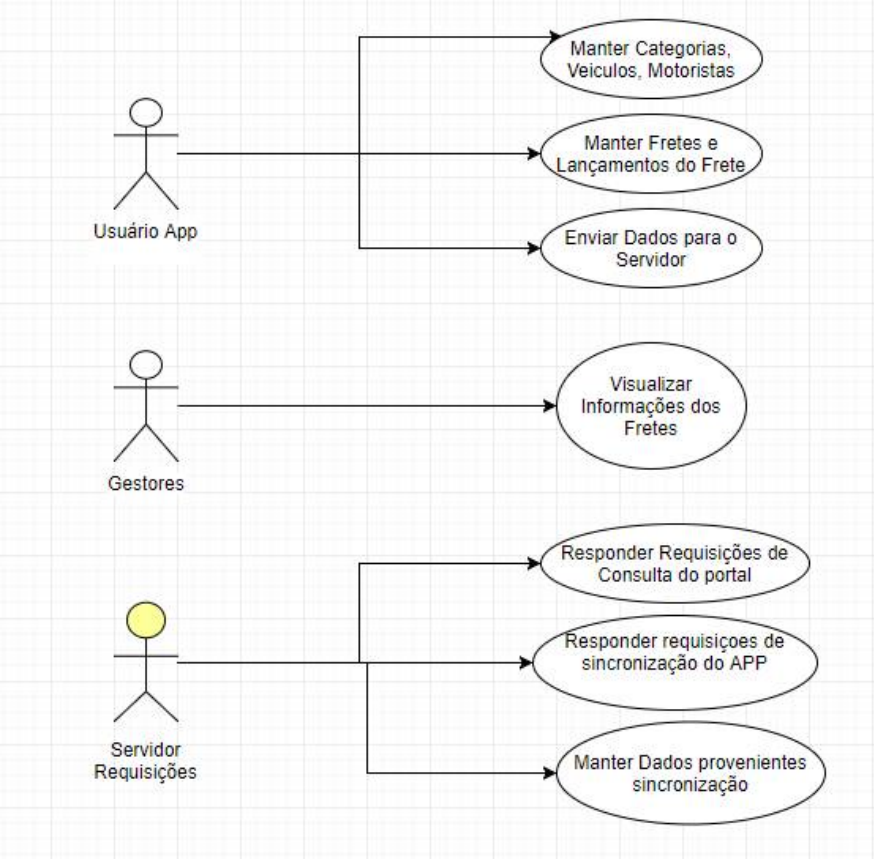

Figura 1 – Diagrama de caso de uso

O modelo de processo, conforme apresentado na Figura 2, representa a responsabilidade de cada ator. O modelo está dividido em três partes, nas quais consistem o aplicativo mobile, o servidor de requisições e o portal de consulta.

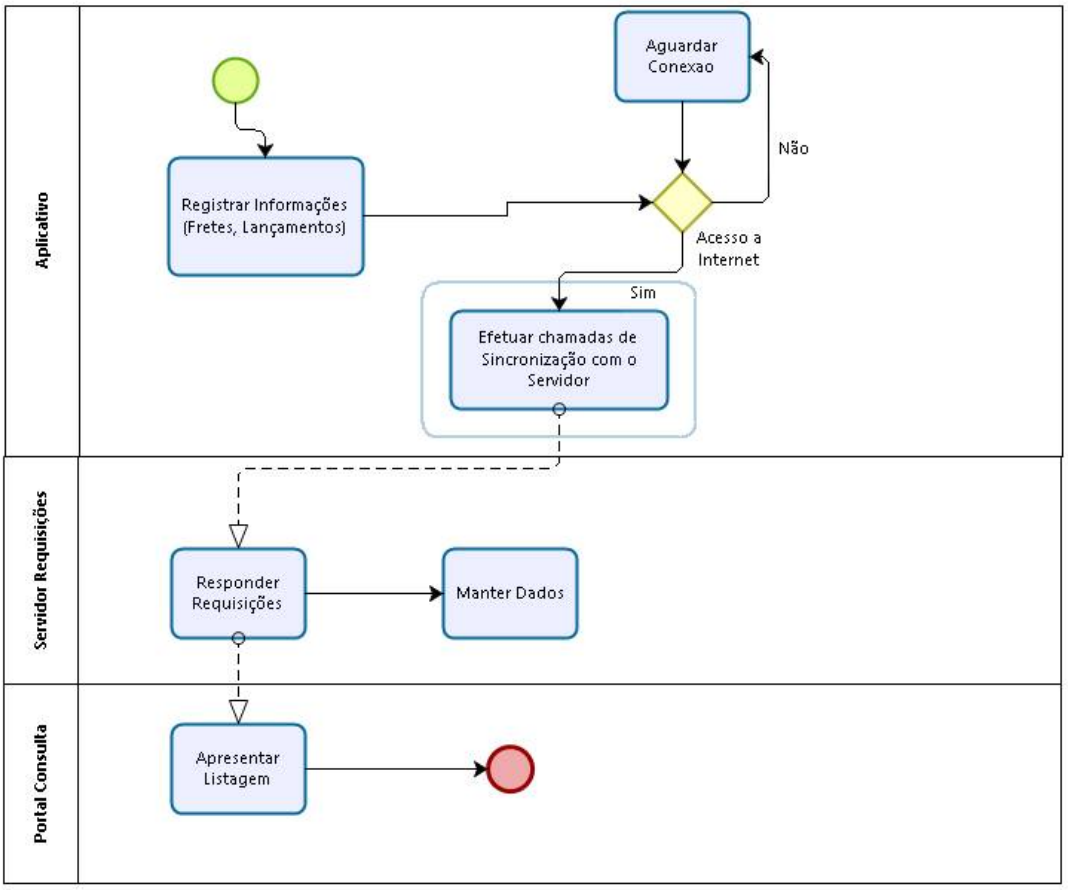

Figura 2 – Modelo de processo - responsabilidade de cada ator

O primeiro quadro representa as responsabilidades do aplicativo, que consiste em manter as informações registradas pelos usuários (motoristas) e no momento de seu registro, caso o dispositivo encontre comunicação com a Internet, será disparado requisições para sincronização com o servidor. Caso não exista conexão com o servidor, o processo fica em *standby* até que exista conexão.

Toda vez que o aplicativo se conectar à Internet, automaticamente os dados serão enviados ao servidor.

O segundo quadro representa o servidor, que sempre que chamado responde às requisições. Ele também tem a responsabilidade de manter os dados provenientes do aplicativo.

No terceiro quadro é apresentado o portal de consulta, que tem como finalidade apresentar os dados armazenados no servidor.

3.3 APRESENTAÇÂO DO SISTEMA

A apresentação do sistema denominado wFrete ocorre em três partes, o aplicativo, o servidor e o portal de consultas.

3.3.1 Aplicativo Mobile

O aplicativo desenvolvido como parte do resultado deste trabalho tem como principal função permitir que o usuário faça o registro de informações referente a prestação de serviço de frete. Esses registros ficam armazenados localmente no dispositivo até uma que uma conexão de Internet seja encontrada, sendo, então, os dados sincronizados com o servidor. Lembrando que não é necessário a comunicação disponível com o servidor para que o aplicativo funcione. Outro detalhe é que para a construção dos leiautes, fez-se uso de componentes do Material Design, a fim de proporcionar interfaces mais atrativas ao usuário.

Ao abrir o aplicativo pela primeira vez, uma tela de “bem vindo” é apresentada – Figura 3 - com o objetivo de direcionar o usuário nos primeiros passos do uso do aplicativo. Essa tela fica visível sempre que o aplicativo for aberto e até que o primeiro frete seja cadastrado. Quando o primeiro frete for cadastrado, a tela de bem-vindo é substituída pela listagem de fretes (apresentada posteriormente).

Para orientar o usuário são apresentados três passos na ordem correta para que o primeiro frete seja lançado.

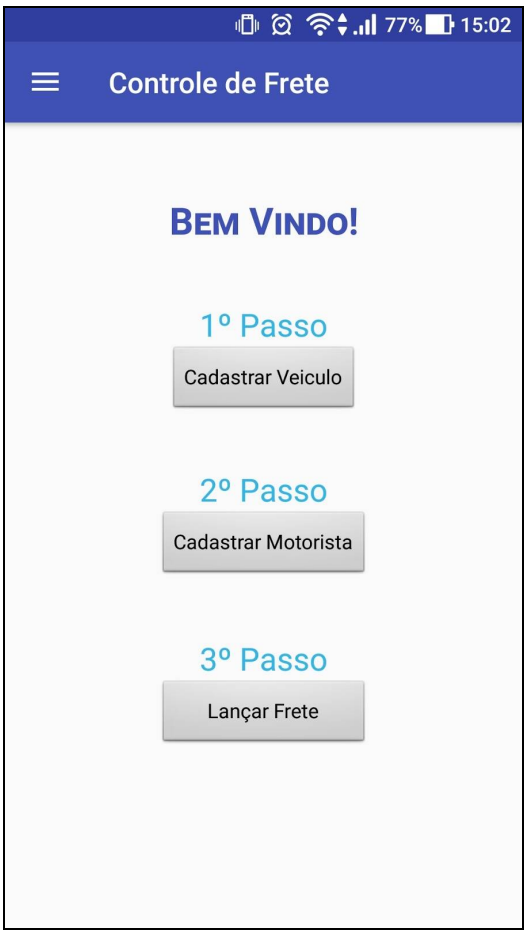

Figura 3 – Tela de Bem Vindo

Ainda na Figura 3, no canto superior esquerdo é apresentado um botão de navegação padrão dos dispositivos. Este botão tem o objetivo de proporcionar a navegabilidade entre as telas do aplicativo. Para isso, ele lista as funções de Cadastro, Lançamento de Frete e configuração do aplicativo. Ao clicar no botão de navegação a Figura 4 é apresentada.

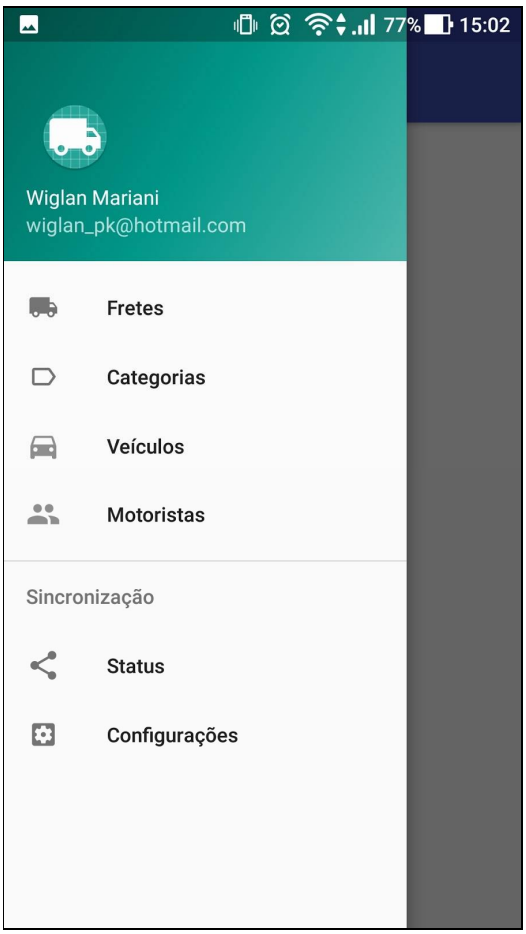

Figura 4 – Menu de navegação

Na Figura 4 são apresentadas, inicialmente, informações do usuário que está utilizando o aplicativo. Em seguida está o menu apresentando o lançamento de frete e o acesso aos três cadastros (categorias, veículos, motoristas). Abaixo dos cadastros é apresentada uma divisão do menu “Sincronização” que apresenta o acesso ao *status* de sincronização de dados com o servidor e o acesso a tela de configuração de comunicação com o servidor. Um detalhe importante é que para cada item do menu, foi adicionado um ícone que faz relação a sua funcionalidade, facilitando assim o entendimento com o usuário.

Na Figura 5 são apresentadas as listagens dos cadastros (categorias, motoristas, veículos). Todas elas seguem um padrão de leiaute criado especificamente para este aplicativo. Em todas as listagens é apresentado um botão para edição do registro, um botão de exclusão e no canto inferior direito, fazendo uso do Material Design, foi criado um botão flutuante responsável por criar um novo cadastro.

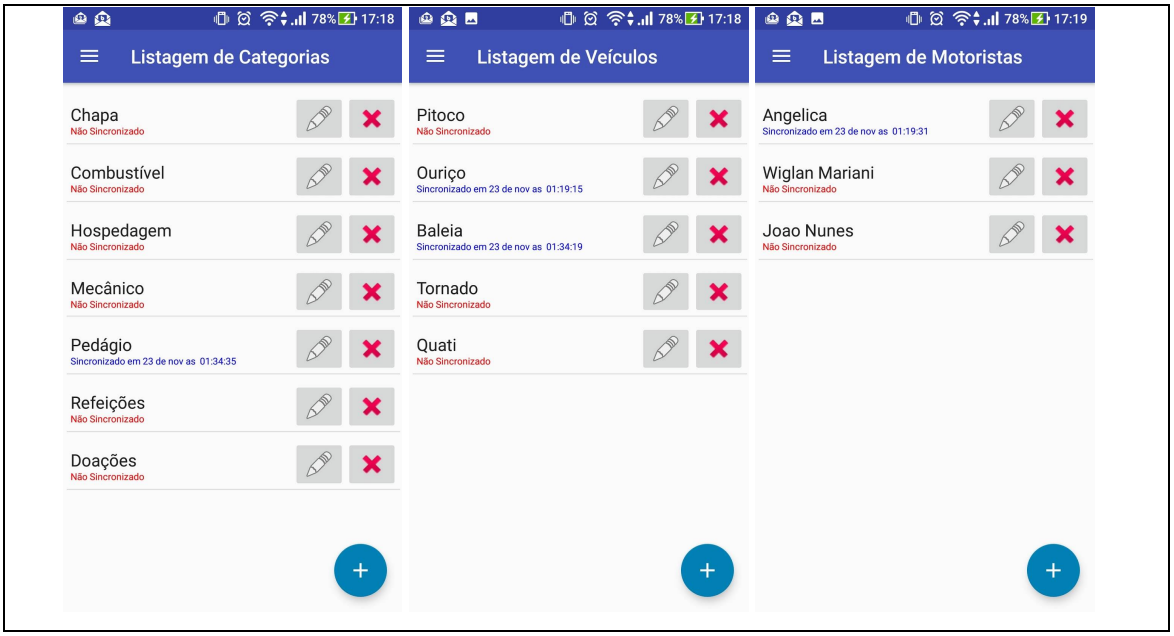

Figura 5 – Listagem de categorias, veículos e motoristas

É importante destacar que abaixo de cada identificação do cadastro, é apresentada a informação referente à data e ao horário que cada registro foi sincronizado com o servidor. Caso o registro não tenha sincronizado, ele é apresentado como “Não sincronizado”. A Figura 6 apresenta melhor esta situação, com a categoria mecânico pendente e a categoria pedágio já sincronizada.

Figura 6 – Explanação dos dados de sincronização

A Figura 7 apresenta a tela de cadastro para as categorias, veículos e motoristas. Essas telas são acessadas por meio do botão Novo apresentado nas telas de listagem e também toda vez que o botão Editar da listagem for pressionado. Para a edição dos registros, a tela de cadastro é apresentada trazendo os dados já salvos anteriormente.

Toda vez que uma edição é confirmada, o registro é salvo no banco de dados limpando os dados de sincronização, fazendo, assim, que eles fiquem pendentes na fila para serem sincronizados novamente. Nas telas de cadastro, um botão Cancelar e um botão Salvar são

apresentados, os quais são responsáveis por retornar à tela chamadora com o cadastro salvo ou cancelado. É possível observar também que um *Hint* pré-estabelecido foi deixado em cada campo, apresentando ao usuário um exemplo de informação que poderia ser cadastrado, facilitando assim o entendimento para os primeiros cadastros.

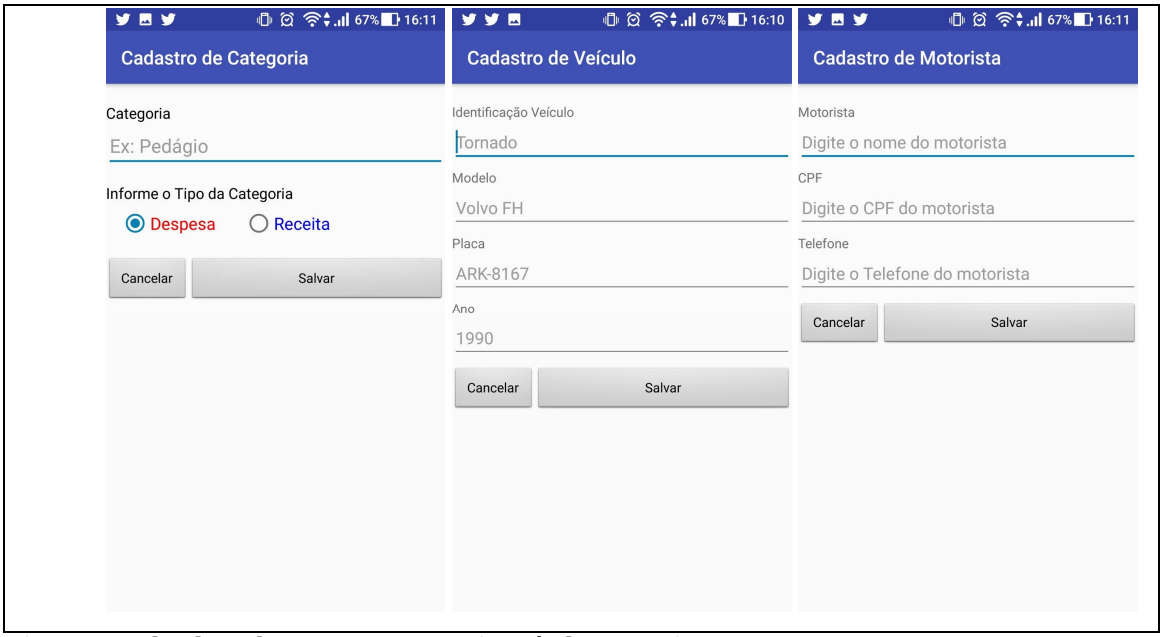

Figura 7 – Telas de Cadastro para a categoria, veículo e motorista

Finalizados os cadastros é apresentada a funcionalidade principal e o objetivo do aplicativo: o registro de fretes e seus lançamentos. A Figura 8, apresenta a listagem dos fretes. Essa listagem será apresentada sempre que existir um frete lançado ou quando chamada pelo Menu-> Fretes. No canto inferior direito, um botão flutuante dá acesso ao cadastro de novos fretes.

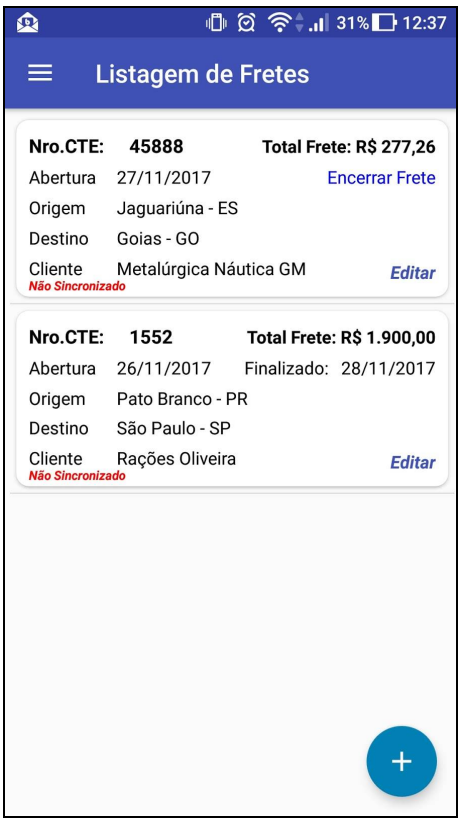

Figura 8 – Listagem de fretes

Buscando apresentar uma listagem com informações que identifiquem o frete e ao mesmo tempo proporcione algumas funcionalidades, preservando um leiaute mais agradável ao usuário, optou-se pelo uso de um componente chamado RecyclerView. Esse componente é o substituto para o ListView, muito utilizado no Android. Cada item da lista representa um frete e para cada item, as principais informações foram apresentadas, como o número do CTE, a origem, destino e o *status* do frete.

Enquanto o frete estiver em aberto, um texto será apresentado permitindo que o frete seja encerrado, conforme apresenta a Figura 9.

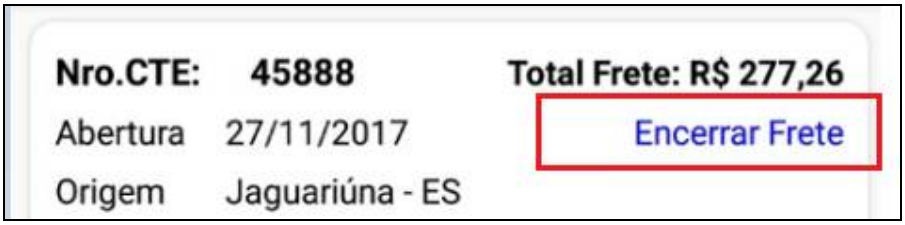

Figura 9 – Encerramento de frete

Ao clicar em encerrar frete, um calendário será apresentado ao usuário para a escolha da data de encerramento, que por padrão, traz a data atual. Sempre que um frete for encerrado, o texto que antes era “Encerrar Frete” passa a apresentar a data em que o frete foi encerrado juntamente com o texto “Finalizado”. A Figura 10 apresenta o calendário de seleção de data para encerramento do frete.

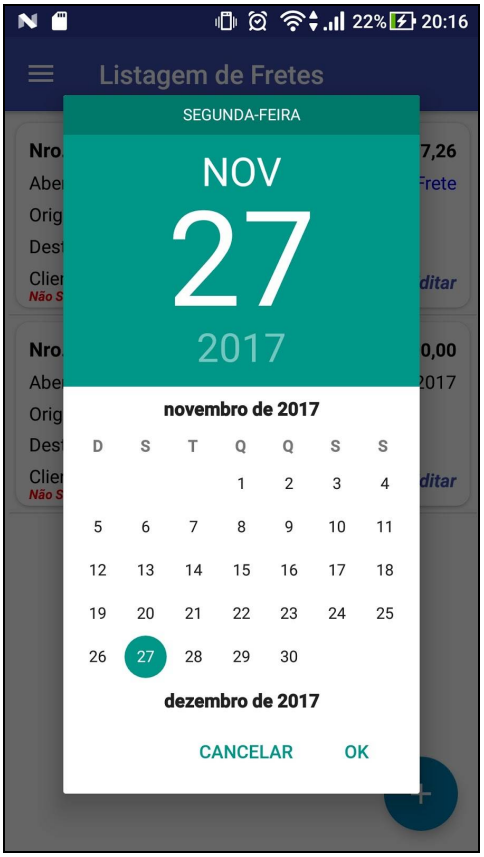

Figura 10 – Calendário para seleção de data do encerramento do frete

Ainda na tela de Listagem de Frete, as informações de sincronização com o servidor ficam disponíveis logo abaixo do Cliente do Frete, conforme o quadro azul destacado na Figura 9. O quadro verde, também na Figura 11, apresenta um texto “Editar” responsável por abrir o cadastro do Frete em operação de Edição, que já carrega o cadastro com todas as informações do Frete.

Figura 11 – Informações de sincronização e função de editar o frete

Quando clicado no botão flutuante de “Novo”, a tela de cadastro é apresentada, conforme Figura 12. Nela, todas as informações do frete precisam ser cadastradas, como número do CTE (informação obrigatória para a prestação de serviço de frete), valor do frete, peso da carga, valor total do frete, origem, destino, cliente, motorista e o veículo. Para facilitar o cálculo do valor do frete, foi adicionado um botão com símbolo “$” que já calcula o total do frete com base no peso da carga e o valor por tonelada. A decisão de informar o peso em quilogramas (KG) e o valor ser por tonelada foi tomada com base nos estudos de campo realizado durante o planejamento do software. Na tela de cadastro do frete, os campos já trazem um indicativo de que informação deve ser preenchida, facilitando assim o entendimento do usuário.

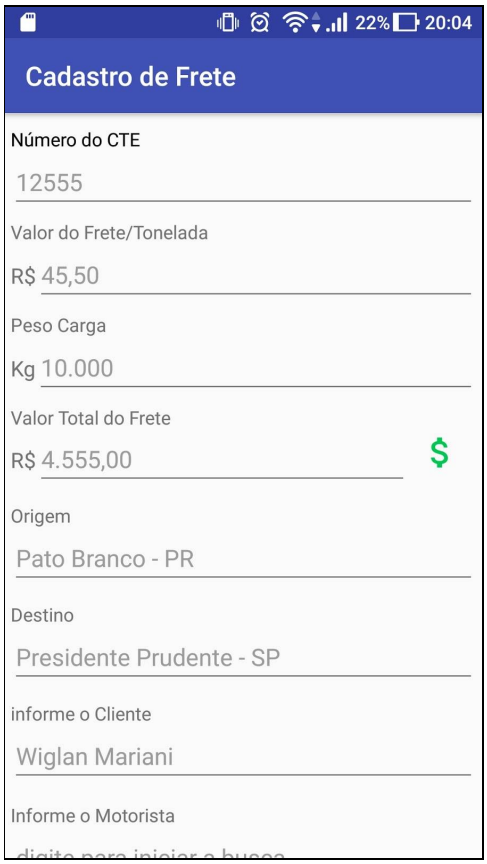

Figura 12 – Cadastro de frete

No cadastro do Frete, para informar qual o motorista e qual o veículo será utilizado, fez-se uso do componente AutoCompleteTextView. Com ele, ao digitar a primeira letra, o aplicativo já traz uma listagem dos possíveis resultados que podem ser utilizados, proporcionando maior agilidade no lançamento. A Figura 13 apresenta o componente e também os Botões de Salvar e Cancelar o lançamento de cadastro.

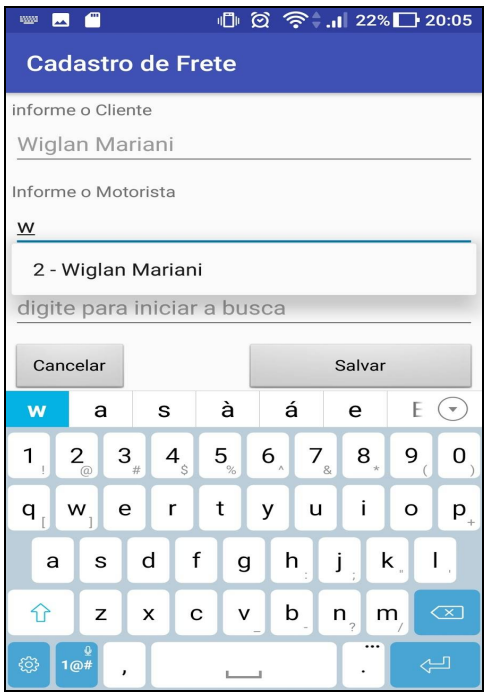

Figura 13 – Componente TextAutoComplete e botões do cadastro

Tendo como base que pelo menos um frete foi lançado, e que este está sendo apresentado na listagem de fretes conforme apresentado na Figura 9, a tela de listagem de lançamentos do frete é apresentada. O acesso a tela de listagem de lançamentos somente é possível pelo clique sobre um frete apresentado na listagem de fretes. A Figura 14 apresenta a tela de Listagem dos Lançamentos.

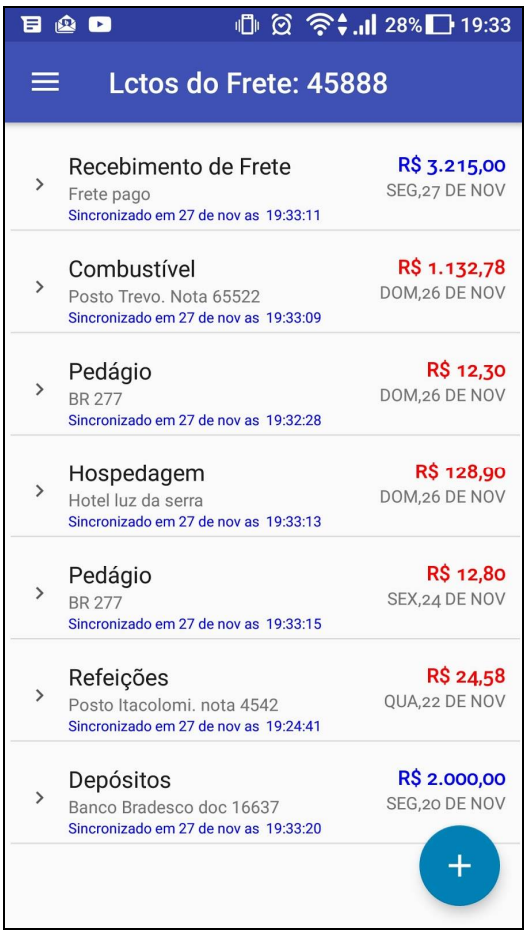

Figura 14 – Listagem dos lançamentos do frete

Na Figura 14, o título da tela faz referência ao frete do qual estão sendo consultados os lançamentos. Nesta listagem é, ainda, apresentado um botão flutuante no canto inferior direito, o qual dá acesso ao cadastro de um novo lançamento. Cada item representa um lançamento e ao ser clicado o seu cadastro é exibido em forma de edição. Esses lançamentos podem ser de receitas ou despesas, caso seja uma receita, o valor será apresentado em azul e caso seja uma despesa, o valor será apresentado em vermelho.

A Figura 15 apresenta um lançamento específico de despesa.

Figura 15 – Exemplo de um lançamento de despesa

Para cada lançamento é apresentada a categoria, uma observação que pode ser cadastrada no momento do lançamento, o *status* de sincronização com o servidor, o valor do lançamento e para qual data ele foi feito.

Por fim, a Figura 16 apresenta a tela de cadastro para um novo lançamento ou a edição de um já existente. Nessa tela é utilizado o mesmo componente de seleção de data utilizado para definir o encerramento do frete, apenas acrescentando a informação do horário do lançamento. Também deve ser informado o valor, a categoria e uma observação caso necessário para o lançamento.

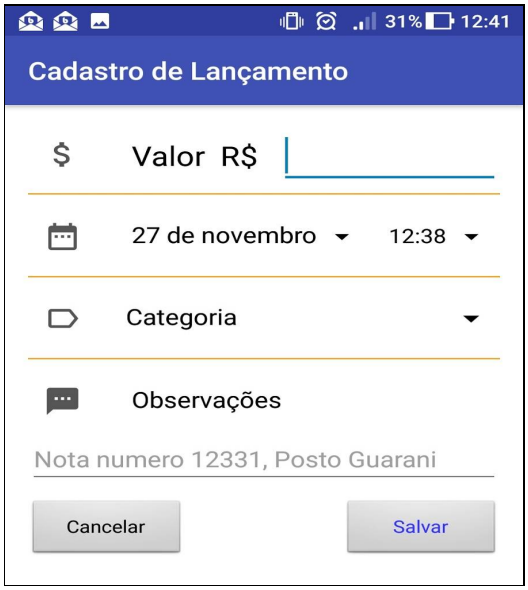

Figura 16 – Cadastro de lançamento

Um detalhe importante é que para a seleção da categoria foi criada uma nova listagem diferente da listagem das categorias cadastradas. Isso porque, tornou-se necessário apresentar uma listagem de forma que ocupasse toda a tela do dispositivo, apresentando as categorias em ordem de receitas e despesas, de forma que ao clicar em uma categoria, retornasse para tela de cadastro de lançamento como a categoria selecionada. Outro requisito era que uma nova categoria pudesse ser incluída sem a necessidade de fechar a tela de lançamento e abrir a listagem padrão de categorias para ter acesso ao botão de nova categoria. Para cumprir tais requisitos foi implementada uma lista personalizada fazendo uso do BaseAdapter, o qual permitiu criar uma listagem personalizada das categorias, incluindo a opção de novo cadastro. Ao clicar na categoria informada na tela de cadastro de lançamento, a listagem personalizada de categorias é apresentada ao usuário, conforme Figura 17.

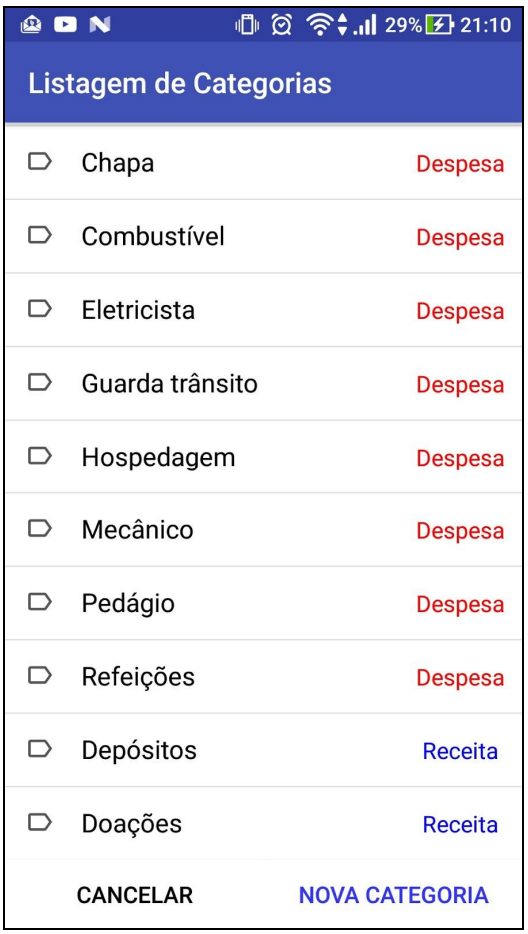

Figura 17 – Listagem personalizada de categorias com opção de novo registro

3.3.2 Servidor de Requisições

O servidor de requisições é responsável por receber as chamadas do aplicativo e respondê-las. Toda vez que o usuário realizar um novo cadastro ou editar um já existente, será executada uma chamada ao servidor passando como parâmetro um JSON com os dados do objeto em questão, que pode ser uma categoria, um frete ou qualquer outro registro. Recebendo essa chamada, o servidor irá persistir os dados no banco de dados PostgreSQL e retornar um objeto JSON com a resposta da requisição. Como o servidor de requisições não possui uma interface gráfica com o usuário, sendo apenas um serviço.

3.3.3 Portal de Consulta

O portal de consulta tem como objetivo a apresentação das informações lançadas pelo celular após elas serem sincronizadas com o servidor. No portal é possível visualizar a listagem de motoristas, veículos, categorias, fretes e seus lançamentos, conforme apesenta a Figura 18.

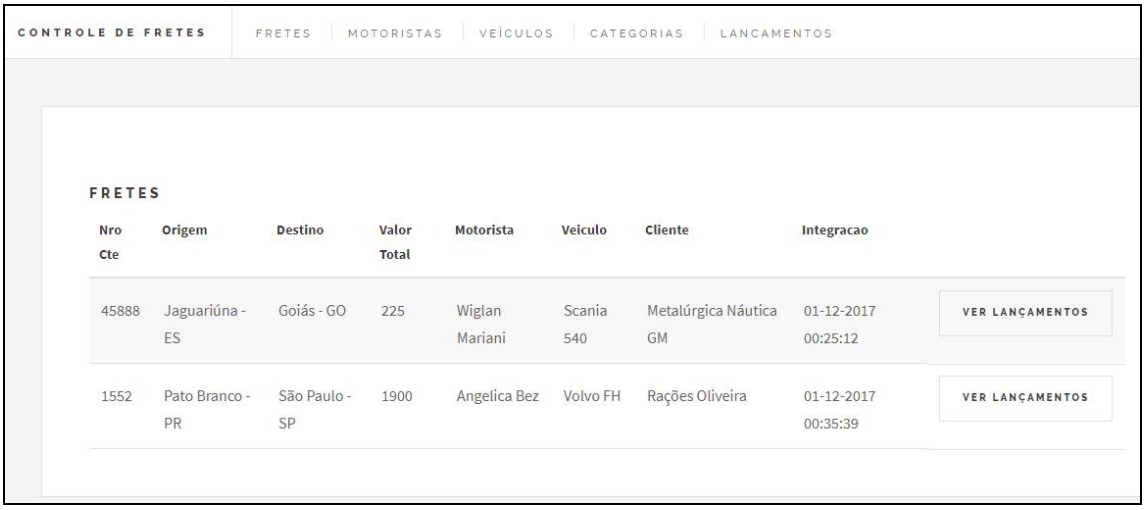

Figura 18 – Portal de consulta web

Ao abrir o portal a primeira página já apresenta os fretes que estão sincronizados com o servidor. Nesta listagem são apresentados os principais campos que identificam o frete e no lado direito de cada registro, um botão com o indicativo “VER LANÇAMENTOS” é apresentado. Quando clicado, esse botão atualiza a página apresentado todos os lançamentos disponíveis no servidor para o frete selecionado.

A Figura 19 apresenta os lançamentos de um frete. Para cada lançamento são apresentadas informações como, data do lançamento, hora, categoria, valor, observação e um campo indicando qual a data e horário que o registro foi sincronizado com o servidor.

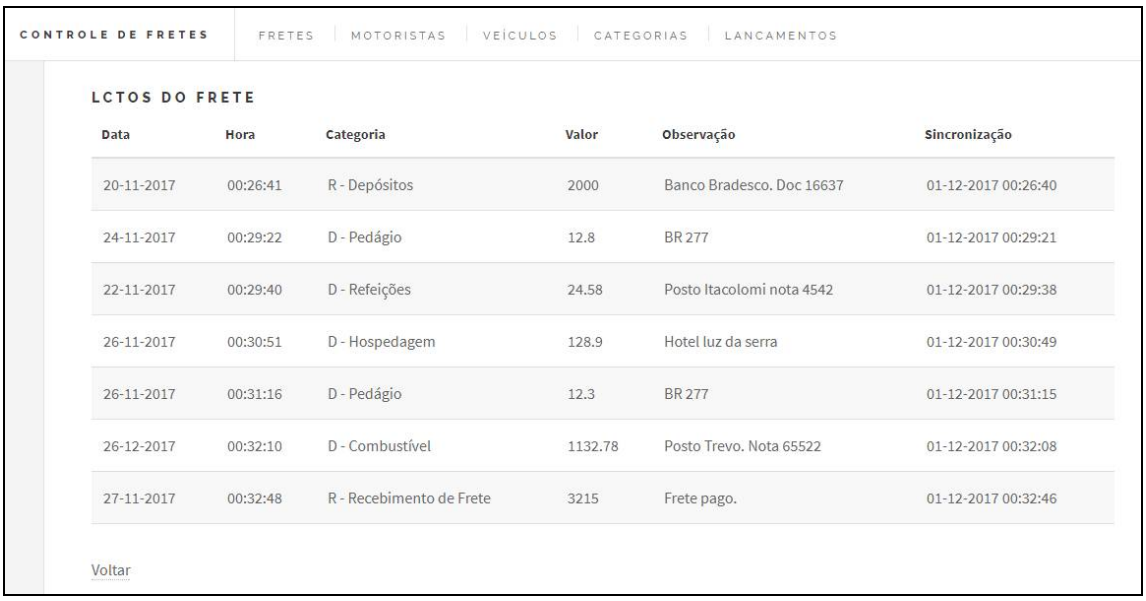

Figura 19 – Lançamentos de um frete

3.4 IMPLEMENTAÇÃO DO SISTEMA

Nesta sessão são apresentados os principais trechos de códigos utilizados para o desenvolvimento do projeto. Para facilitar o entendimento, foi dividido em três partes: aplicativo Android, servidor de requisições e portal de consulta.

3.4.1 Aplicativo Android

A Figura 20 apresenta a estrutura do projeto do aplicativo, separada em pacotes, que serão apresentados na sequência.

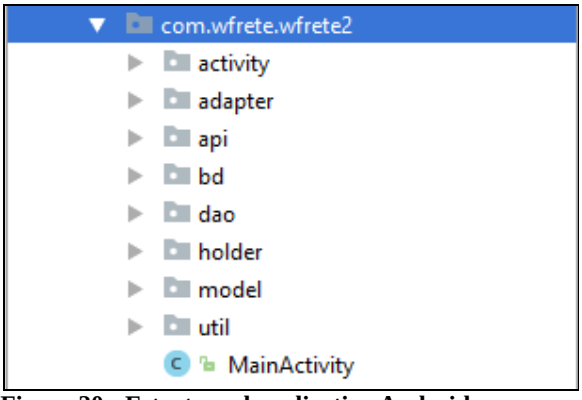

Figura 20 - Estrutura do aplicativo Android

O pacote “activity” apresenta todas as classes que manipula as telas apresentadas ao usuário. Nessas classes são codificadas todas as regras de Negócio e validações.

O pacote “adapter” concentra as classes responsáveis pelas listagens do aplicativo, como, por exemplo, a lista de categorias, de lançamentos e de fretes. Essas classes implementam o componente RecyclerView.Adapter, apresentado posteriormente. Elas são necessárias para construir listagem personalizadas proporcionado um leiaute diferenciado ao usuário.

No pacote “api” são concentradas as classes responsáveis pela comunicação e integração com o servidor de requisições. Toda lógica de integração de dados e utilização do *framework* Retrofit estão neste pacote.

O pacote “bd” apresenta as classes responsáveis pela criação do banco de dados e a disponibilidade de acesso ao banco.

O pacote “dao” concentra as classes que buscam as informações no banco de dados e devolvem em forma de objetos ou listagens. Nessas classes estão também as funções de manipulação de dados no banco, como, por exemplo, *insert*, *update* e *delete*.

O pacote “holder” apresenta as classes que interligam as listagens personalizadas do componente RecyclerView com as classes do pacote “adapter”, pois para cada *adapter* é necessário ter um *holder* correspondente. Nesses *holders* são mapeados os componentes visuais apresentados ao usuário.

O pacote “model” representa as classes cujas são um espelho do banco de dados. Nessas classes estão os métodos *getters* e *setters*.

O pacote “util” armazena classes que são utilizadas por todo o projeto, como, por exemplo, a classe Funções e Constantes, as quais armazenam códigos que são chamados por demais classes do projeto. Desta forma, mantém-se o projeto organizado e de fácil entendimento.

Por último está a classe MainActivity, que tem função de iniciar o aplicativo e fazer os primeiros apontamentos, como, por exemplo, apresentar a tela de Bem Vindo e apresentar a tela de listagem de fretes.

A Figura 21 apresenta todas as classes do projeto.

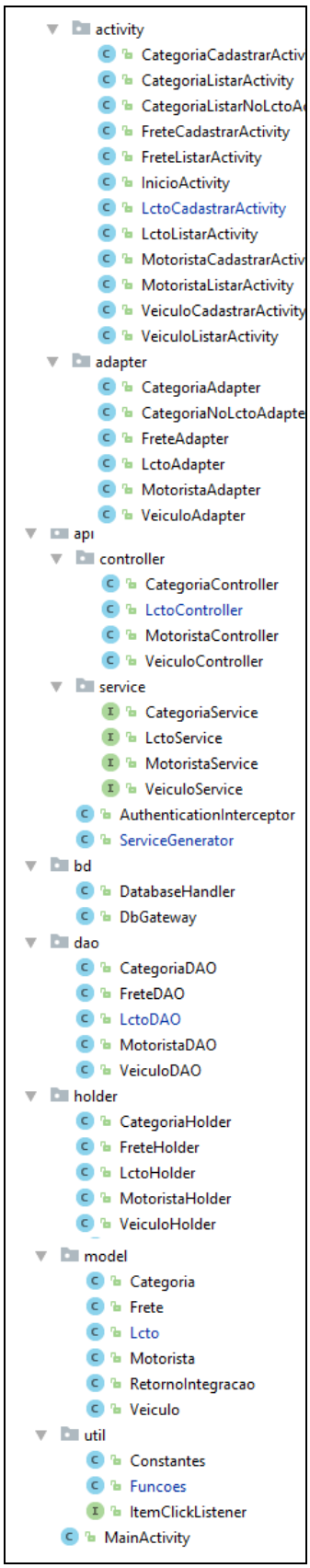

Figura 21 – Listagem completa de classes do aplicativo Android

Para facilitar o entendimento, os principais recursos utilizados para o desenvolvimento do aplicativo serão apresentados em tópicos descrito a seguir.

3.4.1.1 Utilização de Design Pattern Singleton

Design Pattern Singleton visa manter apenas uma instância de determinada classe, mantendo um ponto global de acesso ao objeto. No presente projeto, esse padrão é utilizado para garantir que o objeto de acesso ao banco de dados seja sempre o mesmo, mantém-se, assim, apenas uma conexão com o SQLLite (banco de dados do aplicativo). Desta forma, evita-se manter múltiplas conexões proporcionando maior segurança e melhor desempenho ao aplicativo.

A Listagem 1 apresenta a Classe “DbGatway” , na qual é aplicado o padrão Singleton, ou seja, sempre que o objeto “gw” já estiver instanciado ele é retornado.

```
//Cliente do banco de dados.
public class DbGateway {

    private static DbGateway gw;
    private SQLiteDatabase db;

    private DbGateway(Context ctx){
        DatabaseHandler handler = new DatabaseHandler(ctx);
        db = handler.getWritableDatabase();
    }

    public static DbGateway getInstance(Context ctx){

        //Design Pattern Singleton
        //aqui sempre utiliza a mesm conexao com o banco da dados, para
evitar que o app trablhe com multiplas conexoes simultaneas.
        if(gw == null)
            gw = new DbGateway(ctx);
        return gw;
    }

    public SQLiteDatabase getDatabase(){
        return this.db;
    }

}
```

Listagem 1 – Classe DbGatway: uso de design pattern Singleton

3.4.1.2 Utilização de Design Pattern Data Access Object

Design Pattern *Data Access Object* (DAO) define que se tenha uma camada de abstração de dados, encapsulado o acesso a estes dados de forma que as demais classes não precisem saber sobre como isso é feito.

Neste projeto, esse padrão pode ser visto no pacote “DAO” apresentado anteriormente, no qual toda abstração de dados (acesso ao banco de dados, operações de inserção, edição e exclusão) são realizadas pelas classes DAO. Desta forma, todo lugar que esses objetos são como listagem de dados, comparações e validações, utilizam a classe DAO como origem dos dados.

A Listagem 2 apresenta as funções da Classe MotoristaDAO, como exemplo ao padrão DAO. Os códigos de cada método foram removidos, pois o objetivo desta listagem é apresentar a estrutura e organização da classe, com seus métodos.

```
//Design Pattern Data Access Object
//Fornece uma interface para que as as camadas de aplicação se comuniquem
com o datasource.
public class MotoristaDAO {

    private DbGateway gw;

    public MotoristaDAO(Context ctx){}

    public boolean salvar(String nome, String cpf, String telefone, int
s_id, Date s_datahora){}

    public boolean salvar(int id, String nome, String cpf, String
telefone,int s_id, Date s_datahora){}

    public boolean salvar_dados_integracao(int id, int s_id, String
s_datahora){}

    public boolean excluir(int id){}

    public List<Motorista> ListarMotoristas(){}

    public Motorista retornarUltimo(){}

    public Motorista motoristaById(int idPesquisa){}

    public boolean existeVinculos(int motoristaId) {}
}
```

Listagem 2 – Classe MotoristaDao: uso de design pattern Data Acess Object

3.4.1.3 RecyclerView

Com objetivo de proporcionar um leiaute melhor do aplicativo, buscou-se um componente que permitisse a personalização de listagens. Neste projeto utiliza-se o RecyclerView, componente que substitui o ListView. Este componente foi fundamental para que fossem criadas as listagens de cadastros com botões para editar e excluir os registros e, principalmente, para as listagens de fretes e seus lançamentos. Para sua utilização, três passos são necessários.

Primero deve-se criar um arquivo .xml com o leiaute desejado para a listagem. Esse leiaute representa um item da lista. O arquivo *eXtensible Markup Language* (XML) criado originou o item de cada lançamento do frete, como apresenta a Figura 22.

Figura 22 – Item do lançamento

O segundo passo consiste em criar uma classe que mapeia os componentes listados no XML criado no passo 1. Esse mapeamento é necessário para que o componente saiba onde devem ser atribuídas as informações. Neste projeto o mapeamento dos arquivos XML que contém o leiaute utilizado para o RecyclerVIew estão armazenados no pacote Holder.

Na listagem 3 é apresentado o holder do lançamento.

```
//classe que mapeia os itens do RecyclerView

public class LctoHolder extends RecyclerView.ViewHolder implements
View.OnClickListener, View.OnLongClickListener{

    public TextView txtCategoria;
    public TextView txtObs;
    public TextView txtValor;
    public TextView txtData;
    public TextView txtS_DataHora;

    private ItemClickListener itemClickListener;

    public LctoHolder(View itemView) {
        super(itemView);
        txtCategoria = (TextView)
itemView.findViewById(R.id.txtCateogiraItem_Lcto);
        txtData = (TextView) itemView.findViewById(R.id.txtDataItemLcto);
        txtObs = (TextView) itemView.findViewById(R.id.txtObsItemLcto);
        txtValor = (TextView)
itemView.findViewById(R.id.txtValorItemLcto);
        txtS_DataHora = (TextView)
itemView.findViewById(R.id.txtDataHoraIntegracaoLcto);

        itemView.setOnClickListener(this);
```

```
        itemView.setOnLongClickListener(this);
    }

    public void setItemClickListener(ItemClickListener itemClickListener){
        this.itemClickListener = itemClickListener;
    }

    @Override
    public void onClick(View view) {
        itemClickListener.onClick(view, getAdapterPosition(), false);
    }

    @Override
    public boolean onLongClick(View view) {
        itemClickListener.onClick(view, getAdapterPosition(), true);
        return true;
    }
}
```

Listagem 3 – Classe LctoHolder: responsável pelo mapeamento dos componentes no arquivo XML de laiaute

Por fim, no terceiro passo, é necessário criar um *adapter* para a listagem dos lançamentos. Esse *adapter* faz um *extends* de um *holder*, que neste caso é o LctoHolder. Quando se faz o *extends* de um *holder* é necessário implementar dois métodos, o OnCreateViewHolder e o OnBindViewHolder. No primeiro método é retornado um item de lançamento criado no passo 1 e no segundo método cada componente é alimentado com o valor correspondente. A Listagem 4 apresenta a implementação desses dois métodos.

```
@Override
public LctoHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    return new LctoHolder(LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item_lcto, parent, false));
}

@Override
public void onBindViewHolder(LctoHolder holder, final int position) {

    Categoria categoria =
categoriaDAO.categoriaById(lctos.get(position).getCategoria_id());

    String str =
Funcoes.formataMsgIntegracao(lctos.get(position).getS_datahora());
    if (str.equalsIgnoreCase("Não Sincronizado")){
        holder.txtS_DataHora.setTextColor(Color.RED);
    }else{
        holder.txtS_DataHora.setTextColor(Color.BLUE);
    }
    holder.txtS_DataHora.setText(str);

    holder.txtCategoria.setText(categoria.getNome());
    holder.txtObs.setText(lctos.get(position).getObservacao());

    holder.txtValor.setText("R$ " +
df.format(lctos.get(position).getValor()));
    if (categoria.getTipo().equalsIgnoreCase("Receita")){
        holder.txtValor.setTextColor(Color.BLUE);
    }else {
```

```
            holder.txtValor.setTextColor(Color.RED);
        }

        String dateStr = dateFormat.format(lctos.get(position).getData());
        dateStr = dateStr.replace("-"," DE ");
        dateStr = dateStr.toUpperCase();
        holder.txtData.setText(dateStr);

        final Lcto lcto = lctos.get(position);

        holder.setItemClickListener(new ItemClickListener() {
            @Override
            public void onClick(View view, int position, boolean isLongClick) {
                if((context) instanceof LctoListarActivity){
                    ((LctoListarActivity)context).btEditarLctoOnClick(lcto);
                }
            }
        });
    }
```

Listagem 4 – Métodos implementados responsáveis por atribuir o conteúdo de um objeto lançamento a um item da listagem de lançamentos

3.4.1.3 Base Adapter

Outra forma de personalizar listagens é utilizando o recurso do BaseAdapter, com ele é possível criar listagens personalizadas muito semelhantes ao RecyclerView. Ele foi necessário para este projeto para a listagem das categorias no momento em que um lançamento está sendo realizado, isso por que não foi possível listar as categorias em tela cheia, com botões de no rodapé da listagem utilizando o RecyclerView. Essa listagem só foi possível codificando um novo *adapter* que implementa os métodos do BaseAdapter.

A Listagem 5 apresenta a classe CategoriaNoLctoAdapter, a qual herda da classe BaseAdapter implementando seus métodos obrigatórios.

```
Public class CategoriaNoLctoAdapter extends BaseAdapter {

    private final List<Categoria> categorias;
    private Activity act;

    public CategoriaNoLctoAdapter(Activity act, List<Categoria> categorias)
{
        this.act = act;
        this.categorias = categorias;
    }

    @Override
    public int getCount() {
        return categorias.size();
    }

    @Override
    public Object getItem(int i) {
```

```
        return categorias.get(i);
    }

    @Override
    public long getItemId(int i) {
        return categorias.get(i).getId();
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        View view =
act.getLayoutInflater().inflate(R.layout.item_categoria_no_lcto, parent,
false);
        Categoria categoria = categorias.get(position);

        TextView txtNome = (TextView)
view.findViewById(R.id.txtItem_categoria_nome);
        TextView txtTipo = (TextView)
view.findViewById(R.id.txtItem_categoria_tipo);

        txtNome.setText(categoria.getNome());
        txtTipo.setText(categoria.getTipo());


        if (categoria.getTipo().equalsIgnoreCase("Receita")){
            txtTipo.setTextColor(Color.BLUE);
        }
        else {
            txtTipo.setTextColor(Color.RED);
        }

        return view;
    }
}
```

Listagem 5 – Classe que estende as funções do base Adapter para montagem de uma lista personalizada utilizando o ListView

3.4.1.3 Retrofit

O Retrofit é um *framework* que facilita a execução de chamadas HTTP. Esse *framework* foi essencial para a troca de mensagens entre o servidor e o aplicativo. Primeiro foi necessário criar uma interface que declara quais as funções serão disponibilizadas e que tipo de requisição será feita no servidor, como é apresentado na Listagem 6 a interface CategoriaService.

```
public interface CategoriaService {

    @GET("/categoria/")
    Call<List<Categoria>> getAll();

    @GET("/categoria/{id}")
    Call<Categoria> getOne(@Path("id") Integer id);

    @POST("/categoria/")
```

```
    Call<RetornoIntegracao> save(@Body Categoria categoria);

    @PUT("/categoria/")
    Call<RetornoIntegracao> update(@Body Categoria categoria);

    @DELETE("/categoria/{id}")
    Call<Void> delete(@Path("id") Integer id);
}
```

Listagem 6 – Interface CategoriaService: declaração dos métodos disponíveis para troca de mensagem com o servidor.

Posterior à criação da interface, foi criada uma classe responsável por executar as chamadas ao Servidor. Essa classe também trata das respostas retornadas pelo servidor. A Listagem 7 apresenta a codificação completa da classe CategoriaController.

```
public class CategoriaController {

    public static void wsSalvarCategoria(Context context,  Categoria
categoria){

        int localizadorCategoriaWS = 0;
        int idLocalCategoria = categoria.getId();

        if (Funcoes.internetAtiva(context)){

            try {
                CategoriaService categoriaService =
ServiceGenerator.createService(CategoriaService.class, "TOKEN");

                RetornoIntegracao retornoIntegracao;
                Call<RetornoIntegracao> call;

                //novo categoria
                if (categoria.getS_id() <= 0){
                    localizadorCategoriaWS = Funcoes.getRandomInt() * -1;
                    categoria.setS_id(localizadorCategoriaWS);

                    call = categoriaService.save(categoria);

                }
                //se ja tem um id, ja foi integrado, entao edita o
registro.
                else {

                    //muda o id local pelo id do servidor para chamar a
edicao la
                    categoria.setId(categoria.getS_id());
                    call = categoriaService.update(categoria);

                }
                //executar no serivodr
                retornoIntegracao = call.execute().body();

                //se salvou no servidor, salvar os dados de integracao.
                CategoriaDAO dao = new CategoriaDAO(context);

                if (retornoIntegracao.getDthr_integracao() != null){
                    dao.salvar_dados_integracao(idLocalCategoria,
```

```
                                retornoIntegracao.getId_gerado(),
retornoIntegracao.getDthr_integracao());
                }

            }catch (Exception ex){
                ex.printStackTrace();
            }
        }
    }
}
```

Listagem 7 – Classe CategoriaController: responsável por executar as chamadas ao servidor

Primeiramente é realizada uma verificação se no momento da chamada de integração o dispositivo tem conexão de Internet disponível ou não. Caso tenha conexão, é avaliado se o objeto que será enviado para o servidor já tem um ID gerado no servidor, caso tenha, esse objeto será enviado em modo de edição e caso não tenha, esse objeto será enviado em modo de inserção. Após receber uma resposta do servidor, se ocorreu tudo certo com a integração, o registro local é alterado com o ID gerado no servidor e a data de integração.

Este padrão de integração é aplicado a todos os objetos do projeto, como por exemplo os motoristas, veículos, fretes, lançamentos, etc.

3.4.1.3 Classe de constantes e Funções

Com o objetivo de padronizar os códigos e de preservar uma boa organização foram criadas as classes de “Funções” e “Constantes”.

A classe de Constantes, conforme apresenta a Listagem 8, tem definido identificações que são utilizadas em várias outras classes, as quais identificam operações realizadas. Desta forma, essas constantes foram colocadas em um único lugar, evitando que cada classe existisse um valor diferente, o que poderia tornar a programação confusa e sujeita a falhas.

```
public class Constantes {
    //para os cadastros
    public static final int ID_COM_NOVO_REG_INSERIDO = 2;
    public static final int ID_COM_REG_ALTERADO = 5;
    public static final int ID_COM_CANCELADO = 9;
    public static final int ID_COM_REG_DELETADO = 13;
    public static final int ID_COM_LISTAR = 0;

    //para as consultas
    public static final int ID_COMANDO_NOVO_CADASTRO = 1;
    public static final int ID_COMANDO_EDITAR_REG = 7;

    //base adapter
    public static final int ID_COM_CADASTRAR_NOVO = 1;
    public static final int ID_COM_REG_SELECIONADO = 22;
```

```
    //nome das tabelas do banco.
    public static final String TABLE_CATEGORIAS = "Categorias";
    public static final String TABLE_FRETES = "Fretes";
    public static final String TABLE_LCTO = "Frete_Lcto";
    public static final String TABLE_MOTORISTAS = "Motoristas";
    public static final String TABLE_VEICULOS = "Veiculos";
}
```

Listagem 8 – Classe de constantes

Já a classe Funções, tem em si métodos que são chamados em diferentes lugares e também objetos de formatações como data e hora. Por exemplo, em cada classe de integração com o servidor, é necessário verificar a conexão com a Internet, desta forma criou-se um método de verificação de Internet na classe funções, mantendo assim o código organizado e mais seguro. A Listagem 9 apresenta a codificação da classe funções.

```
public class Funcoes {

    public static SimpleDateFormat dateFormatIntegracao;

    private static SimpleDateFormat timeFormat;

    static {
        dateFormatIntegracao = new SimpleDateFormat("dd-MM-yyyy
HH:mm:ss");
        timeFormat = new SimpleDateFormat("dd-MMM/ HH:mm:ss");
    }

     public static Date gerarData(Date dataLcto, Time horaLcto){

        int ano;
        int mes;
        int dia;
        int minuto;
        int hora;
        int segundo;

        ano = mes = dia = minuto = hora = segundo = 0;

        if (dataLcto != null) {
            Calendar calendar = Calendar.getInstance();
            calendar.setTime(dataLcto);
            ano = calendar.get(Calendar.YEAR);
            mes = calendar.get(Calendar.MONTH);
            dia = calendar.get(Calendar.DAY_OF_MONTH);
        }
        if (horaLcto != null){
            Calendar calendar = Calendar.getInstance();
            hora = calendar.get(Calendar.HOUR_OF_DAY);
            minuto = calendar.get(Calendar.MINUTE);
            segundo = calendar.get(Calendar.SECOND);
        }

        Calendar calendar = Calendar.getInstance();
        calendar.set(ano, mes, dia, hora, minuto, segundo);

        return calendar.getTime();
    }
```

```
    public static boolean internetAtiva(Context context) {
        boolean conectado;
        ConnectivityManager conectivtyManager = (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);
        if (conectivtyManager.getActiveNetworkInfo() != null
                && conectivtyManager.getActiveNetworkInfo().isAvailable()
                && conectivtyManager.getActiveNetworkInfo().isConnected())
{
            conectado = true;
        } else {
            conectado = false;
        }
        return conectado;
    }

    public static int getRandomInt() {
        Random random = new Random();
        return random.nextInt((99999999 - 999) + 1) + 999;
    }


    public static String formataMsgIntegracao(Date data){

        if (data != null){
            String str = "Sincronizado em " + timeFormat.format(data);
            str = str.replace("-", " de ");
            str = str.replace("/", " as ");
            return str;
        }
        else {
            return "Não Sincronizado";
        }
    }
}
```

Listagem 9 – Classe de funções

3.4.2 Servidor de Requisições

Para o desenvolvimento do servidor foi utilizada a IDE Eclipse, juntamente com alguns pacotes de funções, como o pacote do SpringBoot.

O servidor desenvolvido tem como principal função atender as chamadas do aplicativo Android, persistir os dados que chegam até ele e retornar as listagens dos dados solicitadas pelo portal *web*.

O projeto do servidor foi desenvolvido no modelo *Model-View-Controller* (MVC), tendo classes específicas para modelagem, para visualização e controle. A Figura 23 apresenta a estrutura do projeto, os pacotes e suas respectivas classes, distribuídas conforme a função de cada uma.

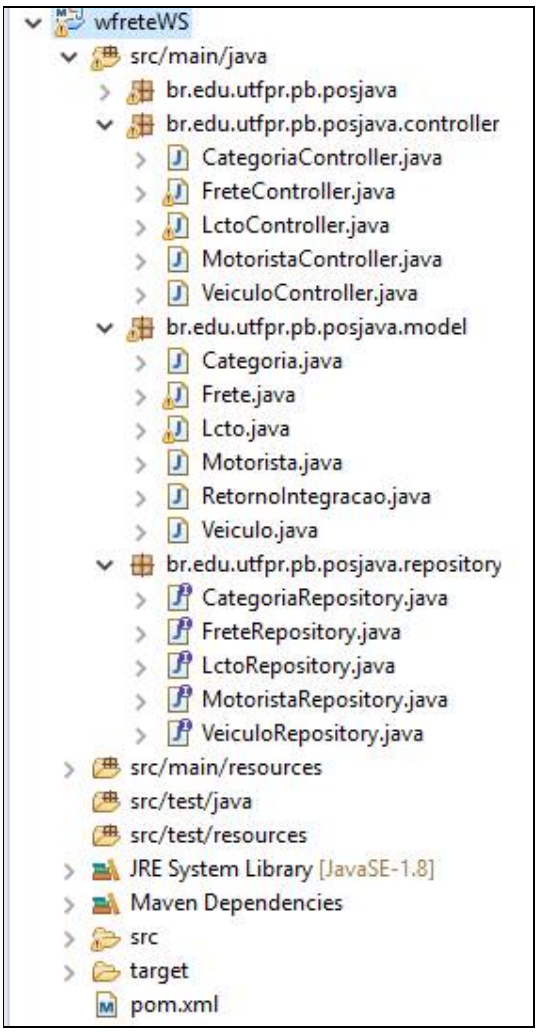

Figura 23 – Estrutura do projeto do servidor

O pacote controller contém as classes que respondem às requisições efetuadas pelo aplicativo e pelo portal. Por meio dos seus métodos é definido o *model* que cada requisição utiliza para trabalhar com os dados que chegam até ela.

O pacote model contém as classes que representam os modelos de objetos do sistema. Essas classes são basicamente um espelho do banco de dados.

No pacote repository são criadas as classes que utilizam o *framework Java Persistence API* (JPA) para persistência e listagem dos dados armazenados no banco de dados.

As principais funcionalidades estão concentradas nas classes de controller. A Listagem 10 apresenta a codificação da classe MotoristaController.

```
@CrossOrigin
@RestController
@RequestMapping("/motorista")
public class MotoristaController {

	@Autowired
```

```
	private  MotoristaRepository motoristaRepository;

	@GetMapping("/")
	public List<Motorista> listar(){
		List<Motorista> motoristas = motoristaRepository.findAll();
		return motoristas;
	}

	@GetMapping("/{id}")
	public Motorista buscar(@PathVariable Integer id){
		return motoristaRepository.findOne(id);
	}


	@GetMapping("/localizador/{localizador}")
	public Motorista buscar2(@PathVariable Integer localizador){
		return motoristaRepository.findByLocalizador(localizador);
	}

	@PostMapping("/")
	public RetornoIntegracao inserir(@RequestBody Motorista motorista){

		RetornoIntegracao r = new RetornoIntegracao();

		motorista.setS_datahora(Util.getdataHoraAtual());
		motorista.setId(null);

		motoristaRepository.save(motorista);


		r.setId_gerado(motorista.getId());

		Date data = motorista.getS_datahora();
		SimpleDateFormat formataData = new SimpleDateFormat("dd-MM-yyyy
HH:mm:ss");
		String datastr = formataData.format(data);
		r.setDthr_integracao(datastr);

		r.setMsg("OK");

		System.out.println("INSERT- " + r.toString());

		return r;

	}

	@PutMapping("/")
	@ResponseStatus(value = HttpStatus.OK)
	public RetornoIntegracao editar(@RequestBody Motorista motorista){

		motorista.setS_datahora(Util.getdataHoraAtual());
		motoristaRepository.save(motorista);

		RetornoIntegracao r = new RetornoIntegracao();
		r.setId_gerado(motorista.getId());
		Date data = motorista.getS_datahora();
		SimpleDateFormat formataData = new SimpleDateFormat("dd-MM-yyyy
HH:mm:ss");
		String datastr = formataData.format(data);
```

```
                r.setDthr_integracao(datastr);
                r.setMsg("OK");

                System.out.println("EDICAO - " + r.toString());

                return r;
        }

        @DeleteMapping("/")
        @ResponseStatus(value = HttpStatus.OK)
        public void remover(@PathVariable Integer id){
                motoristaRepository.delete(id);
        }
}
```

Listagem 10 – Classe MotoristaController

Na Listagem 10 é possível perceber o uso da anotação @CrossOrign, com ela são disponibilizados as requisições para qualquer origem de chamador e, caso seja necessário, filtrar algum chamador específico, como limitar para que a requisição de inserir Motoristas no banco de dados somente seja possível via aplicativo Android, basta configurar a anotação em questão. A anotação @RestController indica ao *framework* que a classe em questão disponibiliza métodos padrões de requisições REST. Por fim, a anotação @RequestMapping(“/motorista”) indica qual *Uniform Resource Locator* (URL) será mapeada pela classe.

É importante destacar que para mapear cada tipo de chamada, uma anotação correspondente foi utilizada, como, por exemplo, a anotação @deleteMapping para exclusão de dados, anotações com @GetMapping para listagens e @PostMapping para inserções.

Uma situação encontrada é que não foi possível transferir via JSON campos do tipo Date. Para isso foi criado um campo auxiliar do tipo String e ao chegar a informação no servido, esta foi convertida para Date para que fosse persistida corretamente no banco de dados.

3.4.3 Portal de Consulta

O portal de consulta tem como objetivo realizar listagens dos dados cadastrados no servidor. Para isso, foi utilizado o Angular, juntamente com HTML e *Cascading Style Sheet* (CSS) para a montagem e a estilização das consultas.

Com a utilização do Angular, um arquivo de rotas foi criado para direcionar a navegação das páginas e ter controle de quais URLs podem ser navegadas. A Listagem 11

apresenta o arquivo de rotas criado, nele é possível perceber que para cada URL navegada é estabelecido uma página HTML que será apresentada ao usuário e um controle que é responsável pela gestão das ações que ocorrem dentro da página solicitada para navegação.

```
angular.module('fretecontrol', ['ngRoute'])
  .config(['$locationProvider', function($locationProvider) {
    $locationProvider.hashPrefix('');
  }])
  .config(function($routeProvider){
    $routeProvider
      .when('/', {
        templateUrl: 'src/frete/frete.html',
        controller: 'FreteCtrl'
      })
      .when('/lancamento/frete/:idfrete', {
        templateUrl: 'src/lcto/lcto.html',
        controller: 'LctoCtrl'
      })
      .when('/frete', {
        templateUrl: 'src/frete/frete.html',
        controller: 'FreteCtrl'
      })
      .when('/categoria', {
        templateUrl: 'src/categoria/categoria.html',
        controller: 'CategoriaCtrl'
      })
      .when('/motorista', {
        templateUrl: 'src/motorista/motorista.html',
        controller: 'MotoristaCtrl'
      })
      .when('/veiculo', {
        templateUrl: 'src/veiculo/veiculo.html',
        controller: 'VeiculoCtrl'
      })
      .otherwise({
        redirectTo: '/'
      });

  });
```

Listagem 11 – Arquivo de rotas

Com a utilização de diretivas do Angular, a apresentação de listagem de dados se torna mais simples. A Listagem 12 apresenta a listagem de motoristas. Inicialmente, foi criado um HTML para exibição dos dados em uma tabela e com a diretiva “ng-repeat”, percorrendo um objeto que representa a lista de motoristas. Para cada motorista da lista é repetida uma linha na tabela com os dados do motorista.

```
  <div>
    <br /> <h3>Motoristas</h3>
    <table>
      <thead>
          <tr>
             <th>Nome</th>
             <th>CPF</th>
```

```
            <th>Telefone</th>
            <th>Sincronização</th>
          </tr>
      </thead>
      <tbody>
        <tr ng-repeat="motorista in motoristas">
          <td>{{motorista.nome}}</td>
          <td>{{motorista.cpf}}</td>
          <td>{{motorista.telefone}}</td>
          <td>{{motorista.s_datahora}}</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

Listagem 12 – Página HTML de listagem dos motoristas.

Por fim, a Listagem 13 apresenta a chamada da requisição de listagem de motoristas que é realizada pelo portal para o servidor.

```
angular.module('fretecontrol')
  .controller('MotoristaCtrl', function($scope, $http){
    $scope.motoristas = [];

    $http.get('http://192.168.100.5:8021/motorista/')
      .then(function(response){
        $scope.motoristas = response.data;
      });
  });
```

Listagem 13 – Requisição de listagem de motoristas executada pelo portal de consulta

4 CONCLUSÃO

Este trabalho teve como resultado o desenvolvimento de uma solução *web* e *mobile* como forma de gestão para microempresas e autônomos na área de prestação de serviço de frete. A solução apresentada consiste em um aplicativo para a plataforma Android responsável por permitir o registro das informações de frete, como lançamentos de receitas e despesas que ocorrem durante a prestação do serviço. Um servidor de requisições responsável por receber os dados do aplicativo e persisti-los de forma que fiquem disponíveis para consulta quando desejado. E de um portal de consulta, com o objetivo de listar as informações presente no banco de dados.

Com o uso de tecnologias e ferramentas bem difundidas no mercado, não foram encontradas grandes dificuldades para o desenvolvimento da solução proposta, exceto a transferência de datas e horas entre o servidor e o aplicativo *mobile*. Não foi possível transferir os dados em sua forma nativa, sendo necessária a conversão destes dados para o tipo primitivo *string*. Desta forma, a comunicação entre o cliente e servidor funcionou corretamente.

Com base nos resultados apresentados é possível afirmar que o projeto atendeu as expectativas estabelecidas no escopo do sistema. É valido citar que a solução pode ser utilizada em sua totalidade ou de forma parcial, ou seja, o uso do aplicativo para registro das informações não precisa necessariamente de um servidor de integração de dados e um portal de consulta. Todas as movimentações realizadas no aplicativo podem ser consultadas e alteradas diretamente no aplicativo. Esta situação é válida para os transportadores autônomos que desejam ter uma forma de controle dos seus gastos e receitas que ocorrem na prestação de serviço, a fim de ter mais segurança e controle sobre sua atividade.

Como forma de melhorias, a implementação de um controle de usuário tanto para acesso ao portal quanto para o acesso ao aplicativo é fundamental, gerando assim maior controle sobre as operações realizadas.

Conclui-se que os propósitos estabelecidos foram alcançados por meio de uma solução que permite gerenciar de forma mais eficaz a prestação de serviço de frete, possibilitando o acesso à informação de forma mais ágil e segura, visando minimizar problemas antes existentes como volume de informações, desconfiança dos resultados e insegurança operacional.

REFERÊNCIAS

ANDROID STUDIO. **Conheça o Android Studio**. Disponível em: <https://developer.android.com/studio/intro/index.html?hl=pt-br>. Acesso em: 25 de novembro de 2017.

BOLETIN CNT. **Boletim estatístico Junho 2017**. Disponível em: <http://www.cnt.org.br/Boletim/boletim-estatistico-cnt>. Acesso em: 24 de novembro de 2017.

CANALTECH. **Os números não mentem**. Disponível em: <https://canaltech.com.br/produtos/os-numeros-nao-mentem-android-ou-ios-qual-e-o-melhor-7657/>. Acesso em 24 de novembro de 2017.

ANISZCZYK, Chris. **Introdução à plataforma Eclipse**. Disponível em: <https://www.ibm.com/developerworks/br/library/os-eclipse-platform> Acesso em: 25 de novembro de 2017.

SCHMITZ, Daniel **Tudo que você queria saber sobre Git e GitHub**. Disponível em: <https://tableless.com.br/tudo-que-voce-queria-saber-sobre-git-e-github-mas-tinha-vergonha-de-perguntar/>. Acesso em: 25 de novembro de 2017.

LIMA, Davi. **Bizagi Modeler**. Disponível em: <http://www.techtudo.com.br/tudo-sobre/bizagi-modeler.html>. Acesso em: 25 de novembro de 2017.

ALECRIM, Emerson. **Banco de dados MySQL e PostgreSQL**. Disponível em: <https://www.infowester.com/postgremysql.php>. Acesso em: 25 de novembro de 2017.

ESTATISTICAS ANTT, **Entre em números**, 2017. Disponível em: <http://portal.antt.gov.br/index.php/content/view/4969.html>. Acesso em: 24 de novembro de 2017.

EXAME. **Estatísticas de uso de celular no Brasil**, 2016. Disponível em: <https://exame.abril.com.br/negocios/dino/estatisticas-de-uso-de-celular-no-brasil-dino89091436131/>. Acesso em: 24 de novembro de 2017.

TECNOBLOG. **Smartphones vendidos no Brasil**. Disponível em:<https://tecnoblog.net/203749/android-ios-market-share-brasil-3t-2016/>. Acesso em: 24 de novembro de 2017.