CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

DESENVOLVIMENTO DE APLICATIVO DE GESTÃO VEICULAR PARA PLATAFORMA ANDROID

LEONARDO HENRIQUE MUNIZ CORNACHIONE

Americana, SP
Dezembro de 2014

CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

DESENVOLVIMENTO DE APLICATIVO DE GESTÃO VEICULAR PARA PLATAFORMA ANDROID

LEONARDO HENRIQUE MUNIZ CORNACHIONE
leonardo.henrique45@gmail.com

Trabalho Monográfico, desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Fatec-Americana, sob orientação do Prof. MSc. Wagner Siqueira Cavalcante.

Área: Aplicativo móvel

Americana, SP
Dezembro de 2014

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

C834d

Cornachione, Leonardo Henrique Muniz
Desenvolvimento de aplicativo de gestão veicular para plataforma Android. / Leonardo Henrique Muniz Cornachione. – Americana: 2014.
62f.

Monografia (Graduação em Tecnologia em Análise e Desenvolvimento de Sistemas). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Me.Wagner Siqueira Cavalcante

1. Dispositivos móveis – aplicativos 2. Computação em nuvem I. Cavalcante, Wagner Siqueira II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU:681.519
681.518

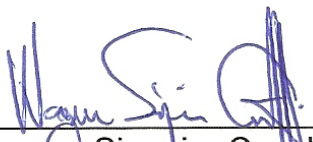
LEONARDO HENRIQUE MUNIZ CORNACHIONE

DESENVOLVIMENTO DE APLICATIVO DE GESTÃO VEICULAR PARA PLATAFORMA ANDROID

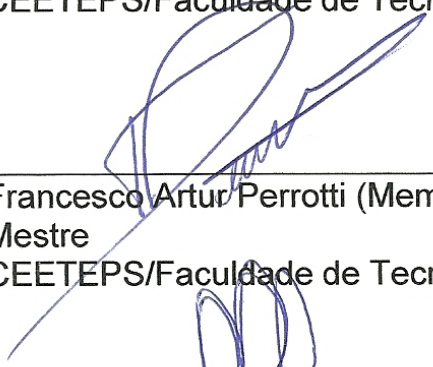
Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – Fatec/ Americana.
Área de concentração: Aplicativo Móvel.

Americana, 03 de Dezembro de 2014.

Banca Examinadora:



Wagner Siqueira Cavalcante (Presidente)
Mestre
CEETEPS/Faculdade de Tecnologia – FATEC Americana



Francesco Artur Perrotti (Membro)
Mestre
CEETEPS/Faculdade de Tecnologia – FATEC Americana

José William Pinto Gomes (Membro)
Especialista
CEETEPS/Faculdade de Tecnologia – FATEC Americana

AGRADECIMENTOS

Primeiramente, agradeço ao meu mentor e orientador, Professor MSc. Wagner Siqueira Cavalcante, por me guiar com paciência e dedicação no processo de pesquisa e desenvolvimento deste trabalho.

Agradeço a todos os colegas de sala, que, assim como o criador da obra, acreditam em um sonho e estão caminhando para que seja possível a realização, principalmente ao Guilherme Henrique Matioli que, com amizade, companheirismo e aconselhamento, esteve presente ao decorrer do curso ajudando principalmente nas horas mais difíceis.

Agradeço aos meus pais, Jair Cornachione e Albani Muniz Cornachione, por todo companheirismo e apoio, nas horas boas e principalmente nas horas mais difíceis, sempre me ajudando e sendo fundamentais para a realização deste trabalho.

Aos amigos e colegas de trabalho por serem candidatos a testadores do aplicativo desenvolvido.

Agradeço a todos aqueles que de alguma forma doaram um pouco de si para que a realização deste trabalho se tornasse possível.

DEDICATÓRIA

Dedico este trabalho aos meus pais, pelo que são e pelo apoio que deles sempre recebi. A Deus, que me deu forças e iluminou meu caminho durante toda esta longa caminhada, pois eu não seria nada sem a fé que tenho nEle.

RESUMO

Este trabalho consiste na criação de um aplicativo para a plataforma *Android*, compatível com as versões 4.0, com API nível 14, ou superior, que também demonstrará uma conexão, manipulação, persistência e consistência de dados com um serviço de banco de dados em um servidor alocado em nuvem. Neste trabalho será feita uma abordagem das tecnologias envolvidas, para a melhor absorção do que será utilizado para o desenvolvimento do aplicativo. O aplicativo desenvolvido, intitulado ECarManager, serve para o controle de finanças de automóveis, de forma que usuários possam ter cadastros únicos de veículos, de suas despesas, serviços e abastecimentos de combustíveis. O aplicativo também disporá de uma análise de gráficos por períodos diários, mensais e anuais dos abastecimentos, bem como o compartilhamento de informações de locais e preços de combustíveis através do Google Maps, o qual poderá traçar rotas até os locais demarcados por ele ou por outros usuários que também compartilharem informações pelo mapa. Serão abordadas as ferramentas utilizadas para criação do projeto como o Eclipse, plug-in ADT, SDK Manager, *MySQL*, Amazon, *HTTP* e *JSON*, fundamentais para o desenvolvimento. Com o ECarManager, o usuário terá maior controle de gastos gerados por seu veículo pessoal, possuindo um histórico que conseguirá acessar de qualquer dispositivo, será preciso somente conectar-se com sua conta e terá todas as informações do seu veículo, o que é possível pelo fato de o aplicativo compartilhar o mesmo banco de dados em nuvem.

Palavras Chave: ADT; SDK Manager; *MySQL*; Amazon; *HTTP*; *JSON*; Aplicativo; *Android*; Banco de Dados.

ABSTRACT

This work consists of creating an app for the Android platform, compatible with versions 4.0, API level 14 or higher who also demonstrate a connection, handling, persistence and consistency of data with a service database on a server allocated cloud. In this paper an approach will be made of the technologies involved, for better absorption of which will be used for application development. The developed application, called ECarManager serves to control the finances of cars, so that users can have unique registrations of vehicles, their costs, services and supplies of fuel. The application will also have a review of charts for daily, monthly and yearly periods of supplies as well as information sharing and local petrol prices using Google Maps, which can plot routes to the places marked by him or by others users also share information the map. The tools used to create the project like Eclipse ADT plugin, SDK Manager, MySQL, Amazon, HTTP and JSON, key development will be addressed. With ECarManager, the user will have greater control of expenses generated by your personal vehicle, having a history that will be able to access from any device, you only need to connect with your account and you will have all the information of your vehicle, which is possible by That the application share the same database in the cloud.

Keywords: *ADT; SDK Manager; MySQL; Amazon; HTTP; JSON; Application; Android; Data Base.*

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

ADT	*Android Development Tools*
ANP	Agência Nacional de Petróleo
API	*Application Programming Interface*
AVD	*Android Virtual Device*
CRUD	*Create, Read, Update, Delete*
HTC	*High-Tech Computer Corporation*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
OHA	*Open Handset Alliance*
PHP	*HiperText Processor*
SDK	*Software Development Kit*
SGBD	Sistema de Gerenciamento de Banco de Dados
URL	*Uniform Resource Locator*
Wi-Fi	*Wireless fidelity*
XML	*eXtensible Markup Language*

SUMÁRIO

INTRODUÇÃO

Segundo Taurion (2009), a imagem da nuvem representa a Internet e indica para onde todos os dados e requisições são direcionados. No início da década, mais precisamente em 2006, empresas como Google e Amazon criaram grandes prédios computacionais, organizados no conceito de nuvem para utilizar em seus próprios negócios. Desta forma, descobriu-se que poderiam criar novos serviços, deixando disponível essa capacidade de processamento para o mercado. Hoje, com a computação em nuvem, a imagem da nuvem representa algo mais amplo e disponibiliza recursos que auxiliam o funcionamento de uma aplicação em dispositivo local, como um *smartphone*, e pode ser destinada a armazenamento de informações.

Acrescenta Taurion (2009), que a computação em nuvem é composta por um conjunto de servidores, denominados *clusters*, que virtualizam recursos de tal forma, que cria-se uma ilusão de disponibilidade infinita, que podem ser virtuais ou físicos, disponibilizando aplicações, plataformas, armazenamento, e capacidade de processamento, disponibilizado como um serviço através da rede, que tem como características permitir que usuários utilizem recursos por demanda, diminuindo ou aumentando a capacidade computacional de forma dinâmica e por último, mas não menos importante, é que o pagamento pelos serviços é feito por demanda, ou seja, somente pelo que é consumido.

De acordo com Souza (2010), há meio século, servidores de tempos partilhados eram usados para compor recursos delimitados. Hoje, com a computação em nuvem, são disponibilizados recursos de infraestruturas complexas de TI, permitindo que usuários acessem as informações sem a necessidade de conhecer a tecnologia utilizada, independentemente da localização.

Para Abadi (2009), Sistemas de Gerenciamento de Banco de Dados (SGDB) são fortes candidatos para a nuvem. Isso se dá pelo motivo de que instalações de sistemas complexos que envolvem grandes quantidades de dados ocasionam altos investimentos de *hardware* e *software*. Para muitas companhias, especialmente as

novas no mercado, como *startups* e médias empresas, o *pay-per-use* em conjunto com o suporte e manutenção do hardware é muito atraente.

Salienta Abadi (2009), que as nuvens têm o potencial de atrair clientes de pequenas até grandes empresas de diversos setores, com o intuito de reduzir custos e solucionar problemas de grandes tráfegos inesperados na rede, seja ele por acesso a sites ou por armazenamento que utiliza infraestrutura de terceiros.

Para tanto, o estudo se **justificou** pela importância e vantagens da computação em nuvem utilizando SGDB. Conforme descrito por Abadi (2009), durante o desenvolvimento de aplicativos de tecnologia móvel para compartilhamento de informações, ao mesmo tempo aplicar ao projeto boas práticas de utilização de código fonte para conexão e manipulação do banco de dados para que seja acessado de qualquer dispositivo em qualquer lugar compartilhando informações via Internet.

O **problema** levantado refere-se a que durante o desenvolvimento do aplicativo nativo da plataforma *mobile*, desenvolvedores podem não fazer corretamente ou não utilizar as melhores práticas para uma conexão e manipulação de banco de dados via Internet, desta forma afetando diretamente o desempenho do banco de dados, o que gera manutenções indesejáveis, com falhas a curto prazo e problemas com atualizações de novas versões.

A **pergunta** que se pretende responder é se é possível obter consistência na utilização de banco de dados nas nuvens para desenvolvimento de aplicativo em plataforma *mobile*?

Três hipóteses foram levantadas, ou sejam: **a)** O aplicativo faz com que as informações fiquem disponíveis com estabilidade, garantindo escalabilidade e consistência de dados. Foi desenvolvido com conexão com banco de dados nas nuvens respeitando as premissas do projeto, e desta forma os riscos foram minimizados consideravelmente, resultando em um *software* com qualidade. **b)** Os riscos de se ter falhas no projeto ficarão muito maiores, pois a velocidade das consultas ao banco de dados via Internet poderá estar lenta, ou o aparelho *mobile* com baixa memória disponível, irá acarretar má qualidade e baixo desempenho do

aplicativo, comprometendo a consistência dos dados. **c)** O aplicativo será capaz de realizar manipulação de dados como *Create, Read, Update e Delete* (CRUD) com alto desempenho, desde que seja colocado em prática detalhadamente o que estiver especificado no projeto, tenha ótima conexão com a Internet e o dispositivo *mobile* tenha memória suficiente para executar o aplicativo. O desenvolvedor também tem que ser criterioso e extremamente prático em todas as etapas do desenvolvimento.

O **objetivo geral** consiste em estudar o modelo de armazenamento de dados nas nuvens disponibilizado pela Amazon e suas camadas de arquitetura e premissas de desenvolvimento para a plataforma *Android*, objetivando o desenvolvimento de um aplicativo *mobile* com conexão ao banco de dados para que possa ser usado em qualquer hora e em qualquer lugar via Internet através de um dispositivo *mobile* compartilhando informações entre usuários.

Os objetivos específicos são: **a)** Fazer um levantamento de técnicas de gerenciamento de projeto e funcionamento de banco de dados nas nuvens, visando o desenvolvimento de um projeto de banco de dados e o entendimento do método de virtualização de recursos e sua função na área de TI. **b)** Estudar os dados deste levantamento junto ao estudo da plataforma *Android*, para a instalação e configuração da IDE de desenvolvimento, objetivando a integração de um banco de dados relacional da Amazon para incluir, entre a aplicação nativa com o banco de dados nas nuvens, uma conexão via Internet para que seja desenvolvido um aplicativo mobile. **c)** Realizar testes de estudo do aplicativo, afim de verificar consistência de dados e funcionamento correto, e também o cumprimento dos objetivos.

Como **metodologia**, do ponto de vista da sua natureza, a pesquisa é aplicada para solucionar problemas específicos, como conexão e manipulação de banco de dados nas nuvens, envolvendo verdades e interesses locais.

Do ponto de **vista da forma de abordagem**, a pesquisa é qualitativa, uma vez que discutirá o problema do aplicativo, descrevendo por etapas como criar uma aplicação com conexão em banco de dados nas nuvens.

Do ponto de **vista dos objetivos,** a pesquisa é explicativa, de modo que irá explicar como desenvolver aplicações com conexão nas nuvens, bem como o motivo para se realizar tal desenvolvimento, e identificar melhores técnicas.

Do ponto de **vista dos procedimentos técnicos**, a pesquisa é bibliográfica, de modo que foram realizadas pesquisas em teses, livros e demais referências acadêmicas disponíveis em biblioteca e na Internet para a realização do trabalho, e é também uma pesquisa experimental, de modo que, ao finalizar o projeto, se tornam necessários os devidos testes para comprovar que a aplicação *mobile* funciona conforme foi planejado e discutido.

O trabalho foi estruturado em **quatro capítulos**, sendo que o **primeiro** conceitua as tecnologias abordadas para o desenvolvimento do projeto isso inclui o estudo da plataforma *Android*, funcionamento do banco de dados nas nuvens e escolha do SGDB, protocolo de comunicação *HTTP* e a linguagem de comunicação de dados *JSON*, o **segundo** descreve o desenvolvimento do aplicativo a partir de um estudo de caso, com isso identificando os passos do projeto**,** o **terceiro** é reservado para discussões e resultados da implementação, informando também o que se espera para as próximas versões e estudos sobre as plataformas.

Com base nos estudos realizados no capítulo anterior, o **quarto** capítulo se reserva às Considerações Finais.

1. TECNOLOGIAS ABORDADAS

Neste capítulo apresentam-se as tecnologias e metodologias utilizadas para a elaboração do projeto, as quais são: Plataforma *Android*, requisições utilizando o protocolo *HTTP* (*HyperText Transfer Protocol*), Servidor Ubuntu alocado como serviço em *Amazon Elastic Compute Cloud* (EC2), SGDB *MySQL*, e respostas baseadas em *JSON* (*JavaScript Object Notation*). A Figura 1 demonstra a relação entre as tecnologias utilizadas.

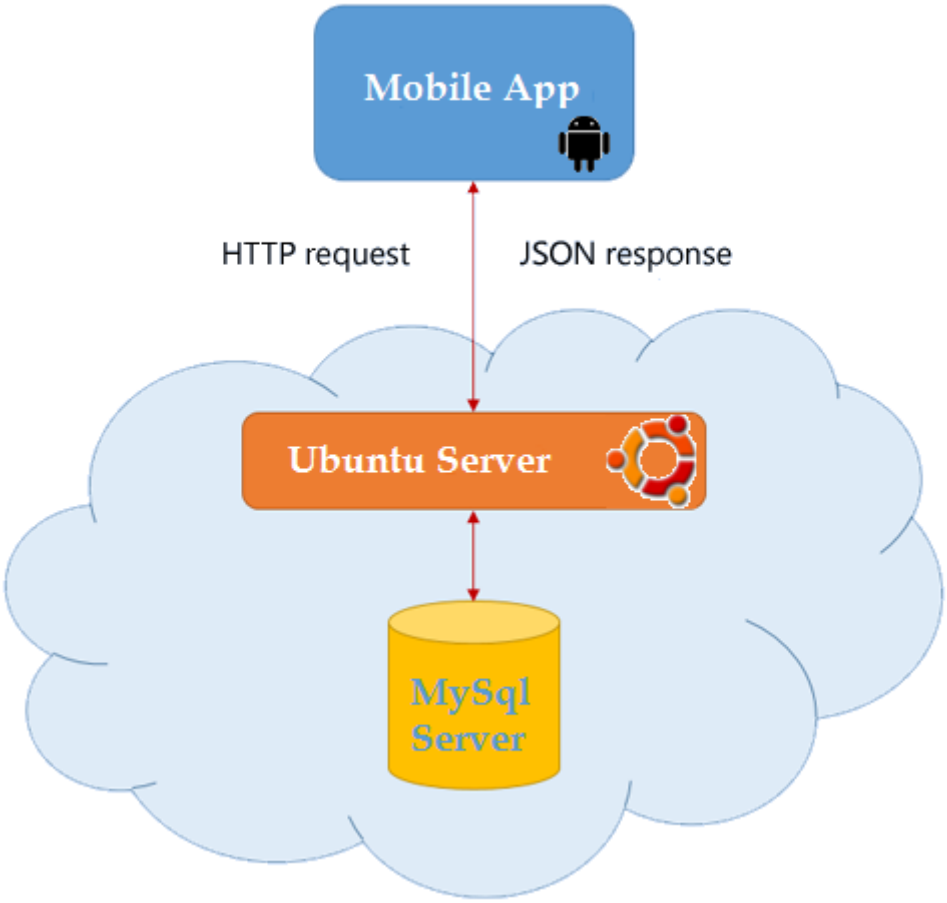


Figura 1: Representação das tecnologias abordadas
(Fonte: Autor, 2014)

A seguir, serão apresentadas algumas definições e exemplos sobre essas tecnologias, para proporcionar um maior entendimento da parte técnica descrita no capítulo 2 deste trabalho, que apresenta o desenvolvimento do aplicativo.

1.1. *Android*

Em junho de 2005, o Google comprou a *Android* Inc., uma empresa *startup* que desenvolvia um sistema para celulares. Em 5 de novembro de 2007, o Google anunciou o nascimento da plataforma *Android*, e assim foi estabelecida uma junção entre o Google e mais 33 empresas parceiras da época, que foi intitulada OHA (*Open Handset Alliance*), cuja função é manter o *Android* alinhado com todas as expectativas demandadas do mercado. Dentre as empresas nesta junção estão Google, Samsung, Intel, HTC, Motorola. Em 12 de novembro de 2007, o *Android* SDK (*Software Development Kit*) foi mostrado a vários desenvolvedores, porém ainda não era *Open Source* (código aberto). Finalmente, em 21 de outubro de 2008, o *Android* tornou-se *Open Source*, e um dia depois, foi lançado o primeiro smartphone com a plataforma *Android*, o HTC G1 (LUCIEL, 2010).

O *Android* é um sistema operacional baseado em Linux, voltado para dispositivos móveis como *Smartphones* e *Tablets*. Atualmente é, segundo pesquisas IDC (*International Data Corporation*), a plataforma móvel mais utilizada no mundo, conforme análise por trimestre do gráfico a seguir, cujos valores demonstrados estão entre o segundo trimestre de 2011 (2011Q2) e o segundo trimestre de 2014 (2014Q2). Este levantamento é ilustrado na Figura 2.

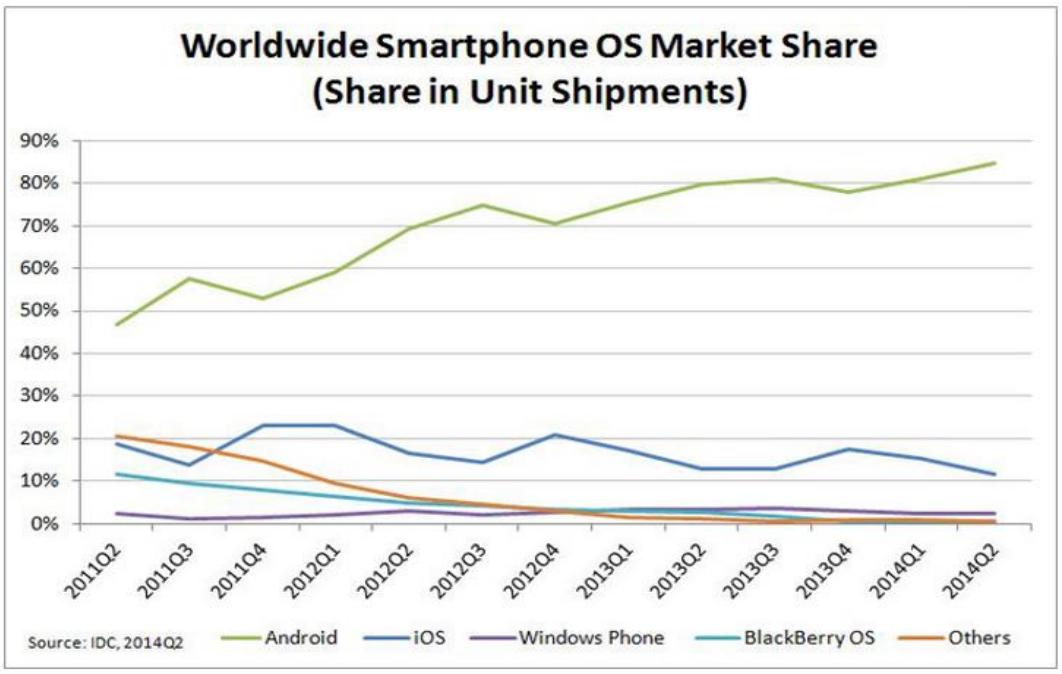


Figura 2: Venda de mercado em todo mundo
(Fonte: IDC, 2014)

Como foi adotado oficialmente em 2008, o *Android* é considerado uma plataforma recente, porém, o uso do *Android* por variados fabricantes de smartphone cresceu muito, e o motivo deste sucesso é identificado, principalmente, a sua característica de ser *free* (para uso livre) e O*pen Source*, além de ser um sistema baseado em linguagem Java, o que facilita muito o aprendizado por parte de desenvolvedores, já que é uma linguagem tão difundida e familiarizada nos dias de hoje.

1.1.1. Arquitetura *Android*

A arquitetura do *Android* é composta por diversas camadas e componentes, conforme demostrado na Figura 3, as quais serão detalhadas a seguir.

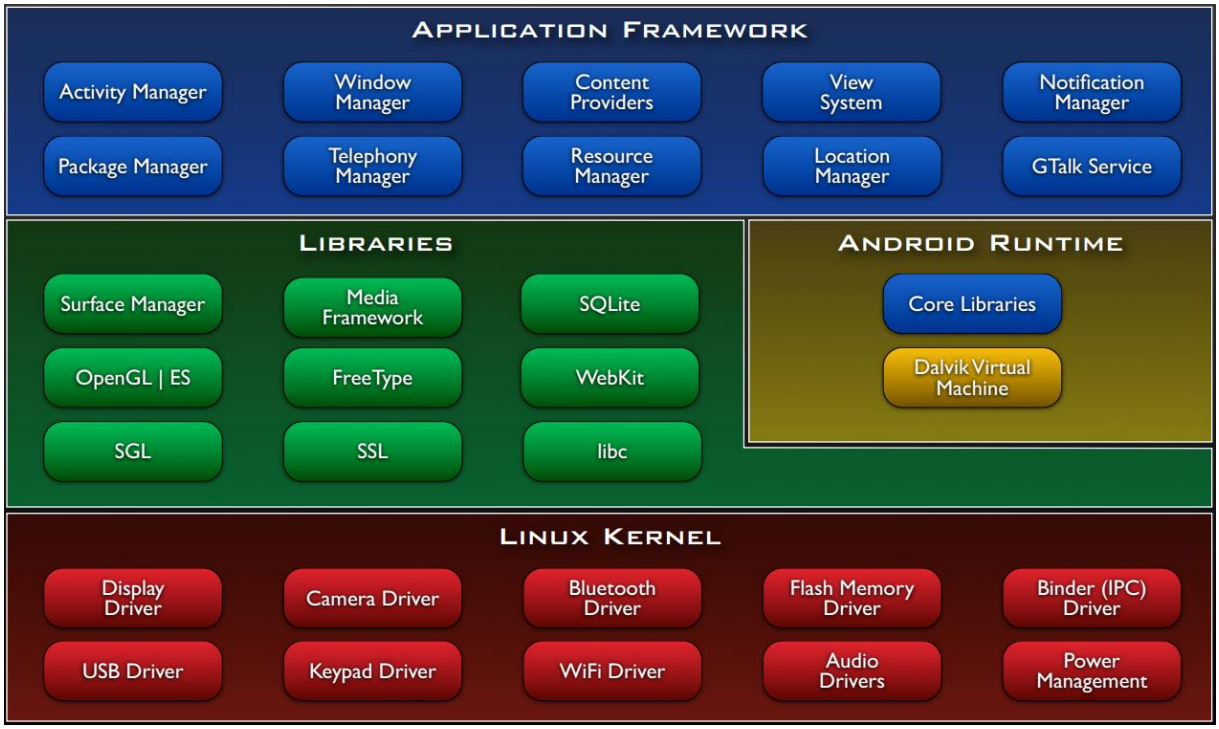


Figura 3: Diagrama da arquitetura do *Android*
(Fonte: Google. *Android: Low-Level System Architecture*, 2014)

Na base das camadas está **Linux Kernel**. O Google utilizou a versão 2.6 do Linux para implementação do Kernel do *Android*, o que inclui o gerenciamento de serviços de baixo nível, como memória, energia, segurança e *drivers* de *hardware*. O

gerenciamento de memória e processos possibilita executar diversas aplicações ao mesmo tempo, deixando algumas em segundo plano, sem que o usuário perceba, o que faz com que nenhuma aplicação dependa da outra, ou seja, caso uma aplicação pare de funcionar, isto não irá afetar quaisquer outras em funcionamento no dispositivo.

Na camada ***Libraries*** estão as bibliotecas nativas escritas em C/C++, que fazem parte da plataforma. Um conjunto de instruções que diz como o dispositivo lidará com diferentes tipos de dados ou requisições estão nesta camada APIs (*Application Programming Interface*), como por exemplo, OpenGL ES (para renderização 3D), SQLite (gerenciador de bancos de dados) e também suporte a diversos formatos de áudio e vídeo.

A camada ***Android Runtime*** se instancia o componente chamado de *Dalvik Virtual Machine* que dará suporte a execução de aplicações para a plataforma e o componente *core libraries*, que contém bibliotecas importantes para que métodos e funcionalidades contidas na API Java possam ser acessadas.

Por fim, a camada ***Application Framework*** representa as aplicações que executam na plataforma *Android*. Elas podem ser tanto aplicações nativas, como por exemplo gerenciador de telefonia, localização, contatos, navegador, calendário, etc., quanto aplicações criadas por terceiros. Assim se dá a característica de flexibilidade da plataforma *Android*, pois ela não diferencia uma aplicação nativa de uma aplicação de terceiro (GOOGLE. *Android*: *Low-Level System Architecture*, 2014).

1.1.2. Estrutura das Aplicações *Android*

Nas aplicações desenvolvidas para a plataforma *Android* há uma estrutura baseada em componentes, conforme demostrado na Figura 4. Geralmente utiliza-se uma combinação destes componentes.

Figura 4: Componentes
(Fonte: TOSIN, 2011)

Activities são as representantes das telas da aplicação, ou seja, se for necessário criar uma nova tela, precisa-se de uma nova *Activity*, a qual irá apontar qual *view* (exibição visual) irá definir como será feita a exibição visual desta tela para o usuário. As activities são responsáveis por tratar os eventos gerados na tela e coordenar o fluxo da aplicação.

Os **Services** são códigos que executam em segundo plano sem interface. Geralmente são utilizados para tarefas que levam muito tempo de execução, como uma consulta ao banco de dados, por exemplo. Um serviço tem um ciclo de vida próprio delimitado por começo, meio e fim.

Os ***Content Providers*** (provedores de conteúdo) são a maneira utilizada pela plataforma para compartilhar dados entre aplicações. Um exemplo é uma aplicação desenvolvida por terceiro, que pode utilizar um *Content Provider* para acessar a lista de contatos, que é uma aplicação nativa.

Os ***Broadcast Receivers*** tratam reações de eventos externos, pois são mecanismos de alerta. Um exemplo é utilizar um *broadcast receiver* para realizar algum tipo de tarefa quando o dispositivo estiver recebendo uma ligação.

O arquivo **AndroidManifest.xml** é obrigatório e único para cada aplicação. Nele são configuradas todas as permissões de acessos a *hardwares* do dispositivo,

como acesso à Internet, câmera, contatos, etc., no qual também são declaradas todas as *activities* do projeto.

Juntando tudo isto, existe a figura do *Android Core*, que é a plataforma *Android* propriamente dita, que proporciona a interação entre os componentes e as aplicações, assim tornando possível a execução do código (TOSIN, 2011).

1.1.3. Ciclo de Vida da *Activity*

O ciclo de vida de uma *Activity* se define por que tipo de estado que ela está em determinado momento, que podem ser: executando, temporariamente interrompida, em segundo plano ou completamente destruída (LECHETA, 2014 p. 115).

Entende-se como ciclo de vida, algo que tenha começo, meio e fim, também chamados de *Entier Lifetime*, *Visible Lifetime*, *Foreground Lifetime*.

Entier Lifetime é o ciclo de vida completo entre o início e destruição de uma *Activity*, que ocorre apenas uma vez entre as chamadas dos métodos *onCreate* e *onDestroy*, geralmente utilizados para criação de algum recurso em memória e sua respectiva liberação.

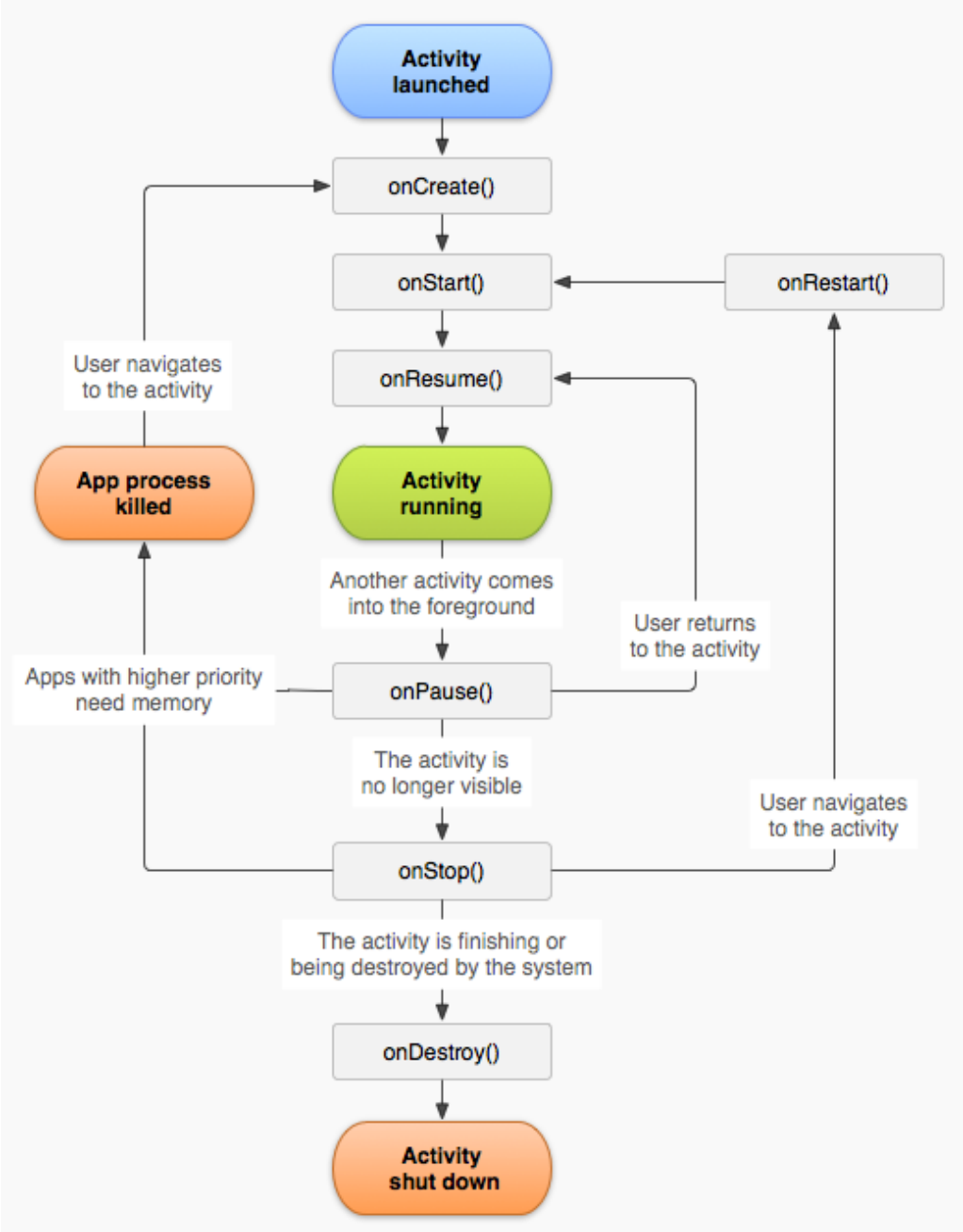


Figura 5: Ciclo de vida da *Activity*
(Fonte: Google. *Android*: *Activity Lifecycle*, 2014)

Visible Lifetime corresponde a uma situação em que a *Activity* está visível para o usuário ou parada em segundo plano. Este ciclo ocorre entre os métodos *onStart* e *onStop*. Ao ocorrer a chamada do método *onStart* define-se uma ordem de execução que se repete entre as chamadas dos métodos *onStart, onResume, onPause, onStop, onRestart, onStart.* É importante lembrar que, neste caso, a Activity só fica visível para o usuário após a execução do método *onResume.*

Foreground lifetime é quando a *Activity* está visível para o usuário. Este ciclo ocorre entre os métodos *onPause* e *onResume*. Pode ser chamado quando o celular está bloqueado em modo de espera e quando o usuário o ativa retornando à aplicação respectivamente. Estes métodos são usados muitas vezes. Desta forma é importante que não se usem códigos que demandem muitas tarefas ou alocação de memória.

O método *onCreate* é obrigatório e chamado apenas uma vez, no qual deve-se criar uma *view* chamando o método *setContentView*. Assim que o método é finalizado, o método *onStart* é chamado para exibição da *view*.

O método *onStart* é chamado para iniciar a exibição da *Activity* que já contém uma *view* criada. Este método pode ser chamado depois do método *onCreate* ou *onRestart*, dependendo do estado da *Activity* e sempre precede o método *onResume*.

O método *onRestart* é chamado quando a *Activity* volta a ser iniciada.

O método *onResume* é chamado quando a *Activity* está pronta para a interação com o usuário. *Activity Running* é o termo que se refere à execução propriamente dita da atividade é precedida pelo método *onStart*.

O método *onPause* é chamado quando o dispositivo entra em modo de espera para economizar energia ou quando outra *Activity* é iniciada.

O método *onStop* é chamado quando a *Activity* está sendo encerrada ou não está mais visível ao usuário, e pode ocorrer quando outra *Activity* é iniciada.

O método onDestroy é chamado para encerrar a execução de uma *Activity*.

Vale ressaltar que o sistema operacional pode destruir o processo da *Activity* caso a memória do dispositivo esteja saturada, o que pode ocorrer na chamada dos métodos *onPause*, *onStop* e *onDestroy* (GOOGLE. *Android*: *Activity Lifecycle*, 2014).

1.1.4. Interface

Todo componente de tela do *Android* é uma *View*, ou seja, *View* é a classe base para os componentes de tela, e *ViewGroup* é a classe base para os *layouts*.

Uma *View* é representada pela classe *android.view.View*, na qual podem-se obter informações de medidas da tela, mudanças de foco, barra de rolagem, captura de eventos, e também para estender outros *widgets* como *Text*, *EditText*, *InputMethod*, *MovementMethod*, *Button*, *RadioButton*, *Checkbox*, and *ScrollView*.

O *ViewGroup* é representado pela classe *android.view.Viewgroup* e é considerado um *layout*, ou seja, um conjunto de *Views* ou de outros *ViewGroup*, que pode ser utilizado para criar estruturas de dados mais complexas, ricas e robustas, usada como base para outros *layouts* como *LinearLayout*, *RelativeLayout*, *AbsoluteLayout* (RABELLO, 2007).

1.1.5. Gerenciamento de *Layout*

Um *layout* define a estrutura visual para a interface do usuário. O gerenciador *AbsoluteLayout* permite especificar a localização exata (coordenadas X / Y) de seus filhos. *AbsoluteLayout* é menos flexível e mais difícil de manter do que outros tipos de *layouts*, pois há diferenças no tamanho de telas dos dispositivos, o que proporciona dificuldade de gerenciamento e posicionamento dos componentes de tela. A Figura 6 ilustra um exemplo.

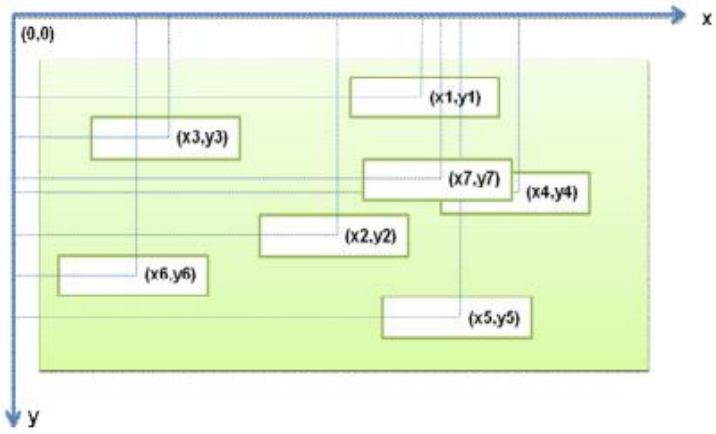


Figura 6: *AbsoluteLayout*
(Fonte: RABELLO. Construindo *layouts* complexos, 2007)

FrameLayout é o tipo mais comum e simples de *layout*, utilizado por um componente que precisa preencher a tela inteira, é utilizado para componentes que precisam sobrepor ao outro como uma pilha, apresentação de slides por exemplo. A Figura 7 ilustra um exemplo.

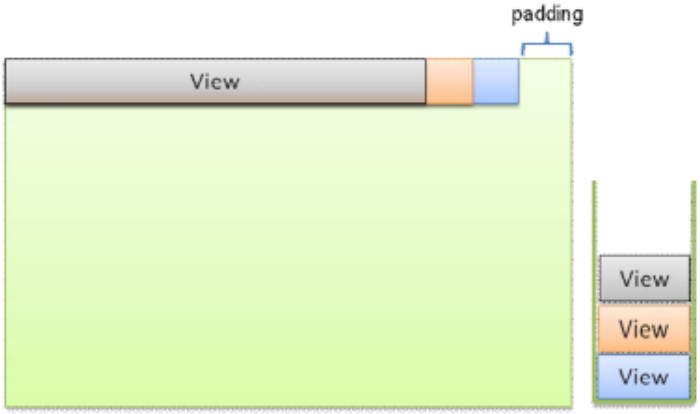


Figura 7: *FrameLayout*
(Fonte: RABELLO. Construindo *layouts* complexos, 2007)

LinearLayout é utilizado para organizar os componentes na vertical ou horizontal por meio de um atributo de orientação que pode ter dois estados, vertical ou horizontal. Este tipo de *layout* é o mais utilizado e mais recomendado para ter maior organização dos componentes dentro da tela. A Figura 8 ilustra um exemplo.

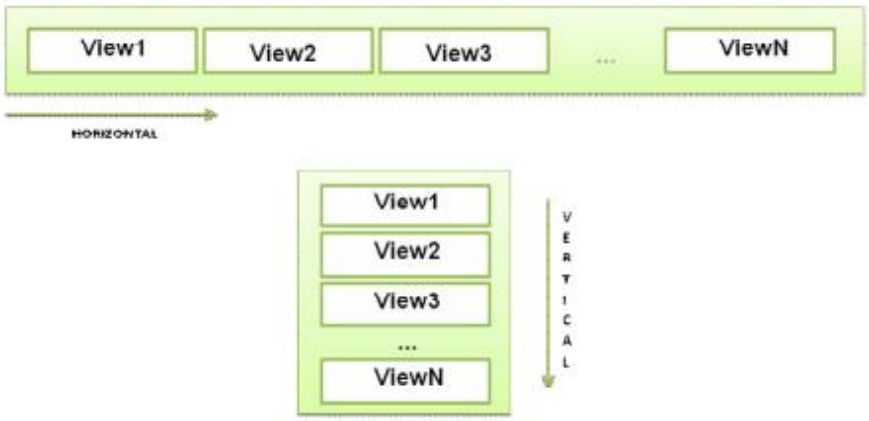


Figura 8: *LinearLayout*
(Fonte: RABELLO. Construindo *layouts* complexos, 2007)

O gerenciador *TableLayout* é uma herança de *LinearLayout* e pode ser utilizado para organizar os componentes em uma tabela, com linhas e colunas, e é muito utilizado para construção de formulários, nos quais cada componente está presente em uma célula. A Figura 9 ilustra um exemplo.

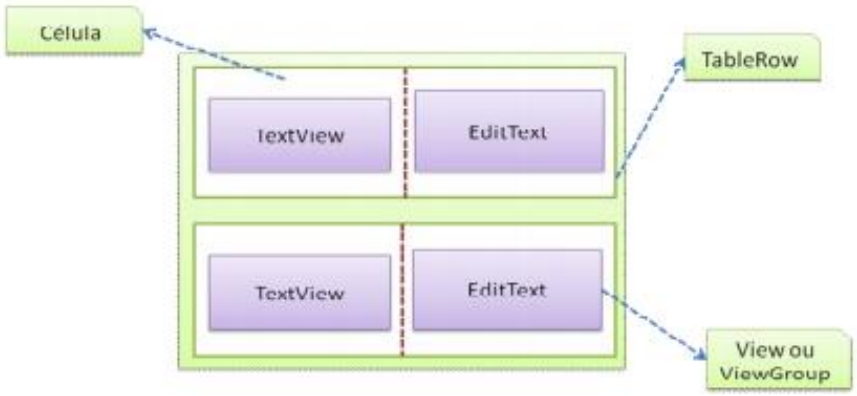


Figura 9: *TableLayout*
(Fonte: RABELLO. Construindo *layouts* complexos, 2007)

RelativeLayout posiciona um componente relativo a outro já existente, para isso é necessário definir um atributo de identificação para todos os componentes. A Figura 10 ilustra um exemplo.

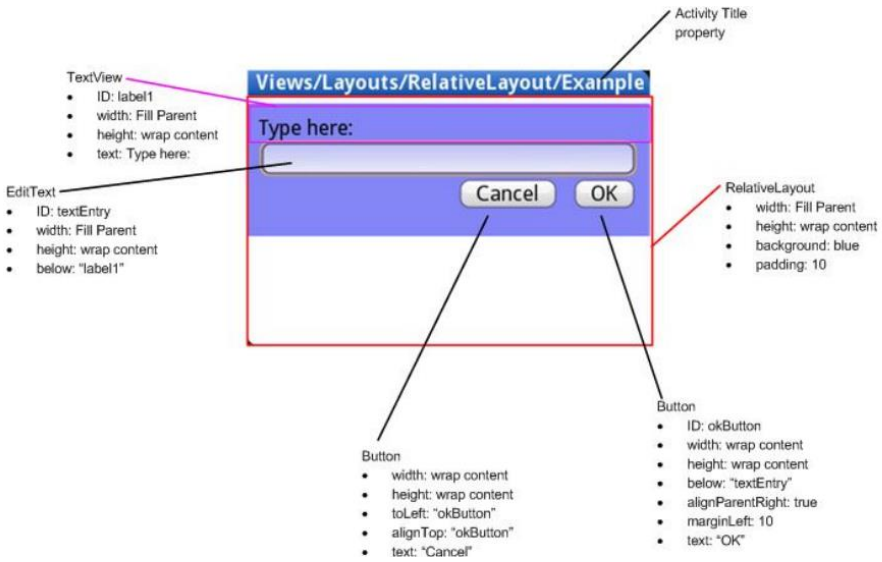


Figura 10: *RelativeLayout*
(Fonte: RABELLO. Construindo *layouts* complexos, 2007)

Todos os gerenciadores de *layout* são baseados na classe *android.view.ViewGroup*. Acima foram descritos os principais e mais utilizados *layouts*, porém há outras subclasses de *ViewGroup,* por exemplo *ScrollView*, *GridView*, *Gallery*, *WebView* e *TabHost*. Apesar de não serem gerenciadores de *layouts*, são subclasses de *ViewGroup* e são capazes de conter outras *Views* e organizá-las na tela do dispositivo (RABELLO. Construindo *layouts* complexos, 2007).

1.1.6. Componentes de Interface

Para que o *layout* do aplicativo *Android* componha uma interface iterativa com o usuário, é necessário adicionar componentes na tela, os quais denominam-se *widgets*, subclasses de uma *View*.

TextView é a primeira e mais simples das subclasses e representa um texto na tela. A Figura 11 ilustra um exemplo de um *TextView* dentro de um *layout*.

Figura 11: *TextView*
(Fonte: Google. *Android*: *Input Controls*, 2014)

EditText é uma subclasse de *TextView* permite que o usuário digite e edite informações em um campo de texto. Pode ser especificada uma configuração de tipo dado que o componente irá receber, que podem ser número, e-mail, senha ou simplesmente um texto. Ao configurar o tipo de dado de entrada, o teclado do *Android* se ajusta ao tipo de dado, o que é feito pelo atributo *android:inputType.* A Figura 12 ilustra um teclado de e-mail, número e texto.

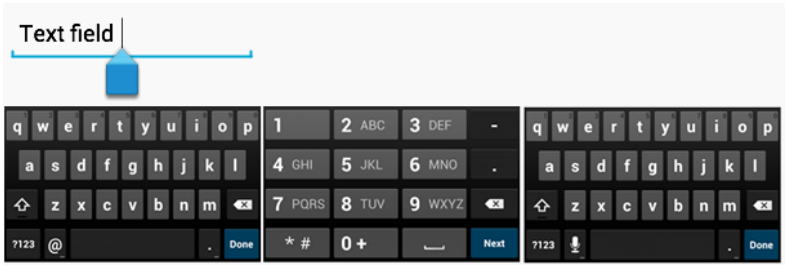


Figura 12: *EditText* e Teclado
(Fonte: Google. *Android*: *Input Controls*, 2014)

O *ImageButton* é um botão na tela que possui uma imagem. O *Button* também refere-se a um botão e pode possuir, tanto um texto, quanto uma combinação de texto e imagem. A Figura 13 ilustra o *Button*, *ImageButton* e *Button* respectivamente.

Figura 13: *Button* e *ImageButton*
(Fonte: Google. *Android*: *Input Controls*, 2014)

RadioButton permite que usuário selecione uma única opção de uma determinada lista, utilizado quando há a necessidade do usuário enxergar todas as opções. A Figura 14 ilustra um exemplo.

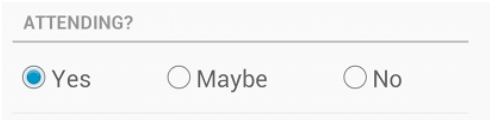


Figura 14: *RadioButton*
(Fonte: Google. *Android*: *Input Controls*, 2014)

Tougle Button ou *Switch* é um botão que pode assumir dois estados, ligado e desligado, por exemplo. A Figura 15 ilustra esta situação.

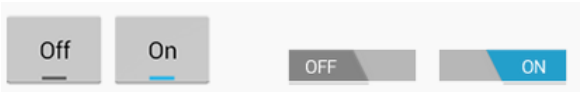


Figura 15: *Tougle Button* e *Switch*
(Fonte: Google *Android*: *Input Controls*, 2014.)

Checkbox permite que usuário selecione uma ou mais opções de uma determinada lista. A Figura 16 ilustra um exemplo.

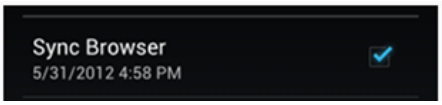


Figura 16: *Checkbox*
(Fonte: Google. *Android*: *Input Controls*, 2014.)

Spinners fornece uma maneira rápida para selecionar um valor a partir de um conjunto. Ao tocar no *Spinner*, exibe-se um menu suspenso com todas opções disponíveis (Google, *Android*: *Input Controls*, 2014). A Figura 17 ilustra um exemplo.

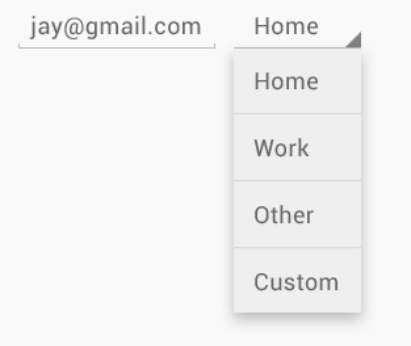


Figura 17: *Spinner*
(Fonte: Google, *Android*: *Input Controls*, 2014)

1.1.7. Persistência de Dados

Android fornece várias opções para se gravar os dados de uma aplicação. É preciso levar em conta os seguintes fatores: se os dados são privados ou públicos, o tamanho do dado em disco, a qualidade do dado. Neste projeto, será utilizada a opção *Network Connection* (Google, 2014. *Data Options*). A Tabela 1 ilustra as opções de persistência de dados possíveis.

Tipo de armazenamento	Definição
Shared Preferences	Armazenar dados primitivos privadas em pares chave-valor.
Internal Storage	Armazenar dados privados na memória do dispositivo.
External Storage	Armazenar dados públicos sobre o armazenamento externo compartilhado (SD Card)
SQLite Databases	Armazenar dados estruturados em um banco de dados privado.
Network Connection	Armazenar dados na web com seu próprio servidor de rede.

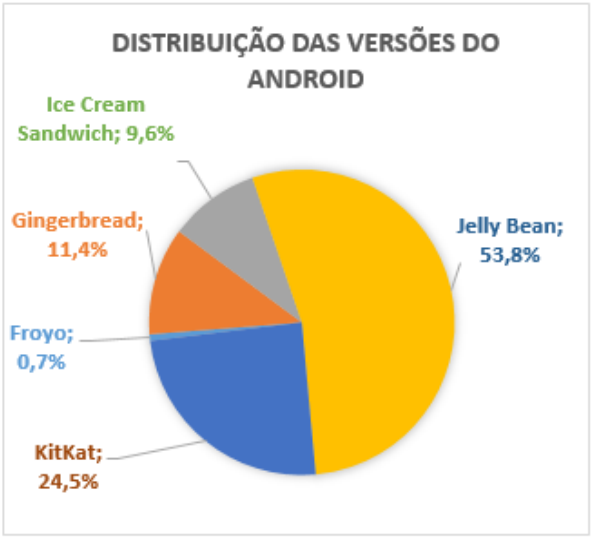
Tabela 1: Tipos de persistência de dados
(Fonte: Google. *Data Options*, 2014.)

1.1.8. Versões da Plataforma *Android*

As versões da plataforma *Android* são definidas por codinomes de doces, padrão adotado pela Google a partir da versão 1.5, os quais seguem uma sequência em ordem alfabética na primeira letra do codinome. Pode-se notar, conforme análise dos dados do site oficial do Google extraídos com base na Play Store (Loja on-line)

em setembro de 2014, que 87% das versões do *Android* em dispositivos móveis estão concentrados na versão 4.0 API nível 15 ou superior. Observa-se também que recentemente foi lançado o *Android* 5.0 de codinome *Lollipop*, versão com uma API poderosa e que será destinada a vários dispositivos, como *Smartphones*, *Wearables*, *Tablets*, *SmartTVs* e Carros. A Figura 18 mostra a evolução das versões, codinomes, datas de lançamento, APIs e distribuição, e também um gráfico comparativo, que leva em conta as versões com utilização acima de 0,1% (Google Dashboard, 2014).

Versão	Codinome	Lançamento	API	Distribuição
1.0	Alpha	26 Set. 2008	1	0,0%
1.1	Beta	9 Fev. 2009	2	0,0%
1.5	Cupcake	30 Abr. 2009	3	0,0%
1.6	Donut	15 Set. 2009	4	0,0%
2.0	Eclair	26 Out. 2009	5	0,0%
2.0.1		03 Dez. 2009		
2.1		11 Jan. 2010		
2.2	Froyo	20 Mai. 2010	8	0,7%
2.2.1		18 Jun. 2010		
2.2.2		22 Jun. 2010		
2.2.3		21 Nov. 2010		



Versão	Codinome	Lançamento	API	Distribuição
2.3	Gingerbread	6 Dez. 2010	9	11,4%
2.3.3		9 Fev. 2011	10	
2.3.4		28 Abr. 2011		
2.3.5		25 Jun. 2011		
2.3.6		2 Set. 2011		
2.3.7		21 Set. 2011		
3.0	Honeycomb	22 Fev. 2011	11	0,0%
3.1		10 Mai. 2011	12	
3.2		15 Jul. 2011	13	
3.2.1		30 Ago. 2011		
3.2.2		20 Out. 2011		
		Dez. 2011		
3.2.6		Fev. 2012		
4.0.1	Ice Cream Sandwich	19 Out. 2011	14	9,6%
4.0.2		28 Nov. 2011		
4.0.3		16 Dez. 2011	15	
4.0.4		29 Mar. 2012		
4.1.1	Jelly Bean	9 Jul. 2012	14	0,0%
4.1.2		9 Out. 2012	16	25,1%
4.2		26 Out. 2012	17	20,7%
4.2.1		27 Nov. 2012		
4.2.2		30 Abr. 2013		
4.3		24 Jul. 2013	18	8,0%
4.4	KitKat	31 Out. 2013	19	24,5%
4.4.4		21 Jul. 2014	20	
5.0	Lollipop	15 Out. 2014	21	Lançamento

Figura 18: Distribuição por versão do *Android*
(Fonte: Google. *Dashboard*, 2014)

O conhecimento e interesse sobre a plataforma *Android* foi adquirido através do guia disponível no site oficial do *Android*, consulta a livros Google *Android* e *Android* para Tablets (LECHETA, 2014), fundamentais para o desenvolvimento da aplicação e vídeos do Google I/O, maio de 2013 e junho de 2014 nos Estados Unidos.

1.2. Serviços de Banco de Dados

Para o desenvolvimento deste projeto, utilizou-se um serviço de banco de dados do tipo relacional, pois é necessário guardar informações para que o usuário tenha um controle e histórico de informações veiculares. Conforme Date (2003, p.6), banco de dados é definido por:

> “[...] um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando as solicitar."

Para o autor, informação é definida como algo que tenha significado ao indivíduo de forma que possa utilizar para uma finalidade específica (Date 2003, p. 6).

1.2.1. Modelo Relacional

Muitas maneiras de persistência de dados surgiram e desapareceram ao longo dos anos, e nenhum conceito teve mais poder de permanência do que o banco de dados relacional. Mesmo na era da nuvem, quando modelos não relacionais como *Big Data* e *NoSQL* roubam regularmente as manchetes, os serviços relacionais de banco de dados estão em demanda consistente para permitir que os aplicativos empresariais de hoje fiquem em execução na nuvem. É aí que a grande maioria dos dados corporativos do mundo está armazenada e é o ponto de partida para cada aplicativo corporativo, que muitas vezes têm um tempo de vida maior, pois os dados continuam persistindo, mesmo após a aplicação desaparecer (KEITH, SCHINCARIOL, 2013 p. 1).

Neste modelo, a representação dos dados é a entidade, uma tabela com linhas não ordenadas e colunas e consiste de um esquema que especifica o nome da entidade e o nome e domínio de cada coluna, também denominada atributo ou campo da relação. O domínio do atributo é definido no esquema, com o objetivo de restringir o valor que o atributo pode assumir. O esquema de uma entidade não varia ao longo do tempo, só será modificado por comandos específicos. Abaixo tem-se um exemplo de entidade, atributos e domínio:

Carro (*car_id: integer*, *car_marca: string*, *car_modelo: string*, *car_placa: string*, *car_odomentro: integer*, *car_motor: string*, *car_tanque: float* e *car_flag: char*)

Neste caso, a entidade de nome *Carro* está definida com atributos *car_id*, *car_marca*, *car_modelo*, *car_placa*, *car_odomentro*, *car_motor*, *car_tanque* e *car_flag*, cujos domínios são *integer*, *string, string, string, integer, string, float,* e *char*, respectivamente.

A instância de uma entidade são linhas, também chamadas de tuplas ou registros, que são distintas entre si. Elas compõem a entidade em um determinado instante, o que pode variar com o tempo. Cada linha de uma entidade deve seguir sempre o seu respectivo esquema, respeitando o número de atributos definidos representados como colunas, bem como os seus domínios que são os tipos de dado que pode-se preencher em cada linha. A restrição de domínio é muito importante. O modelo relacional somente considera relações que satisfaçam esta restrição (MACÁRIO, BALDO, 2005).

CARRO							
car_id	car_marca	car_modelo	car_placa	car_odometro	car_motor	car_tanque	car_flag
4	Chevrolet	Celta	EVX6728	22346	1.0	50.00	1
37	Fiat	Uno	EDF2344	125	1.5	50.00	1
42	Fiat	Palio	ETX3455	123345	1.0	45.00	0

Tabela 2: Exemplo de instância da entidade Carro
(Fonte: Autor, 2014)

O número de tuplas que uma instância possui corresponde a sua cardinalidade da relação, e o número de atributos é o seu grau. A instância de relação da Tabela 2 tem cardinalidade 3 e grau 8. Pode-se notar que a cardinalidade é variável, mas o grau não.

Um banco de dados relacional é um conjunto de uma ou mais entidades com nomes distintos. A coleção dos esquemas de cada entidade do banco de dados relacional é o que compõe o banco de dados (MACÁRIO, BALDO, 2005).

Compreender dados relacionais é fundamental para o desenvolvimento de software de sucesso. Bases de dados é um obstáculo comumente conhecido no desenvolvimento de software (KEITH, SCHINCARIOL, 2013 p. 1).

1.2.2. Restrições de Dados

A restrição de **integridade** de entidade estabelece que o valor da chave primária não pode ser nulo, pois é usada para identificação das tuplas individuais relacionadas. Se uma ou mais estiverem com valor nulo, não será capaz de distingui-las e não será possível realizar referência a elas por intermédio de outras relações.

Na Figura 19, pode-se observar que cada conjunto da entidade Carro ocupa no mínimo e máximo um usuário. Porém, um usuário pode ocupar nenhum ou muitos Carros. Para a identificação das tuplas para a entidade Carro tem-se o atributo car_id, e para o usuário, tem-se o atributo user_name. O campo de relação entre as tabelas é definido pelos atributos *car_user* da entidade Carro e o *user_name* da entidade Usuário.

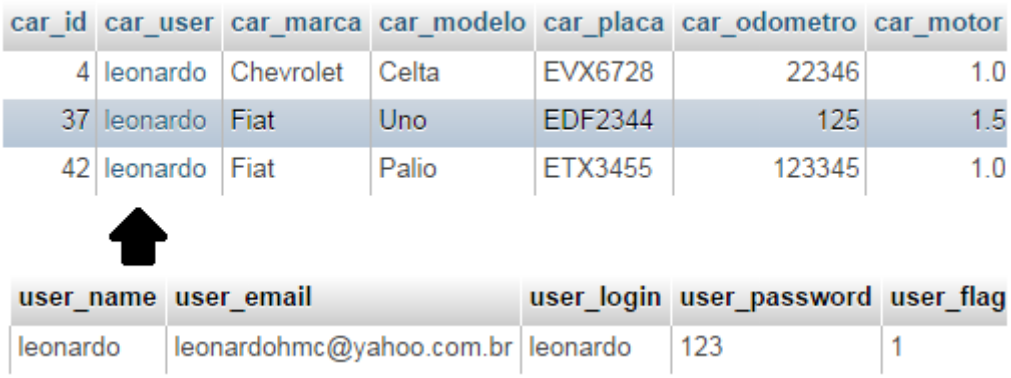


Figura 19: Relação entre entidades
(Fonte: Autor, 2014)

1.2.3. Sistema de Gerenciamento de Banco de Dados MySQL

O SGDB utilizado para desenvolvimento do projeto foi o *MySQL*, que é o Sistema Gerenciador de Bancos de Dados Relacional mais utilizado no mundo. Um dos motivos para este *status*, está na característica de ser *Open Source*, ou seja, possui código fonte aberto, o que faz com que qualquer um possa utilizar e modificar o programa, e também possui licença gratuita por meio da GNU (*General Public License* – Licença Pública Geral). Além disto, é considerado de nível corporativo, fácil de usar, multiplataforma, devido ao fato de suas fontes serem escritas em C/C++, que proporciona portabilidade e suporte total a servidores multiprocessados, privilegiando

o desempenho e a segurança dos dados, de acordo com a necessidade do usuário (RANGEL, 2004 p.5 e 32)

1.3. *Amazon Web Services* (AWS)

AWS é um conjunto de serviços (*web services*) e aplicações de computação que são acessados remotamente via Internet. O AWS proporciona uma infraestrutura de computação completa que funciona em diversos níveis de processamento, desde as mais simples até as mais eficazes, que inclui a gerência dos recursos disponibilizados. As características destes recursos são: escalabilidade, disponibilidade, elasticidade e desempenho (Amazon, 2014). Dentre os serviços utilizados neste projeto encontram-se o Amazon EC2 e o Amazon S3.

O Amazon EC2 fornece servidores privados escaláveis com capacidade computacional redimensionável. A Figura 20 mostra o método de cobrança dos serviços EC2 alocado.

	Uso otimizado do EBS
Instâncias padrão	
m1.large	$0.025 por hora
m1.xlarge	$0.05 por hora
Instâncias padrão de segunda geração	
m3.xlarge	$0.025 por hora
m3.2xlarge	$0.05 por hora
Instâncias de CPU de alto desempenho	
c3.xlarge	$0.02 por hora
c3.2xlarge	$0.05 por hora
c3.4xlarge	$0.10 por hora
c1.xlarge	$0.05 por hora
Instâncias de GPU	
g2.2xlarge	$0.05 por hora

	Uso otimizado do EBS
Otimizadas para memória – Geração atual	
r3.xlarge	$0.02 por hora
r3.2xlarge	$0.05 por hora
r3.4xlarge	$0.10 por hora
Memory Optimized - Previous Generation	
m2.2xlarge	$0.025 por hora
m2.4xlarge	$0.05 por hora
Otimizadas para armazenamento	
i2.xlarge	$0.02 por hora
i2.2xlarge	$0.05 por hora
i2.4xlarge	$0.10 por hora

Figura 20: Cobrança pelo serviço Amazon EC2
(Fonte: Amazon, 2014)

O Amazon S3 (*Simple Storage Service* – Serviço de Armazenamento Simples) fornece infraestrutura de armazenamento através de interfaces de serviços web (*HTTP*, REST, SOAP e BitTorrent), e é totalmente redundante para armazenar e recuperar dados. A Figura 21 mostra o método de cobrança pelos serviços S3.

	Definição de preço
Solicitações PUT, COPY, POST ou LIST	$0.005 por 1.000 solicitações
Solicitações de arquivamento e restauração do Glacier	$0.05 por 1.000 solicitações
Excluir solicitações	Free †
Solicitações GET e todas as outras	$0.004 por 10.000 solicitações
Restaurações de dados do Glacier	Free ‡

Figura 21: Cobrança pelo serviço Amazon S3
(Fonte: Amazon, 2014)

1.3.1. Infraestrutura Como Um Serviço (IaaS)

Infraestrutura como Serviço (IaaS) é um tipo de plataforma de *cloud computing* em que a organização do cliente terceiriza sua infraestrutura de TI, incluindo armazenamento, processamento, memória, rede e outros recursos. Clientes podem acessar esses recursos através da internet com uma configuração de hardware pré determinada que independe da utilização efetiva e a configuração pode ser redimensionada dinamicamente para atender às suas necessidades em constante mudança, o que pode ser feito por períodos específicos de tempo, além de ser um modelo de pagamento somente pelo o que usar (*on demand*).

PMEs (Pequenas e Medias Empresas) podem reduzir seus investimentos e satisfazer as necessidades contingentes que anteriormente era disponível apenas para governos e grandes corporações.

IaaS é oferecido em três modelos: privado, público e de nuvem híbrida:

- Privado: implica que a infraestrutura reside na premissa do cliente.

- Público, ele está localizado no data center do fornecedor
- Híbrido é uma combinação dos dois, conforme necessidade e escolha do cliente, utilizando o “melhor dos dois mundos”.

Os prós da implantação de IaaS como plataforma de computação em nuvem:

- Escolher dinamicamente uma CPU (*Central Processing Unit*), memória e configuração de armazenamento para atender às necessidades.
- O acesso ao poder computacional disponível na plataforma.
- Elimina o investimento em hardware de TI raramente usados.
- Despesas gerais de TI tratado pelo fornecedor de plataforma.
- A infraestrutura de TI local pode ser dedicada a atividades centrais para a organização.

Os contras da implantação de IaaS plataforma de computação em nuvem:

- Há um risco de o fornecedor da plataforma ter acesso aos dados da organização. Pode ser evitado, optando por nuvem privada.
- Depende da disponibilidade da internet.
- Depende da disponibilidade de serviços de virtualização.
- Pode limitar a privacidade do usuário e opções de personalização.

IaaS plataforma de computação em nuvem não pode eliminar a necessidade de um departamento de TI local, pois será necessário monitorar e configurar os serviços adquiridos.

Avarias do fornecedor da plataforma pode ser um problema para o negócio, e é importante avaliar as finanças e estabilidade para certificar-se de que os SLAs (*Service Level Agreement* – Acordo de Nível de Serviço) forneçam backups, caso haja falhas de hardware, rede, dados e aplicativos.

Amazon Web Services (AWS) é o primeiro e mais popular em serviços IaaS. Outras empresas que fornecem os serviços são *Rackspace*, *Google*, *GoGrid*, e *Joyent* (SERCHCIO.IN, 2011).

1.4. Protocolo de Transferência de Hipertexto (*HTTP*)

HTTP é, talvez, o protocolo de aplicação mais popular usado na Internet, considerado cliente-servidor de solicitação-resposta assimétrica. Um cliente *HTTP* envia uma mensagem de solicitação para um servidor *HTTP*. O servidor, por sua vez, envia uma mensagem de resposta. Em outras palavras, o *HTTP* é um protocolo para requisitar dados. O cliente recebe a informação a partir do servidor. *HTTP* é um protocolo sem estado. Em outras palavras, a solicitação atual não sabe o que foi feito nos pedidos anteriores. *HTTP* permite negociação de tipo de dados e de representação, de forma a permitir que os sistemas sejam construídos independentemente dos dados a serem transferidos. É um protocolo de nível de aplicativo para distribuídos e corporativos, genérico e sem estado, que pode ser usado para muitas tarefas. Além de hipertexto, com orientação a objetos pode ser usado como nome de servidor para gerenciamento de sistemas distribuídos.

Sempre que alguém emitir uma URL (*Uniform Resource Locator*) do seu navegador para obter um recurso da Web usando *HTTP*, o navegador transforma a URL em uma mensagem de solicitação e envia para o servidor *HTTP*. O servidor *HTTP*, por sua vez, interpreta a mensagem de pedido e retorna uma mensagem de resposta adequada, que pode ser o recurso solicitado ou uma mensagem de erro (HOCK-CHUAN, 2009). Este processo é ilustrado na Figura 22.

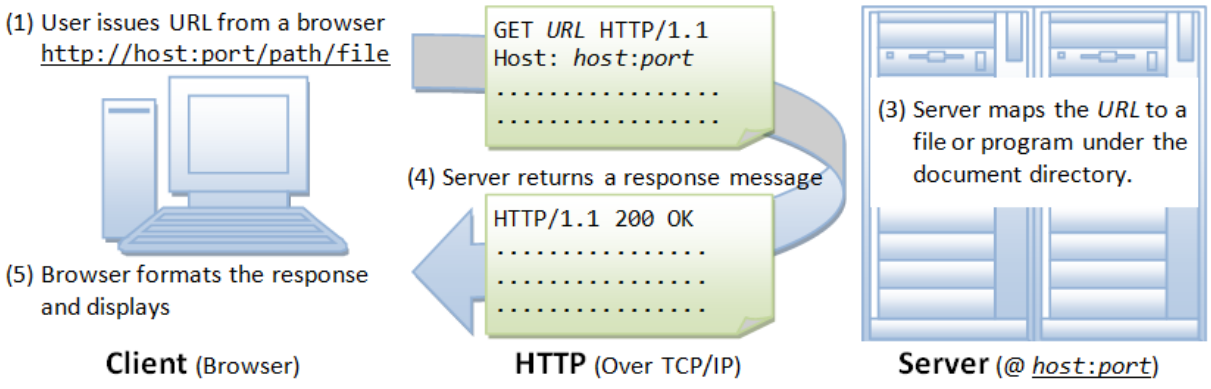


Figura 22: Requisição *HTTP* Cliente-Servidor
(Fonte: HOCK-CHUAN, 2009)

1.4.1. Localizador Uniforme de Recursos (URL)

URL é usado para identificar um recurso na web, e tem a seguinte sintaxe: *protocolo:///hostname: porto/diretório-arquivo* (HOCK-CHUAN, 2009).

- Protocolo: O protocolo de nível de aplicação utilizado pelo cliente e servidor, por exemplo, *HTTP*, FTP e telnet.
- Hostname: O nome de domínio DNS (por exemplo, www.test.com) ou endereço IP (por exemplo, 192.128.1.2) do servidor.
- Porto: O número da porta TCP que o servidor está atendendo às solicitações de entrada dos clientes; quando não é especificado o número do porto, obrigatoriamente toma sua forma padrão que é a porta TCP 80.
- Caminho-arquivo: O nome é o local do recurso solicitado, sob o diretório base de documentos do servidor.

1.4.2. Métodos de Requisições *HTTP*

Protocolo *HTTP* define um conjunto de métodos de solicitação que o cliente pode enviar para um servidor *HTTP* (HOCK-CHUAN, 2009).

- *GET*: Um cliente pode usar a requisição *GET* para obter dados.
- HEAD: Um cliente pode usar a solicitação HEAD para obter o cabeçalho que uma requisição *GET* teria obtido. Uma vez que o cabeçalho contém a data da última modificação de dados, isto pode ser usado para verificar contra a cópia cache local.
- *POST*: Usado para enviar dados para o servidor web.
- PUT: Pergunta ao servidor para armazenar os dados.
- DELETE: Pergunta ao servidor para apagar os dados.
- OPTIONS: Pergunta ao servidor para retornar a lista de métodos de solicitação que ele suporta.

1.4.3. Código de *Status* de Resposta

A primeira linha da mensagem de resposta contém o código de estado da resposta, o qual é gerado pelo servidor para indicar o resultado do pedido (HOCK-

CHUAN, 2009). O código de *status* é um número de 3 dígitos e é classificado da seguinte forma:

- 1xx (Informativo): Pedido recebido, servidor é a continuação do processo.
- 2xx (sucesso): O pedido foi recebido com sucesso, compreendidas, aceitas e atendidas.
- 3xx (redirecionamento): Outras medidas devem ser tomadas a fim de completar o pedido.
- 4xx (Erro de cliente): O pedido contém sintaxe inválida ou não possa ser compreendido.
- 5xx (Erro de Servidor): O servidor não cumpriu uma solicitação aparentemente válida.

1.5. *JavaScript Object Notation* (*JSON*)

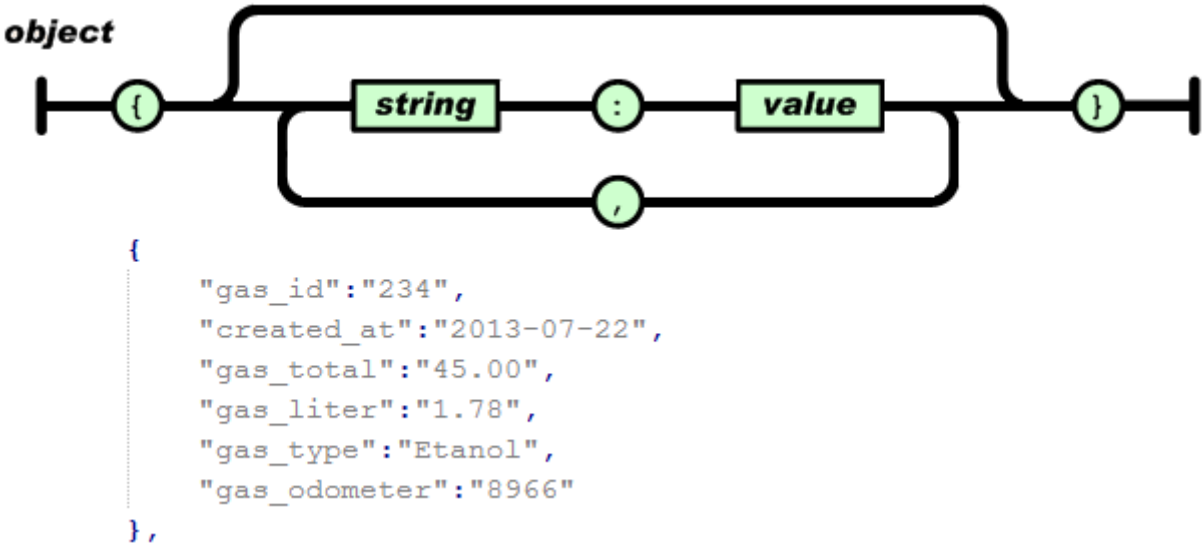
JSON é uma formatação leve de armazenamento e comunicação de conjunto de dados que é legível a olho humano e a máquinas, pois está baseado em um subconjunto da linguagem JavaScript. Por ser uma linguagem de texto, é considerada genérica, ou seja, é completamente independente de qualquer linguagem de programação. Utiliza convenções que são familiares para as linguagens C++, C#, *Java*, *JavaScript*, *Perl*, *Python* e outras que são derivadas da linguagem C, o que faz do *JSON* o mais indicado para a troca de dados definidos por um pequeno conjunto de regras estruturadas. Os valores que um *JSON* pode assumir são *object, array, number, string, true, false* ou *null (*ECMA INTERNATIONAL, 2013).

Para o desenvolvimento deste projeto será necessário entender o que é um *Object* e um *Array,* pois são os dois tipos mais utilizados, o que, porém, dependerá da necessidade de cada projeto para a implementação dos tipos de estruturas *JSON*.

1.5.1. Objeto

Um objeto nada mais é que uma estrutura de dados organizada. Na Figura 23, pode-se observar um exemplo de objeto *JSON*, que é delimitado por colchetes e dentro do colchete podem-se definir nenhum ou vários itens que são identificados conjuntos de chave/valor. Neste exemplo tem-se como chaves *gas_id*, *created_at*,

gas_total, *gas_liter*, *gas_type*, *gas_odometer*, e valores respectivamente 234, 2013-07-22, 45.00, 1.78, Etanol, 8966, como uma representação de um objeto. (ECMA INTERNATIONAL, 2013).

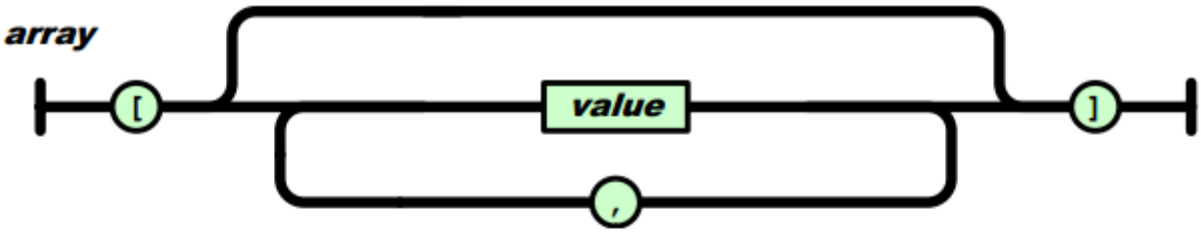



```
{
    "gas_id":"234",
    "created_at":"2013-07-22",
    "gas_total":"45.00",
    "gas_liter":"1.78",
    "gas_type":"Etanol",
    "gas_odometer":"8966"
},
```


Figura 23: Objeto *JSON*
(Fonte: ECMA INTERNATIONAL, 2013)

1.5.2. Array

Uma estrutura de *Array* é uma lista de objetos *JSON* que é delimitado por um par de colchetes. Os objetos *JSON* estão delimitados por chaves e separados por vírgulas conforme exemplo ilustrado na Figura 23. A ordem da chave/valor é significativa. A Figura 24 representa um exemplo de um objeto do tipo *Array* de nome objeto, que contem 2 objetos *JSON* na lista, o primeiro objeto tem-se como chaves *gas_id*, *created_at*, *gas_total*, *gas_liter*, *gas_type*, *gas_odometer*, e valores respectivamente 234, 2013-07-22, 45.00, 1.78, Etanol, 8966, e o segundo objeto tem-se como chaves *gas_id*, *created_at*, *gas_total*, *gas_liter*, *gas_type*, *gas_odometer*, e valores respectivamente 238, 2014-09-05, 120.00, 1.78, Gasolina, 145879.




```
"objeto":[
        {
            "gas_id":"234",
            "created_at":"2013-07-22",
            "gas_total":"45.00",
            "gas_liter":"1.78",
            "gas_type":"Etanol",
            "gas_odometer":"8966"
        },
        {
            "gas_id":"238",
            "created_at":"2014-09-05",
            "gas_total":"120.00",
            "gas_liter":"1.78",
            "gas_type":"Gasolina",
            "gas_odometer":"145879"
        }
    ],
```

Figura 24: Exemplo *Array JSON*
(Fonte: ECMA INTERNATIONAL, 2013)

2. DESENVOLVENDO O APLICATIVO

Neste capítulo serão apresentadas as ferramentas de desenvolvimento, o estudo de caso, requisitos, diagramas e trechos de códigos utilizados para o desenvolvimento do projeto.

2.1. Ambiente de Desenvolvimento

Para desenvolver aplicativos para a plataforma *Android*, é necessária instalação e configuração do *JDK* (*Java Development Kit*), *SDK* e da *IDE* de desenvolvimento *Eclipse*, configurar um ambiente *Java* com a última versão do *JDK*, que possui ferramentas necessárias para o desenvolvimento, e também o *Android SDK*, que disponibiliza biblioteca de *API*, e ferramentas de desenvolvimento necessários para construir, testar e depurar destinados às variadas versões do *Android*.

Hoje já existe uma IDE própria para desenvolvimento *Android* chamada *Android* Studio, porém ainda está em versão Beta, então para segurança no desenvolvimento foi utilizado a IDE Eclipse versão 4.4 (Codinome LUNA) versão disponibilizada em junho de 2014, devido ser a mais indicada para o desenvolvimento e também pela vantagem de ser código aberto. Neste IDE pode ser instalado vários plugins disponíveis para estender suas funcionalidades, dentre eles o ADT (*Android* Development Tools) que possibilita gerenciamento e controle com as ferramentas do *Android*. Este plugin irá fazer uma ponte entre o SDK e o Eclipse. As instruções para a instalação do ambiente de desenvolvimento *Android*, funcionamento de componentes, e exemplos de trechos de código poderão ser encontradas no site oficial (Google, *Android*: *Get the Android SDK*, 2014).

2.2. *Android Virtual Device* (AVD)

AVD é um emulador que permite modelar um dispositivo virtualmente para realizar testes do aplicativo. É muito importante ter uma configuração virtual, pois cria-se um ambiente de desenvolvimento muito próximo ao de um dispositivo real. É gerenciado pelo *ADT* (Dispositivo Virtual do Android) instalado no *Eclipse*, pois nele podem ser definidas opções de hardware, como por exemplo, tamanho de

armazenamento interno, cartão de memória, câmera, memória *RAM*, processador e o tipo de dispositivo propriamente dito, levando em conta os modelos existentes, também opções de software, como a versão do *Android* a ser emulada (Google, *Android*: *AVD Manager*, 2014).

2.3. Estudo de Caso

O protótipo que será demonstrado para a plataforma *Android* intitulado *ECarManager* tem a finalidade de manter cadastros de usuários, seus respectivos veículos, os custos relacionados ao veículo como despesas, serviços, abastecimentos, também pontos no Mapa, de forma que, sejam localizados os postos e os preços dos combustíveis atualizados por data. Através dos dados de abastecimento é gerado um gráfico contendo o histórico diário, mensal e anual. O aplicativo visa o gerenciamento destas informações e serve como exemplo de aplicação com banco de dados em nuvem para a plataforma *Android*.

2.4. Requisitos Funcionais

Requisitos funcionais são funcionalidades que o sistema deve fornecer, como deverá reagir a partir de entradas específicas e como deverá reagir caso determinada situação aconteça. Depende do tipo de software que será desenvolvido e o tipo de usuários que irão utilizar o sistema e deve-se sempre manter a consistência evitando contradições na definição (SOMMERVILLE, 2003 p. 83 e 84). Abaixo estão listados alguns dos requisitos funcionais encontrados no aplicativo desenvolvido:

- Manter veículo;
- Manter abastecimentos;
- Manter despesas;
- Manter serviços;
- Gerar gráfico de abastecimentos;
- Manter usuário
- Manter pontos no mapa;

2.5. Requisitos Não Funcionais

Requisito não funcional define as propriedades e restrições sob as funcionalidades ou serviços que são oferecidas pelo software. Pode estar relacionado à segurança, tempo de resposta e até mesmo espaço em disco, levando em conta o sistema como um todo (SOMMERVILLE, 2003 p. 83 e 85). Abaixo estão listados alguns dos requisitos não funcionais encontrados no sistema:

- Segurança e autenticação utilizando *login* e senha;
- Portabilidade plataforma *Android*;
- Desempenho do SGBD *MySql*;
- Consistência de dados.

2.6. Diagrama de Caso de Uso

Um modelo de caso de uso descreve como diferentes tipos de atores interagem com o sistema para resolver um determinado problema. Pode ser representado por uma descrição simples utilizando uma linguagem natural, identificando os objetos e o que o sistema deverá fazer (SOMMERVILLE, 2003). A Figura 25 representa o diagrama de caso de uso referente ao ator e os módulos do aplicativo para o dispositivo móvel.

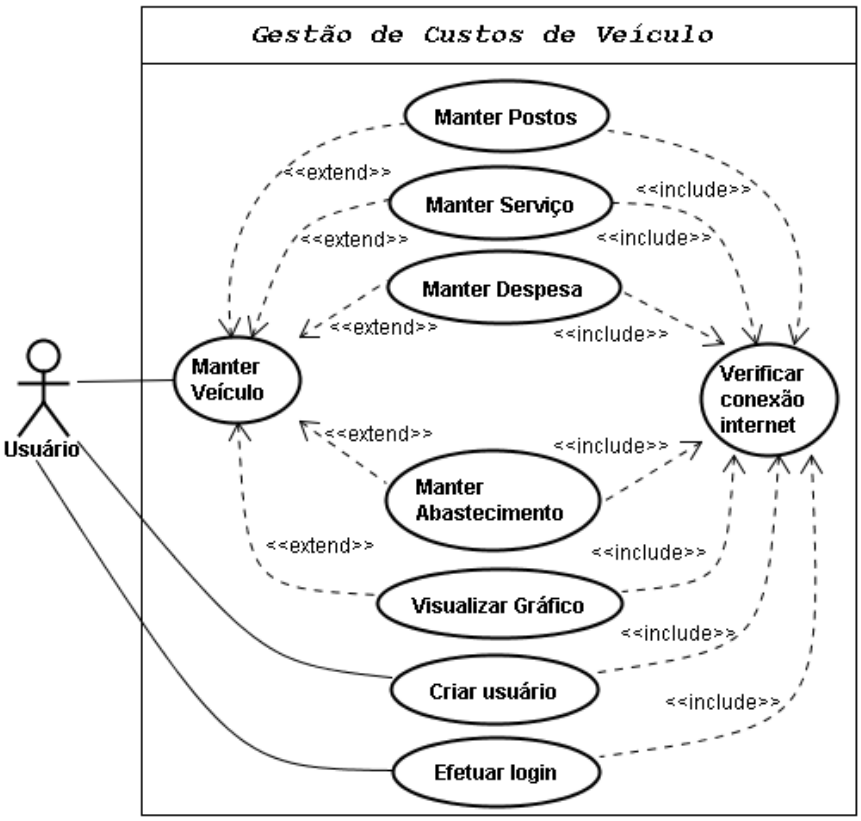


Figura 25: Representação do Diagrama de Caso de Uso
(Fonte: Autor, 2014)

No primeiro acesso, o usuário deverá fazer um cadastro de usuário. Ao finalizar o cadastro será redirecionado para outra tela de cadastro e visualização de veículos, na qual será necessário cadastrar um veículo para gerenciar.

Ao selecionar o veículo para gerenciamento, deverá ser visualizado o menu principal que apresentará opções para gerenciar despesas, abastecimentos e serviços para o veículo selecionado.

Ao selecionar qualquer uma das opções, como despesas, abastecimento ou serviço do veículo, o usuário poderá ter acesso às manipulações de dados CRUD (*Create, Read, Update, Delete*), para que deixe o controle total na mão do usuário. Após realizar a alteração necessária, a aplicação terá a opção de retornar à tela de menu para seguir conforme demanda.

Após efetuar os registros dos valores de abastecimento, o usuário também terá a opção de análise de valores por período através de gráficos, contando com análises diárias, mensais e anuais.

O aplicativo também contará com serviço de localização, baseado em coordenadas geográficas, disponibilizada pelo Google Maps, com as opções de inserção de pontos no mapa para demarcar a localização de postos e valores de combustíveis, o que poderá ser compartilhado com todos os usuários. A Figura 26 representa um estudo que foi feito inicialmente para demonstrar a estrutura das principais telas.

1) Menu principal: escolha da tela para a qual deseja abrir
2) Tela de manipulação de dados CRUD.
3) Tela de Geolocalização, utilizando a estrutura do Google Maps.
4) Tela de Análise de custos por período, representado por gráficos.

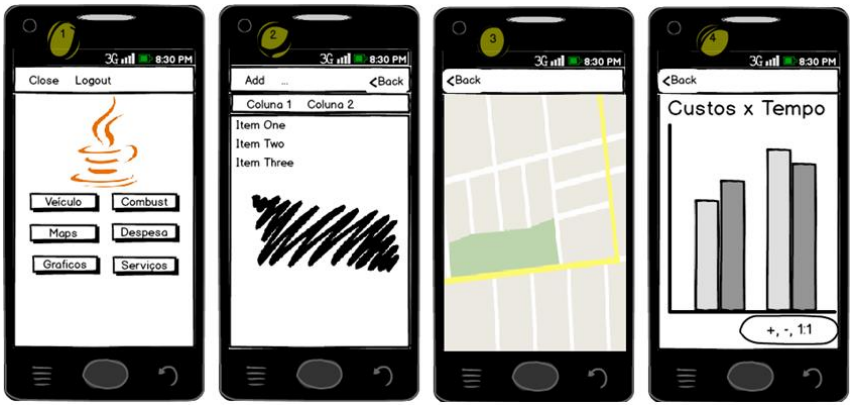


Figura 26: Estudo inicial das principais telas do aplicativo
(Fonte: Autor, 2014)

O aplicativo funcionará somente em conexão com a internet (*online*). Qualquer tipo de manipulação de dados só será concretizado em um repositório de dados remoto criado em um servidor em nuvem alocado como serviço pela Amazon, motivo pelo qual terá uma validação de conexão com a Internet a cada atividade acessada.

O sistema estará preparado para funcionar em dispositivos móveis com a plataforma *Android* versão 4.0 ou superior.

2.7. Banco de Dados

É necessário ter um banco de dados para armazenamento das informações dos veículos para listar e manipular informações. Desta forma elaborou-se um projeto de banco de dados. A Figura 27 mostra o diagrama Entidade-Relacionamento, no qual podem-se verificar as entidades e a disponibilidade dos campos, bem como suas relações.

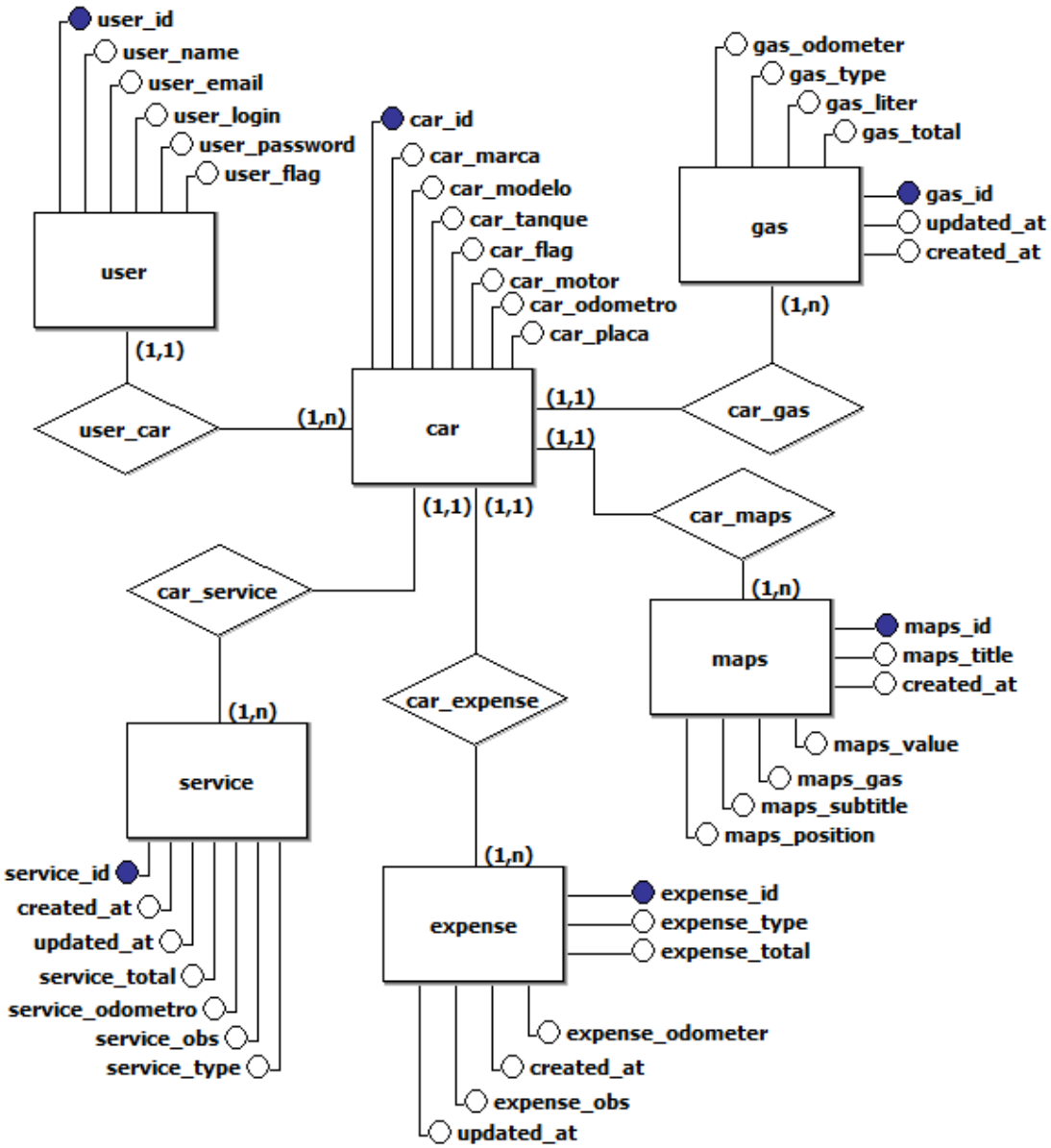


Figura 27: Diagrama Entidade-Relacionamento
(Fonte: Autor, 2014)

Este banco de dados relacional será criado no servidor *MySQL*, utilizando o Sistema operacional Ubuntu, alocado como serviço pela Amazon Web Service, para que se consiga maior persistência e consistência na manipulação dos dados.

2.8. Implementação

O nome do projeto é *ECarManager*, que, para a implementação foi utilizada a linguagem nativa para desenvolvimento *Android*, o *Java*, e para criação da estrutura visual de telas, botões, campos de texto e outros componentes da interface de usuário, utilizou-se a linguagem XML. A Figura 28 mostra a estrutura de diretórios da *IDE Eclipse*, na qual pode-se observar as classes e as interfaces do projeto desenvolvido, bem como a sequência 1, 2 e 3, sendo que uma fica abaixo da outra. A estrutura de diretórios é dividida em dois diretórios muito importantes, que são *src* (*source*) e o *res* (*resource*).

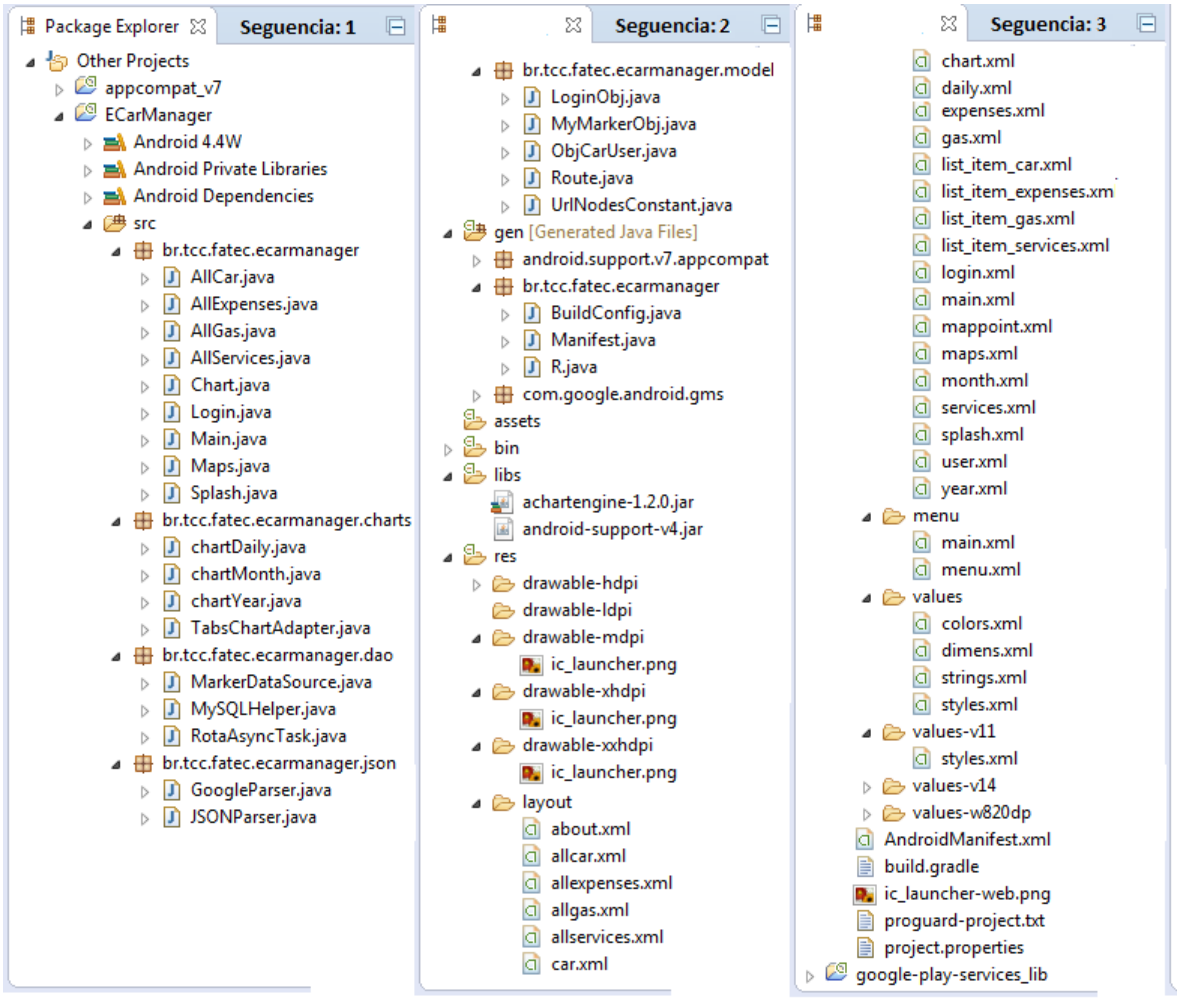


Figura 28: Estrutura de diretórios do projeto, IDE Eclipse
(Fonte: Autor, 2014)

O *src* é o diretório onde ficam todos os pacotes e classes *Java* do projeto. O pacote *ecarmanager* é onde ficam todas as classes principais do projeto. O pacote *charts* é onde ficam classes que irão manipular o gráfico contido no aplicativo. O pacote *dao* é onde ficam os arquivos de manipulação do *Google Maps*. O pacote *JSON* para é onde ficam as requisições *HTTP*. Por último, o pacote *model* é onde ficam todos os objetos e constantes.

No diretório *res*, localizam-se as imagens e arquivos XML. Os ícones e imagens ficam nos diretórios *drawables*. As interfaces de usuário ficam em *layout*, com os menus. Por fim, e não menos importante, o diretório *values* é onde ficam os arquivos de externalização, que são: *strings.xml*, *dimens.xml*, *colors.xml* e *styles.xml*, que armazenam as variáveis que podem ser reutilizadas em todo o projeto.

Para o atendimento às compatibilidades entre diferentes *APIs*, definem-se os diretórios *values-v11*, *values-14* e *values-w820dp*, o que quer dizer que se o aplicativo for rodar em um *Android* de API 11 irá referenciar os valores do arquivo de *styles.xml* que está dentro do diretório *values-v11*, caso seja API 14 irá referenciar o arquivo *styles.xml* do diretório *values-v14*, e por último caso esteja rodando em um dispositivo que tenha a tela com tamanho *width* 820dp irá utilizar os valores que estão dentro do arquivo *dimens.xml,* que está dentro do diretório values-w820dp, e assim por diante. Esta técnica pode ser adotada para *layouts* com diferentes tamanhos e orientações, bastando somente ter um diretório para a versão ou tamanho de tela que será necessário para o desenvolvimento.

O diretório *gen* é onde está a classe *R.java*, gerada automaticamente, que irá referenciar todos os componentes e classes do projeto.

O ECarManager está referenciado a bibliotecas externas importadas ao projeto. Conforme Figura 29 pode-se observar a biblioteca *appcompat_v7,* que tem por função adicionar classes e códigos básicos/padrão ao criar um projeto ou uma atividade, e o *google-play-services_lib*, onde estão as bibliotecas para manipulação do *Google*

Maps. A *IDE Eclipse* consegue dar suporte para realizar o *download* destas bibliotecas, através do *Android SDK Manager*.

Cada classe *Java* está associada a um arquivo *XML*. Na Figura 29 pode-se observar a classe *Main.java,* o método onCreate e o método setContentView.

```
public class Main extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
```

Figura 29: Classe Main.java do aplicativo, IDE Eclipse
(Fonte: Autor, 2014)

O método *setContentView* irá colocar a atividade no topo da pilha do ciclo de vida do aplicativo. Ao chamar este método, deve-se passar como parâmetro o *layout* utilizado. Para isto, na classe Main encontra-se o *R.layout.main*, onde R que irá referenciar o *resource layout*, e dentro deste tem-se acesso ao arquivo *main.xml*. Há o método *onCreateOptionsMenu* que irá inflar um menu na barra superior do *layout* que foi criado no método *onCreate*.

2.9. Classes Assíncronas de Manipulação de Dados

Quando uma aplicação está funcionando, inicia-se uma classe principal que mantém os recursos visuais para o usuário. Porém, quando há a necessidade de consultas em banco de dados externos, pode se levar tempo para resgatar os dados. Para isto é necessário criar uma classe assíncrona que funcione sem interferir da classe principal, para que aplicação não fique travada. A Figura 30 ilustra uma classe implementada na tela de usuário do projeto, chamada *AsyncTask,* que tem por função principal cadastrar um usuário na base de dados do *MySQL* no servidor em nuvem (LECHETA, 2014 p. 407-415). Esta classe possui três métodos principais, que são

onPreExecute, doInBackground e *onPostExecute*. No método *onPreExecute*, inicia-se um componente de diálogo que informa o que o aplicativo está executando para o usuário naquele momento. No método *doInBackground* encontra-se a lista de parâmetros a cadastrar, referenciados nas *Views* do projeto, a requisição *HTTP* passando para a classe de requisições a URL, o método e os parâmetros, e uma condição que testa se o registro foi feito com sucesso. O método *onPostExecute* fecha o componente de diálogo de *status*.

```
class CreateNew extends AsyncTask<String, String, String> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        pDialog = new ProgressDialog(Login.this);
        pDialog.setMessage(getString(R.string.createU));
        pDialog.setIndeterminate(false);
        pDialog.setCancelable(true);
        pDialog.show();
    }
    protected String doInBackground(String... args) {
        try {
            List<NameValuePair> params = new ArrayList<NameValuePair>();
            params.add(new BasicNameValuePair(
                    UrlNodesConstant.TAG_NAME,nome    .getText().toString()));
            params.add(new BasicNameValuePair(
                    UrlNodesConstant.TAG_EMAIL,email    .getText().toString()));
            params.add(new BasicNameValuePair(
                    UrlNodesConstant.TAG_LOGIN,user    .getText().toString()));
            params.add(new BasicNameValuePair(
                    UrlNodesConstant.TAG_PASSWORD,password    .getText().toString()));

            JSONObject json = jParser.makeHttpRequest(
                    UrlNodesConstant.url_create_user,"POST", params);
            Log.d("Create Response", json.toString());

            int success = json.getInt(UrlNodesConstant.TAG_SUCCESS);
            if (success == 1) {
                car_user = new ObjCarUser(userUser.getText().toString());
                Intent intent = new Intent(getApplicationContext(), AllCar.class);
                Bundle bundle = new Bundle();
                bundle.putSerializable("obj", car_user);
                intent.putExtras(bundle);
                startActivity(intent);
                finish();
            } else { return json.toString(); }
        } catch (JSONException e) { e.printStackTrace();}
        return null;
    }
    protected void onPostExecute(String file_url) { pDialog.dismiss(); }
}
```

Figura 30: Classe de criação de usuário, método assíncrono
(Fonte: Autor, 2014)

2.10. Conexão Com o Banco de Dados

A Figura 31 ilustra a classe *JSONParser* que faz a requisição *HTTP* no servidor de banco de dados. O método, chamado *makeHttpRequest*, recebe os parâmetros enviados pela classe assíncrona. Este método, através da URL informada, realizará uma requisição *GET* ou *POST* a um arquivo com extensão *PHP* localizado no servidor *MySQL*, que irá fazer a requisição e retornar um *array JSON*, assim a tela do usuário será atualizada com os dados (LECHETA, 2014, p. 584-599).

```
public class JSONParser {

    static InputStream is = null;
    static JSONObject jObj = null;
    static String json = "";

    public JSONParser() {}
    public JSONObject makeHttpRequest(String url,
            String method, List<NameValuePair> params) {
        try {
            if(method == "POST"){
                DefaultHttpClient httpClient = new DefaultHttpClient();
                HttpPost httpPost = new HttpPost(url);
                httpPost.setEntity(new UrlEncodedFormEntity(params));

                HttpResponse httpResponse = httpClient.execute(httpPost);
                HttpEntity httpEntity = httpResponse.getEntity();
                is = httpEntity.getContent();
            }else if(method == "GET"){
                DefaultHttpClient httpClient = new DefaultHttpClient();
                String paramString = URLEncodedUtils.format(params, "utf-8");
                url += "?" + paramString;
                HttpGet httpGet = new HttpGet(url);

                HttpResponse httpResponse = httpClient.execute(httpGet);
                HttpEntity httpEntity = httpResponse.getEntity();
                is = httpEntity.getContent();
            }
        }
        catch (UnsupportedEncodingException e) {    e.printStackTrace(); }
        catch (ClientProtocolException e) { e.printStackTrace(); }
        catch (IOException e) { e.printStackTrace(); }

        try {
            BufferedReader reader = new BufferedReader(
                    new InputStreamReader(is, "iso-8859-1"), 8);
            StringBuilder sb = new StringBuilder();
            String line = null;
            while ((line = reader.readLine()) != null) {
                sb.append(line + "\n");
            }
            is.close();
            json = sb.toString();
        } catch (Exception e) {
            Log.e("Buffer Error", "Error converting result " + e.toString()); }
        try {
            jObj = new JSONObject(json);
        } catch (JSONException e) {
            Log.e("JSON Parser", "Error parsing data " + e.toString());
        }
        return jObj;
    }
}
```

Figura 31: Classe de requisições *HTTP*
(Fonte: Autor, 2014)

2.11. Interface Gráfica

A interface gráfica foi desenvolvida com bases em estudos de interface homem computador, respeitando uma avaliação heurística. A avaliação é feita por profissionais que realizam a inspeção da interface para identificar problemas de usabilidade. Dentre as possibilidades de avaliação pode-se citar a visibilidade de *status* do sistema, relacionamento entre a interface do sistema e o mundo real, liberdade e controle de usuário, consistência, prevenção de erros, reconhecimento ao invés de lembrança, flexibilidade e eficiência de uso, estética e *design* minimalista (NIELSEN, 1995).

A Figura 32 ilustra as telas de *Splash* que constituem a abertura do sistema, a tela de *Login* para que o usuário possa se autenticar ou se cadastrar no sistema, e a tela de gerenciamento e manipulação de registros de veículos.

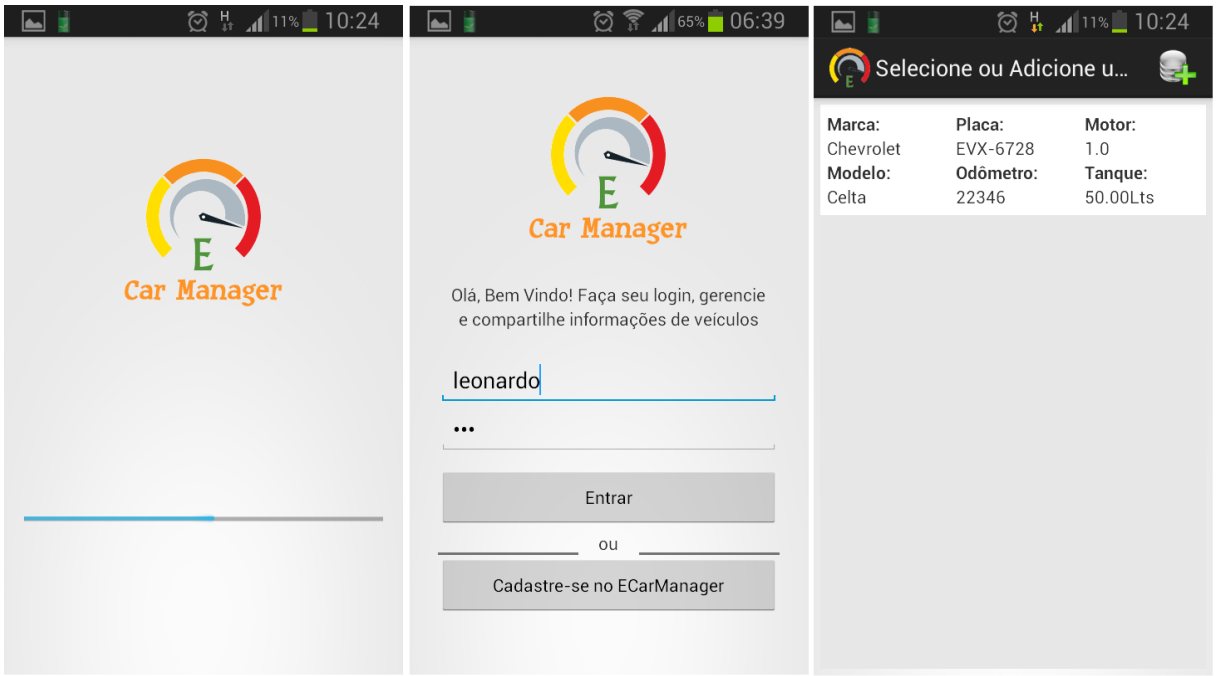


Figura 32: Telas: Splash, Login e Veículo
(Fonte: Autor, 2014)

A Figura 33 ilustra o momento em que o usuário seleciona um veículo, quando o mesmo irá receber a seguinte pergunta “O que gostaria de fazer?”, fazendo com que o usuário tenha que tomar uma decisão quanto ao que será feito.

Caso selecione “Gerenciar Veículo”, o usuário será redirecionado para a tela principal onde terá acesso a todas as telas de gerenciamento de custos do veículo, localização de postos, análise por gráfico e a tela sobre o sistema.

A última é a tela de “Sobre o Sistema”, que traz informações importantes, como versão do aplicativo, desenvolvedor e data de atualização.

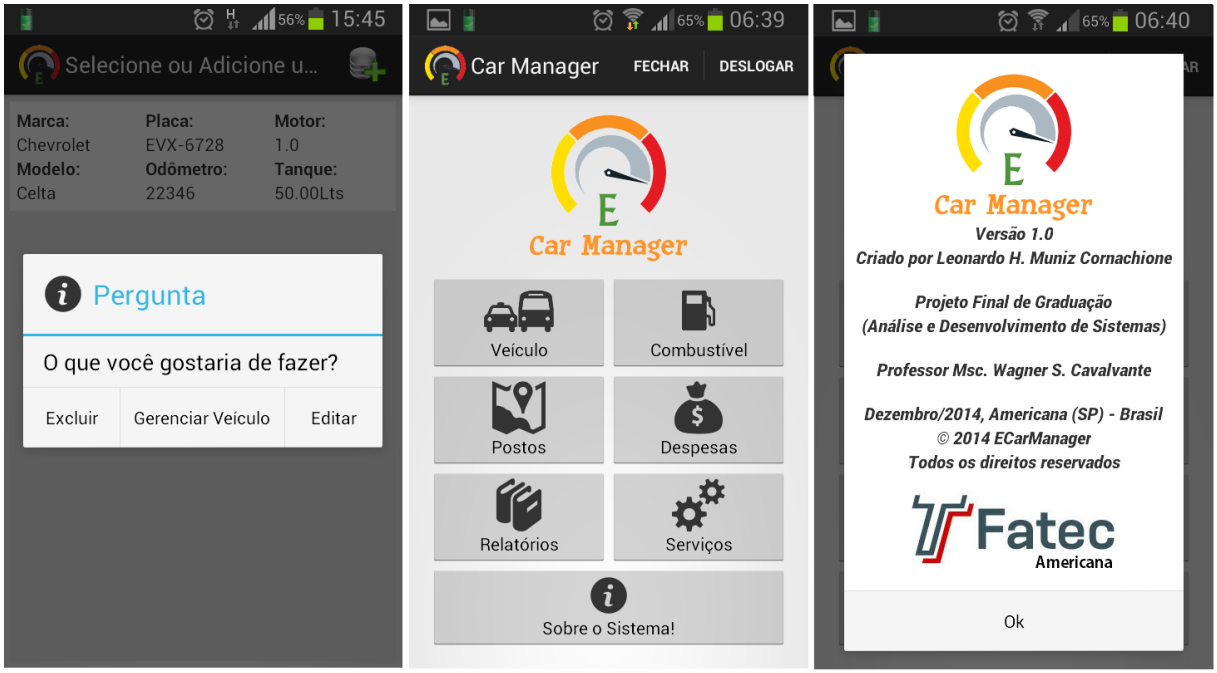


Figura 33: Telas: Gerenciamento Veículo, Principal e Sobre
(Fonte: Autor, 2014)

A Figura 34 ilustra as telas de gerenciamento de custos do veículo, que são: cadastros de combustíveis, despesas e serviços, que é definido por um histórico de data de criação.

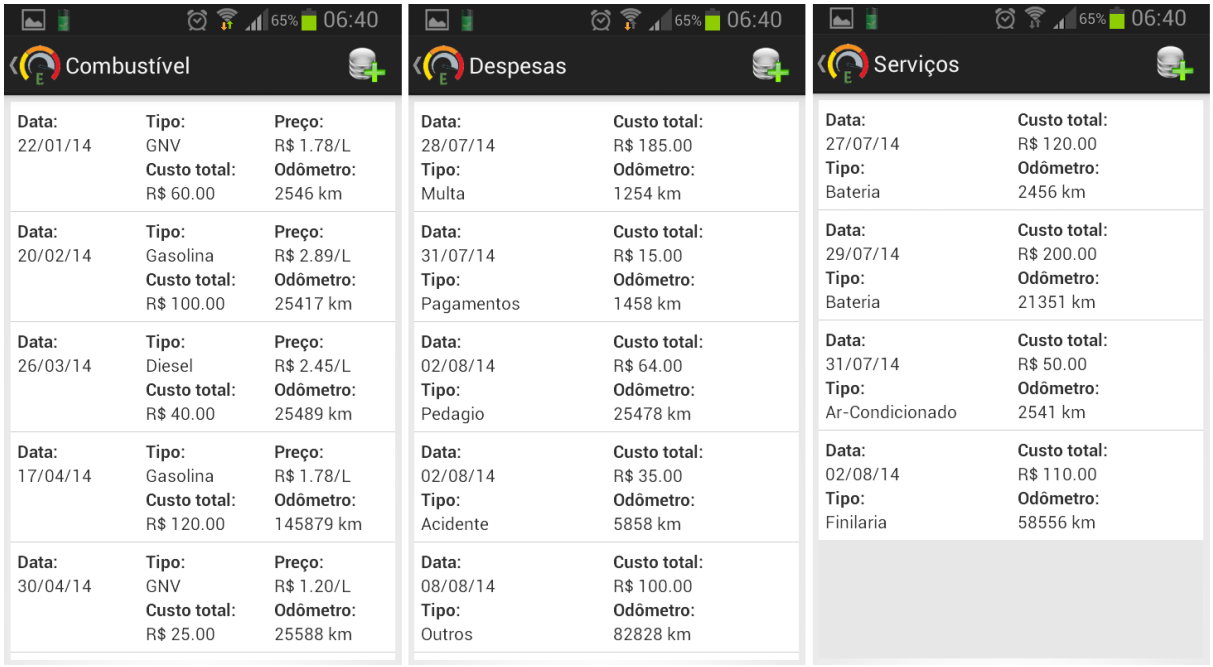


Figura 34: Telas: Combustível, Despesas e Serviços
(Fonte: Autor, 2014)

A Figura 35 ilustra o gráfico de histórico de abastecimentos onde o usuário pode obter análises diárias delimitadas ao mês corrente, análise mensal também delimitado do ano corrente, e análises Anuais.

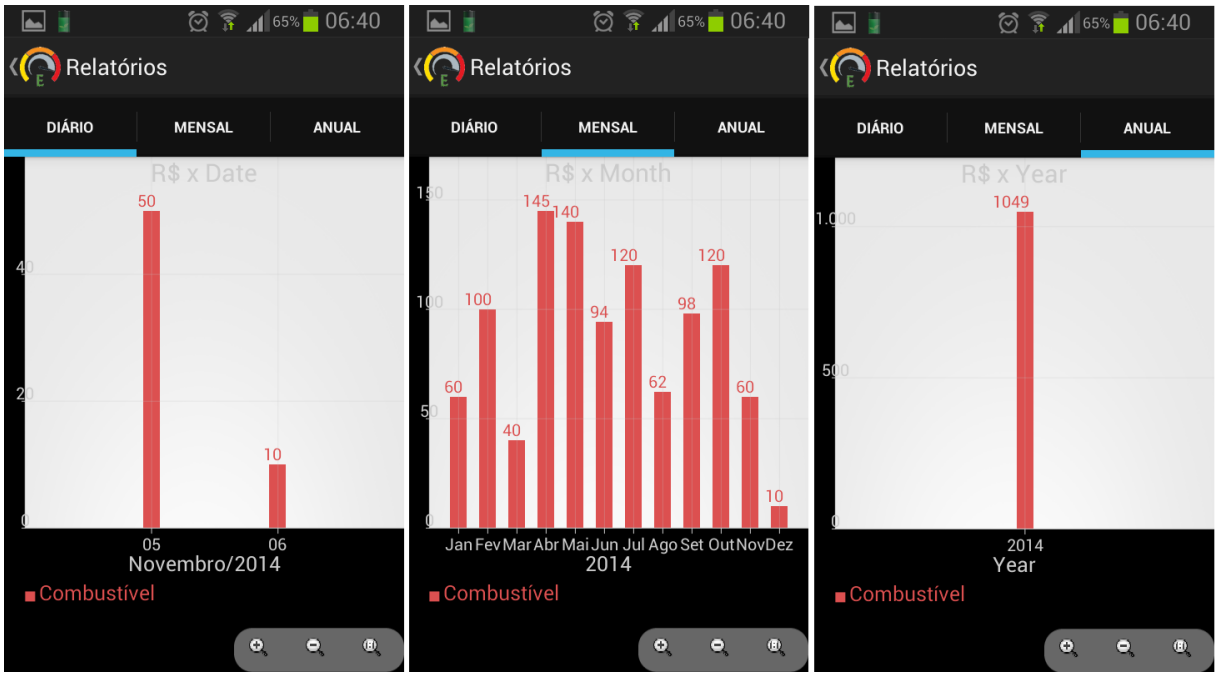


Figura 35: Telas: Gráfico de combustível, Diário, Mensal e Anual
(Fonte: Autor, 2014)

A Figura 36 ilustra o exemplo da tela de cadastro de veículo, do Mapa e tela de *status*. Em todas as telas de cadastro e gerenciamento de registro nota-se que há um botão na barra superior que tem por função abrir uma tela com campos específicos para o usuário adicionar as informações e criar um novo registro.

A tela do Mapa serve para adicionar um posto de combustível no local onde o usuário está, através do GPS do dispositivo. Para isso é preciso pressionar o botão “incluir”, na barra superior. Caso o usuário já tenha saído do local onde estava o posto e não fez a marcação, o mesmo pode navegar pelo mapa e escolher o ponto que quer adicionar para aquele posto, e, para adicionar apenas é preciso pressionar o ponto e segurar por três segundos.

A última tela é de *status*, que informa o que está ocorrendo enquanto o usuário aguarda, utilizada em telas de manipulação dos dados.

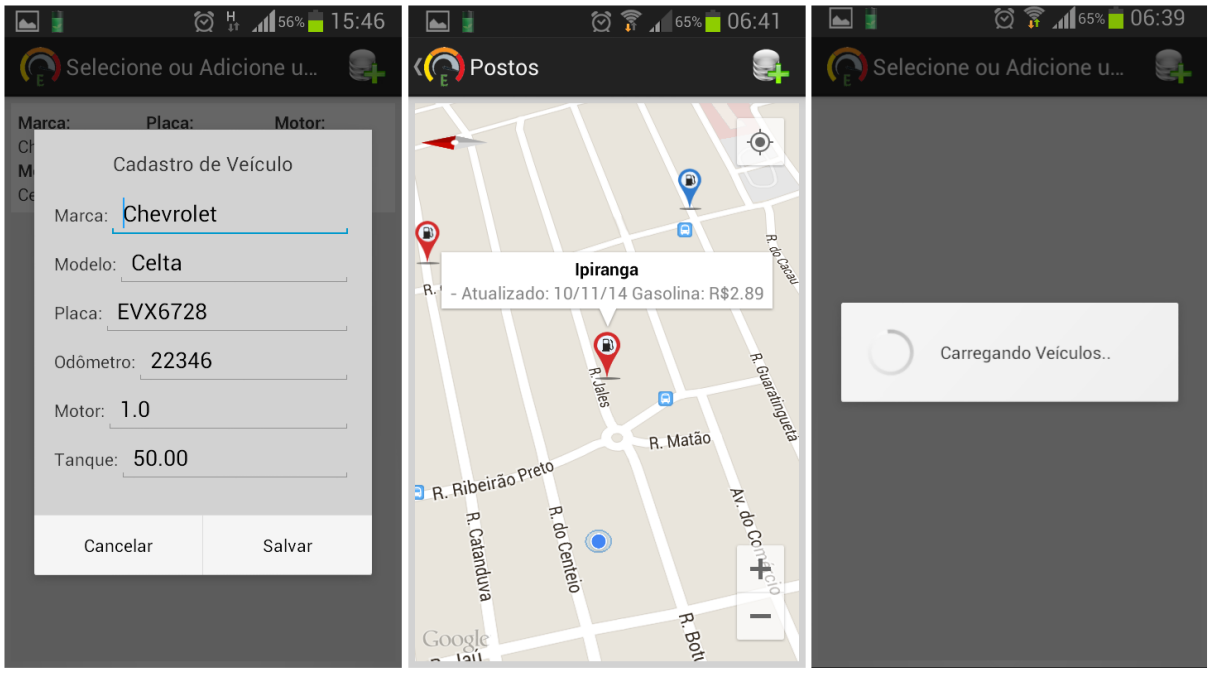


Figura 36: Telas: Cadastro, Google Maps, Carregando
(Fonte: Autor, 2014)

3. EVOLUÇÃO E DISCUSSÃO

Este capítulo trará um levantamento de padrões de projeto definido como boas práticas, consistência de dados e o que se espera para novas versões do aplicativo, o que pode ser melhorado e o que ainda não foi implementado.

3.1. Boas Práticas

Com o *JAVA* é possível criar e manipular componentes e *Views* programaticamente sem a utilização de arquivos XML. O recomendado é ter a associação de arquivos *JAVA* e *XML*, para manter padrões de desenvolvimento, desta forma, considera-se uma boa prática.

Também é considerada uma boa prática a externalização de arquivos *XML*. Como citado na Figura 28, o *resource values* contém arquivos XML de atributos que são acessados por arquivos XML ligados a uma interface de usuário e que podem ser reutilizados em outras interfaces.

Vale acrescentar, conforme Figura 30, a *TAG_NAME, TAG_EMAIL, TAG_LOGIN*, *TAG_PASSWORD*, *TAG_SUCCESS* e *url_create_user*, são constantes declaradas em uma classe abstrata, e que são acessadas em mais de uma classe de manipulação de dados.

O projeto foi desenvolvido respeitando as práticas citadas que são padrões em todos os projetos, dispondo de reutilização de código e boas práticas de projeto, porém há muitas formas de implementar, o que poderia ser alterado, dependendo do projeto ou do objetivo final da aplicação.

3.2. Consistência de Dados

A consistência de dados deste projeto foi testada e provada, uma vez que os dados cadastrados são acessados normalmente do banco de dados em nuvem. São

fidedignos, proporciona coerência, confiabilidade e gestão das informações de veículos.

3.3. O que pode se esperar para novas versões

Para novas versões do aplicativo pode-se esperar aprimoramentos na interface com o usuário, de modo que se adapte à nova versão do *Android* (Versão 5.0, de codinome *Lollipop*), uma vez que será uma versão com interface inovadora e que promete novos recursos visuais.

Outra futura inovação seria a de proporcionar opções de cadastro utilizando informações de rede sociais, como *Facebook* e *Google Plus*, deixando o aplicativo mais atraente e fácil de usar. Podem-se inserir novas análises por gráfico, de forma que o usuário tenha informações mais completas e sumarizadas, como, por exemplo, análises de rendimento de quilômetros por litro e também reincidências de gastos anuais.

Pode-se criar um banco de dados local (no dispositivo) para se ter acesso aos recursos *off-line* e, também, tornar possível um sincronismo com o banco de dados em nuvem para que os dados sempre estejam atualizados e acessíveis a outros dispositivos.

Pode ser criado um agendamento de lembretes, como troca de óleo do veículo ou manutenções preventivas, carregar pontos de postos no mapa de fontes confiáveis, como o site da Agência Nacional de Petróleo (ANP).

Por fim, e não menos importante, pode-se trabalhar na internacionalização do aplicativo para que possa ser utilizado virtualmente em qualquer parte do planeta (*anywhere*), traduzindo-o para línguas estrangeiras, pois onde houver veículos, haverá a necessidade de controlar custos de forma eficiente.

4. CONSIDERAÇÕES FINAIS

Ao longo do desenvolvimento e estudos sobre a tecnologia móvel, pode-se observar que esta é uma área em crescimento e que promete um futuro de inovações tecnológicas, apresentando variadas tecnologias relacionadas, o que abre muitas opções para o profissional que quer ingressar nela.

Muitas empresas já possuem um aplicativo móvel utilizando a tecnologia *Android* devido o vasto campo de possibilidades de negócios e também por ter a maior quantidade de usuários utilizando a plataforma.

A escolha pela plataforma *Android* justifica-se por ser um Sistema Operacional com muitas capacidades, constantemente aprimorado e adaptado a diversos dispositivos de diversas marcas, devido ao fato de ser livre para alterar-se, o que o torna muito recomendado.

O projeto desenvolvido apresentou o sucesso almejado, apesar de ainda haver muitas possibilidades de evoluções e melhorias. Pôde-se observar como é feita a conexão e manipulação de um banco de dados nas nuvens, aderindo-o às melhores práticas de desenvolvimento do projeto, a fim de ter consistência das informações.

A partir de experimentos feitos com candidatos que utilizaram o aplicativo em seus dispositivos, estes comprovaram sua satisfação na utilização do aplicativo, promovida pela facilidade na manipulação das informações e diferenciais dentre outros aplicativos disponíveis no mercado, para auxílio a despesas e informações de veículos, abrindo um leque de novas opções.

5. REFERÊNCIAS

_____. **Referências**: NBR-6023/ago. 2002. Rio de Janeiro: ABNT, 2002.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **Citação**: NBR-10520/ago. - 2002. Rio de Janeiro: ABNT, 2002.

ABADI, D. J. **Data management in the cloud**: Limitations and opportunities. IEEE Data Eng. Bull. Disponível em: <ftp://ftp.research.microsoft.com/pub/debull/A09mar/abadi.pdf>. Acesso em: 5 nov. 2013. 17h55.

AMAZON. **Produtos e Serviços**. Disponível em: <*http*://aws.amazon.com/pt/products/>. Acesso em: 28 set. 2014. 21h54.

DATE, C. J. **Introdução a Sistemas de Bancos de Dados**. 8. ed. Rio de Janeiro: Elsevier, 2003.

ECMA International, **The JSON Data Interchange Format.** Disponível em: <*http*://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf> Acesso em 06 de out. de 2014.

GOOGLE. **Android: *Activity Lifecycle***. Disponível em: <*http*://developer.*Android*.com/reference/*Android*/app/Activity.html> Acesso em 17 de Set. de 2014 as 20h18.

GOOGLE. **Android: *Data Options***. Disponível em: <*http*://developer.*Android*.com/guide/topics/data/data-storage.html> Acesso em 18 de Set. de 2014 as 20h58.

GOOGLE. **Android: Input Controls**. Disponível em: <*http*://developer.*Android*.com/guide/topics/ui/controls.html> Acesso em 17 de Set. de 2014 as 19h38.

GOOGLE. **Android: *Low-Level System Architecture***. Disponível em: <*http*://source.*Android*.com/devices/> Acesso em 07 de Set. de 2014 as 22h15.

HOCK-SHUAN, Chua. **HTTP (*HyperText Transfer Protocol*)**. Disponível em: <*http*://www.ntu.edu.sg/home/ehchua/programming/webprogramming/*HTTP*_Basics.html>, Acesso em 1 de outubro de 2014 as 07h51.

IDC. **Smartphone OS Market Share, Q2 2014**. Disponível em: <*http*://www.idc.com/prodserv/smartphone-os-market-share.jsp> Acesso em 18 de Set. de 2014 as 22h55.

KEITH, Mike, SCHINCARIOL, Merrick. **Pro JPA 2.** 2nd ed. 2013.

LECHETA, Ricardo R. **Google Android: Aprenda a criar aplicações para dispositivos móveis com Android SDK**. 3º ed. São Paulo: Novatec, 2013.

LUCIEL, Tsuharesu. **Dossiê Android**: 3 anos de vida, e muitas datas para serem lembradas [História do *Android*]. Disponível em: <*http*://mobilidade.fm/geral/2010/11/historia-do-*Android*/> Acesso em 07 de set. de 2014 as 22h07.

MACÁRIO, Carla G. do N., BALDO, Stefano M. **O Modelo Relacional**. Disponível em: <*http*://www.ic.unicamp.br/~geovane/mo410-091/Ch03-RM-Resumo.pdf> Acesso em 09 de set. de 2014

NIELSEN, JAKOB. **10 Usability Heuristics for User Interface Design**. Disponível em: < *http*://www.nngroup.com/articles/ten-usability-heuristics/>, Acesso em 10 de nov. de 2014 as 20h40.

RABELLO, Ramon Ribeiro. **Android: Construindo layouts complexos em Android**. Disponível em <*http*://www.cesar.org.br/site/files/file/WM21_*Android_Layouts*.pdf>, Acesso em 18 de set. de 2014 as 22h33.

RABELLO, Ramon Ribeiro. **Android: Um novo paradigma de desenvolvimento móvel**. Disponível em <*http*://www.cesar.org.br/site/files/file/WM18_*Android*.pdf>, Acesso em 14 de set. de 2014 as 22h33.

RANGEL, Alexandre. **MySQL: Projeto Modelagem e Desenvolvimento de Banco de Dados**. 1º ed. Rio de Janeiro: ALTA BOOKS, 2004. p.5 e 32.

SERCHCIO.IN. **IaaS cloud computing platform guide for managers**. Disponível em: <*http*://searchcio.techtarget.in/tutorial/IaaS-cloud-computing-platform-guide-for-managers>. Acesso em: 24 set. 2014. 10h50.

SOMMERVILLE, Ian. **Engenharia de software. 6º ed. São Paulo Pearson Education Companion, 2003.** p. 83-85.

SOUZA, C. R. F. **Gerenciamento de Banco de dados em Nuvem**: Conceitos, Sistemas e Desafios. Disponível em: <*http*://www.es.ufc.br/~flavio/papers/sbbd2010.pdf>. Acesso em: 5 nov. 2013. 14h35.

TAURION, Cezar. **Cloud Computing**: Computação em Nuvem: Transformando o mundo da Tecnologia da Informação. 1ª ed. Rio de Janeiro: Brasport, 2009. p. 1-40.

TOSIN, Carlos. **Conhecendo o Android**. Disponível em: <*http*://www.dicas-l.com.br/print/20110503.html>, Acesso em 7 de set. de 2014 as 14h44.