UNIVERSIDADE FEDERAL DO PAMPA

LETÍCIA GUIMARÃES DA CUNHA

CHARGER BACKUP: UMA SOLUÇÃO PARA SALVAGUARDA DE DADOS

Bagé

2016

LETÍCIA GUIMARÃES DA CUNHA

CHARGER BACKUP: UMA SOLUÇÃO PARA SALVAGUARDA DE DADOS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia da Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia da Computação.

Orientador: Érico Amaral

Bagé

2016

.

C972c Cunha, Letícia Guimarães
CHARGER BACKUP: UMA SOLUÇÃO PARA SALVAGUARDA DE DADOS / Letícia Guimarães Cunha.
94 p.

Trabalho de Conclusão de Curso (Graduação)-- Universidade Federal do Pampa, ENGENHARIA DE COMPUTAÇÃO, 2016.
"Orientação: Érico Hoff Amaral".

1. Android. 2. Backup. 3. Dispositivos Móveis. I. Título.

CHARGER BACKUP: UMA SOLUÇÃO PARA SALVAGUARDA DE DADOS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia da Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: dia, mês e ano.

Banca examinadora:

_____

Prof. Dr. Érico Amaral

Orientador

UNIPAMPA

_____

Prof. Dr. Milton Roberto Heinen

UNIPAMPA

_____

Prof. Dr. Sandra Dutra Piovesan

UNIPAMPA

Dedico este trabalho a minha família, pois sem seu apoio, nenhuma conquista faz sentido. Principalmente a minha mãe que sempre esteve do meu lado em todos os momentos da minha vida.

AGRADECIMENTOS

Principalmente a minha mãe por todo o carinho, apoio e dedicação. Por me aguentar durante todo o estresse de realizar esse trabalho, sem nunca diminuir seu amor por mim. Essa conquista com certeza é sua também.

Ao meu namorado Guilherme por todo o seu amor, por ficar sempre ao meu lado, me manter calma e me dar tanta felicidade.

Ao meu pai, que sempre me falou que o estudo é a coisa mais importante da vida, queria poder dividir esta conquista com ele, grande amor da minha vida.

A minha família por todo o apoio e alegria, principalmente meu irmão e cunhada, que me deram o maior presente do mundo que é minha afilhada Maria Clara.

Aos meus colegas e amigos a melhor parte da faculdade é conhecer pessoas maravilhosas que irei levar para o resto da vida. Principalmente minhas grandes amigas Fernanda e Camila, que me ajudaram neste e em muitos outros trabalhos, mas que acima de tudo eu sei que posso contar sempre.

Ao meu orientador Erico, por todos os puxões de orelha, por sempre me ajudar e estimular a pesquisar e escrever artigos. Com certeza fez de mim uma aluna e profissional melhor.

Ao meu amigo Breno, eu queria muito desfrutar contigo minha formatura, tua partida repentina me deixou sem chão. Obrigada por fazer parte da minha jornada. Saudades eternas.

“O sucesso é a soma de pequenos esforços repetidos dia após dia.”

Robert Collier

RESUMO

O rápido crescimento da utilização de dispositivos móveis, nos últimos anos, disponibilizou vários recursos. Os quais atraem os usuários a armazenarem seus dados nestes aparelhos, necessitando assim uma maior preocupação com a segurança destas informações. A proposta desta pesquisa, portanto, é a criação de um modelo para a salvaguarda de dados, em dispositivos que utilizam a plataforma Android, visando a simplicidade, transparência e eficiência. Definem-se as técnicas para a realização de *backup* e restauração de dados, procedimentos de compactação e meios de armazenamento. Emprega-se neste projeto o método cientifico. Aplicando-se as técnicas de observação sistemática, descrição das técnicas observadas e comparação dos diversos métodos de salvaguarda utilizados atualmente. Em posse destas informações, é criado um modelo para *backup* de dados. Para realizar a implementação foi feito um levantamento dos requisitos necessários. Após a finalização da implementação, foi feita a validação do modelo, onde foram realizados testes e analisados os resultados. Através disto, pode-se demonstrar que o sistema é eficiente.

Palavras-Chave: Android, *Backup*, Dispositivos Móveis.

ABSTRACT

The rapid growth in the use of mobile devices in recent years has made available a number of features. Which attract users to store their data on these devices, thus intensifying concern about the security of this information. The proposal of this research is the creation of a model for the safeguarding of data, in devices that use the Android platform, aiming for simplicity, transparency and efficiency. Define techniques for performing backup and restoration of data, compaction procedures, and storage media. The methodology employed in this project makes use of the scientific method. Applying the techniques of systematic observation, description of the techniques observed and comparison of the various methods of safeguard currently used. In possession of this information, a model for data backup is created. To carry out the implementation a survey of the necessary requirements was made. After the implementation was completed, the validation of the model was performed, where the tests were performed and the results analyzed. Through this, it can be concluded that the objective of this work has been reached.

Keywords: Android, backup, mobile devices.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API - Application Programming Interface

BSS - Basic Service set

CD - Compact Disc

DVD - Digital Versatile Disc

FDD - Frequency Divisor Duplex

HD - Hard Disk

IDE- Integrated Development Environment

ISP- Internet Service Provider

JDK - Java Development Kit

PC - Personal Computer

RLE - Run-Length Encoding

SD - Secure Digital

SDK -Software Development Kit

TDD - Time Divisor Duplex

TXT- Text File

UML - Unified Modeling Language

URI – Uniform Resource Identifier

VFS – Virtual File System

XML - Extensible Markup Language

WAP - Wireless Application Protocol

Wi-Fi - Wireless Fidelity

WiMAX - Worldwide Interoperability for Microwave Access

SUMÁRIO

1 INTRODUÇÃO

Um dos principais meios de comunicação utilizados atualmente é os dispositivos móveis. Já que eles possuem um alto número de usuários que vêm crescendo a cada dia e com isso há cada vez há mais dados sendo transferidos e armazenados nestes aparelhos. Por possuírem enumeras aplicações, que podem ser instaladas e utilizadas tanto pessoalmente quanto profissionalmente, ou seja, pode-se armazenar todos os tipos de informações em seu sistema. Isto, somado a sua mobilidade faz com que eles sejam presença constante na vida de diversas pessoas. Nos últimos anos houve um grande aumento na sua fabricação, isto é, conforme descreve Hennessy e Patterson (2012), foram fabricados mais de 600 milhões de dispositivos em 2012, ultrapassando a fabricação de PC's (*Personal Computer*) em 2010. Os dispositivos móveis possuem diversas inovações e recursos que não eram possíveis em computadores desktop, atraindo assim cada vez mais os diversos tipos de consumidores. Entre estas inovações se destaca a mobilidade, por serem dispositivos pequenos podem ser levados a qualquer lugar. Esta vantagem aumenta sua vulnerabilidade, pois estão mais propícios a roubos, perdas e danos, como destaca Shiriwas *et al.* (2013)

O maior medo que se tem ao perder um dispositivo é a integridade dos dados contidos nele. Segundo Muslukhov *et al.* (2012), os usuários utilizam poucas ferramentas de proteção, porém a maior preocupação ao perder ou danificar um aparelho é com as informações contidas nele, pois, as chances de recuperação destes dados são quase nulas. Atualmente, são poucas as pesquisas realizadas para proteção de dados em dispositivos móveis. Em 2012 o Brasil possuía a segunda maior taxa de aparelhos roubados no mundo, conforme relatório da *F-secure*[1], portanto, os cuidados com a proteção de dados devem ser tratados com maior ênfase, pois, quase não há chances de recuperação de um aparelho roubado.

[1]http://canaltech.com.br/noticia/gadgets/Brasil-e-o-2o-pais-com-maior-numero-de-roubos-e-perdas-de-dispositivos-moveis/

Muitas vezes é possível evitar que os dados sejam acessados por pessoas não autorizadas, desta forma, o usuário pode tomar essas providências antes mesmo que ocorra a perda ou o roubo. Existem aplicativos de segurança que bloqueiam e até excluem o conteúdo dos dispositivos remotamente, porém não recuperam os dados perdidos. Por isso, é importante saber como proteger os dados. Segundo Fialho *et al.* (2007), um meio de garantir isto é realizando *backup*, ou seja, a cópia segura de dados é de suma importância, pois ela é a melhor forma de prevenção e recuperação das informações, assim os dados ficam guardados, podendo-se recuperá-los quando necessário. Existem hoje muitas formas de realizar *backups*, porém são poucas as pessoas que as utilizam, isto se deve à demora e complexidade desse tipo de operação. Encontram-se também dificuldades em escolher um local para depositar esses dados, isto é, que seja seguro e compatível com a disponibilidade dos usuários. Através de uma vasta pesquisa pode-se chegar a uma conclusão sobre que local de armazenamento é mais indicado para salvaguardar os dados, ou seja, com uma implementação que possibilite um *backup* mais eficiente e seguro. Assim, é possível estimular os vários utilizadores de dispositivos móveis a realizarem *backups* frequentes, evitando assim a perda de dados.

1.1 Problema de Pesquisa

É possível criar um método de *backup* simples e transparente, para o sistema operacional Android que evite danos à integridade das informações?

1.2 Objetivos Gerais

Esta pesquisa tem por objetivo realizar um estudo sobre métodos e técnicas para salvaguarda de informações armazenadas em dispositivos móveis, a fim de propor uma solução de backup que seja simples, rápida e transparente ao usuário. Questões como facilidade de utilização, profundidade

do backup, padrão de restauração e compatibilidade com o sistema operacional Android são elementos fundamentais para esta proposta.

1.3. Objetivos Específicos

- Realizar uma pesquisa sobre o Sistema Operacional Android;
- Realizar um estudo sobre métodos e técnicas para salvaguarda de informações;
- Realizar um estudo sobre métodos e técnicas para salvaguarda de informações aplicadas a dispositivos móveis;
- Analisar métodos para exportação de dados de *backups* em sistemas Android;
- Propor um método para o *backup* de informações em sistemas Android;
- Documentar e implementar a solução de *backup* proposta;
- Realizar testes e validar o sistema construído.

1.4. Estrutura do Texto

Neste trabalho será inicialmente abordado no capítulo 2 os conceitos gerais e revisão da literatura necessários para a realização deste trabalho. No capítulo 3 será definida a metodologia empregada. O capítulo 4 vislumbra a aplicação Charge Backup. No capítulo 5 são discutidos os resultados obtidos. Já no capítulo 6 são destacadas as conclusões. E no capítulo 7 são expostas as pretensões de trabalhos futuros.

2. CONCEITOS GERAIS E REVISÃO DE LITERATURA

Neste capítulo serão abordados todos os tópicos essenciais para a realização do modelo proposto nesta pesquisa. Na seção 2.1 será apresentada a Era da Informação e da Comunicação com a evolução dos meios de comunicação até os dispositivos atuais, na seção 2.2 aborda-se o sistema operacional Android e suas peculiaridades, já na seção 2.3 serão demonstradas algumas tecnologias de *backup* e restauração de dados, assim como seu funcionamento. Na seção 2.3.1 avalia-se as técnicas de compressão de dados, ainda na seção 2.3.2 são discutidos os meios de armazenamento. Na seção 2.3.3 são verificadas algumas soluções de *backup* para o Android e na seção 2.4 são mostrados alguns trabalhos correlatos.

2.1 A Era da Informação e da Comunicação

A comunicação é inerente ao ser humano, assim as primeiras manifestações que se têm notícias foram a cerca de 30.000 anos atrás. Anos mais tarde, aproximadamente no ano de 4.000 A.C., a civilização suméria criou a escrita, conhecida como cuneiforme, ela foi inventada para auxiliar o controle de produção, além de estreitar as relações de trabalho existentes na época. Assim, entre 3200 e 3100 A.C. na Mesopotâmia foram escritos os primeiros textos conhecidos. A invenção da escrita possibilitou que a informação pudesse se propagar de forma mais precisa, longínqua e até através dos anos, como vislumbra Amorim *et al.* (2011).

Desde a renascença documentos impressos foram a base do repositório das informações. Em 1860 foi criado o telefone e no século XIX a comunicação teve um grande avanço, devido ao início das transmissões de rádio que levavam a informação a uma velocidade muito rápida a vários receptores. Já em 1895 foi concretizado o cinematógrafo, princípio do cinema que

possibilitava a gravação de imagens em movimento. Em 1936, impulsionada pelo cinematografo, foi iniciada a emissão televisiva. Além disso, nos anos 1980 e 1990 inicia a era digital com a globalização e a *internet*, a comunicação passou a ser disseminada através de meios elétricos. Portanto, arquivos digitais foram criados para guardar as informações, iniciando assim a era tecnológica em que estamos hoje. (SIMONSO *et al.,* 2013).

Com o início da *internet* foi possível um compartilhamento mais rápido das informações, assim fazia-se necessária a evolução deste meio, que limitava seu acesso a um lugar fixo em que se conecta um cabo ao aparelho. Como os dispositivos se tornaram cada vez menores, foi necessário a criação de um meio de conexão para uma maior mobilidade, surge então *wireless*, uma telecomunicação multimídia sem fio. Possibilitando, portanto, a interação com qualquer pessoa em qualquer lugar e a qualquer momento, iniciando com a telefonia móvel e tornando-se um meio de acesso à *internet* com grande destaque atualmente, como afirma Hanzo *et al.*(2012). A figura 1 demonstra que com o uso da *internet* pode-se acessar vários meios de comunicação com a utilização de apenas um aparelho.

Figura 1- Acesso a meios de comunicação com um único aparelho.

Fonte: Milldesk, 2016.

Afaq *et al.* (2009), destaca que a primeira geração de sistemas de comunicação móvel (1G) começou a ser projetada em 1970, sendo que a maioria dos sistemas desta geração era baseada na FM analógica, na qual a voz era o tráfico principal, portanto, operava predominantemente na rádio a frequência de 450 MHz. Em 1980 iniciou o desenvolvimento da segunda geração (2G), que ainda era desenvolvida para a transferência de voz, porém era baseada na tecnologia digital e possuía uma melhor qualidade de som. Assim sendo, incluía o processamento digital de sinal e assim foi possível realizar a transferência de dados com uma velocidade de até 14,4Kbp (Kilobits, por segundo). Logo após, surgiu uma evolução da 2G que foi chamada 2,5G em que suportava o padrão WAP (*Wireless Application Protocol*). À vista disso, possuía uma taxa de transferência de dados de até 115 Kbp, assim os dados foram divididos em pacotes, possibilitando assim uma conexão permanente.

Devido ao grande aumento na procura por celulares com acesso à *internet* e com evolução da Banda Larga (conexão de alta velocidade), surge uma nova geração, a 3G, inovando as redes de comunicação móvel, possuindo taxa de transmissão superior a 144 Kbp e boa qualidade no acesso a informações. Emprega a técnica TDD (*Time Divisor Duplex*) permitindo que as transmissões *upload* e *download* ocorram na mesma faixa de frequência, utilizando intervalos sincronizados, dividindo o tempo entre transmissão e recepção. Assim sendo, pode aplicar também a técnica FDD (*Frequency Divisor Duplex*) que realiza as transmissões *upload* e *download* em taxas frequências diferentes e específicas, que são atribuídas a uma única conexão. Os aparelhos 3G automaticamente fazem a realocação de seu modulo de transmissão para utilizar a faixa de espectro (entre 1,9GHz e 2,1GHz), disponível na região. A geração mais recente é chamada 4G com conexões mais rápidas, na qual é possível assistir vídeos e realizar vídeo conferências com alta definição, assim a velocidade pode chegar a ser 10 vezes maior que a da geração anterior, dependendo do aparelho e da rede, podendo chegar a 50 Mbp (Megabits por segundo).

Segundo Fernandes (2010), existem outros meios para conexão sem fio além dos ligados a telefonia celular, o mais popular deles é o padrão *WiFi* (*Wireless Fidelity*), também conhecido como *Wireless Lan* ou IEEE 802.11. Portanto, ela permite que o usuário navegue na *internet* com velocidade de banda larga, sua arquitetura consiste em vários componentes que interagem para proporcionar uma *LAN* sem fio. A célula básica da IEEE802.11 é a BSS (*Basic Service set*), que é um conjunto de estações moveis ou fixas, é capaz de atingir distâncias de 100 metros e velocidade de 11 a 300Mbp. O novo padrão IEEE 802.11n (draft 2.0) para rede sem fio é aliado à tecnologia MIMO 2x2 (duas antenas para transmissão e recepção simultânea de informações). Portanto, uma evolução deste padrão é a WiMAX (*Worldwide Interoperability for Microwave Access*) que pode atingir velocidade de tráfego de até 1Gbp (Gigabit por segundo) em um raio de até 50km² em condições especiais, foi desenvolvida em 2001 por uma iniciativa da Intel e da Alvarion.

Com altas taxas de transferências de Informações sendo realizadas e, como destaca Oliveira *et al.*(2014), com a desmaterialização dos dados físicos existentes, é necessária a construção de repositórios digitais confiáveis. Na área de Gestão da Informação um dos objetivos principais é a certificação destes repositórios em contexto organizacional, sendo assim é de suma importância que o armazenamento seja seguro, de acesso controlado e preserve a informação. Frente a estes aspectos, uma tecnologia se destaca, isto é, o armazenamento em nuvem, pois para que o usuário acesse a nuvem é necessário apenas que ele possua um Sistema Operacional, um navegador e acesso à *internet*. Assim, não necessita de capacidade de armazenamento ou poder computacional, a nuvem é acessada através da rede e utiliza os servidores como um único arquivo, uma máquina virtual, permitindo movê-los de um hardware para outro.

Segundo Wahid *et al.* (2014), a computação em nuvem é uma grande aliada dos dispositivos móveis, desta forma, existem várias aplicações móveis que utilizam a computação em nuvem. Logo com a grande quantidade de informações dispostas nestes dispositivos, o armazenamento em nuvem se torna essencial, pois a capacidade nestes aparelhos é limitada. Os dispositivos móveis devido a capacidade de acesso à rede, com velocidades das conexões

3G, 4G e *WiFi* têm diversos tipos de informações inseridas em seu sistema. Hoje em dia existem bibliotecas virtuais e pode-se realizar todo tipo de pesquisas on-line, além disso, o próprio aparelho possui dispositivos para tirar fotos e criar vídeos, assim, o envio e o recebimento de dados ocorrem em larga escala. O Relatório da Cisco (2015), aponta que até 2018 será realizado o tráfego de 5,1 Exabytes de dados mensais nestes aparelhos, ou seja, três vezes mais do estimado em 2013.

Dentre os Sistemas Operacionais para dispositivos móveis, o Android é o mais utilizado, segundo dados da Canaltech (2015), 81% dos celulares no mundo fazem uso desta plataforma. Desta forma, é necessária a realização de pesquisas na área de segurança de informações, considerando que estes aparelhos são constantemente empregados no dia a dia das pessoas, além de conter diversos dados neles que têm valores irreparáveis ás vítimas de perdas.

2.2 Sistema Operacional Android

Como destaca Freire (2013), a primeira versão do Android foi lançada em 2008, as atualizações desenvolvidas a partir desta, foram realizadas para correções de bugs, inserção de novas funcionalidades e troca completa de gerenciamento interno de software e hardware. Assim sendo, quando a inovação é muito significativa, o nível de API (*Application Programming Interface*) cresce, esta graduação serve para os desenvolvedores saberem quais aplicações podem ser utilizadas na nova versão. Logo, cada nova composição comporta os níveis das anteriores, então uma nova nomenclatura é destinada a versão desenvolvida, estas são sempre nomes de doces e seguem a ordem alfabética, como mostra a figura 2.

Figura 2 - Versões do Android.

Fonte:http://optclean.com.br/google-atualiza-dados-de-distribuicao-do-android-marshmallow-sobe-para-75-em-maio-de-2016/(2016)

Segundo Lecheta (2013), o Android é um Sistema Operacional para dispositivos móveis, de código aberto, desenvolvido pela *Open Handset Alliance*(OHA)[2], um grupo formado por empresas com a intenção de criar padrões abertos para telefonia móvel, o qual é liderado pelo Google. Além disso, é baseado no *Kernel* 2.6 do Linux e se responsabiliza pelo gerenciamento da memória, os processos threads, rede, drivers e a segurança dos arquivos e pastas. Isso permite que várias aplicações possam ser executadas simultaneamente, possibilitando que algumas sejam realizadas em segundo plano, sem atrapalhar as operações efetivadas pelo usuário.

O sistema de arquivos Android consiste em várias partições, que são divisões lógicas do armazenamento de dados, alguns fabricantes modificam o sistema de partições, mas algumas são padrão do sistema como: *boot*, *system*, *data*, *cache* e a partição do *SDcard,* que são abstraídos do usuário pelo uso de um sistema de arquivos virtual (VFS). Segundo Huang & Wen (2012), a arquitetura do controle de acesso ao sistema de arquivos inclui:

- Módulo de Política de Obtenção: Quando uma aplicação solicita o acesso a um arquivo, este módulo captura esta solicitação e chama o Módulo de

[2] http://www.openhandsetalliance.com/

Comunicação do *Kernel,* isto é, para obter as informações de autorização correspondente ao arquivo;

- Módulo de Controle de Acesso: Depois que o Módulo de Política de Obtenção consegue as informações de autorização, o Módulo de Controle de Acesso autentifica esta licença, comparando o comportamento da operação com a informação de autorização. Ou seja, de acordo com a operação ser legal ou não, ele irá permitir ou negar o acesso;
- Módulo de Comunicação do *Kernel*: É considerado uma ponte entre o espaço do *Kernel* e o espaço do usuário. Portanto, troca mensagens com o Módulo de Gerenciamento de Políticas e recebe as informações de política a partir do espaço do usuário;
- Módulo de Gerenciamento de Políticas: Assim que receber a mensagem do *Kernel*, ele irá analisar o arquivo de política para descobrir as regras de segurança correspondentes, em seguida, enviará as informações da política de volta ao *Kernel*;
- Política de Arquivos: É a função que guarda a Política de Segurança para Controle de Acesso de Arquivos, assim, desempenha um papel importante no sistema de controle de acesso a arquivos Android, portanto, ele contém todas as regras de segurança no sistema. O sistema Android será ameaçado seriamente, uma vez que, a Política de Arquivos seja modificada violentamente.

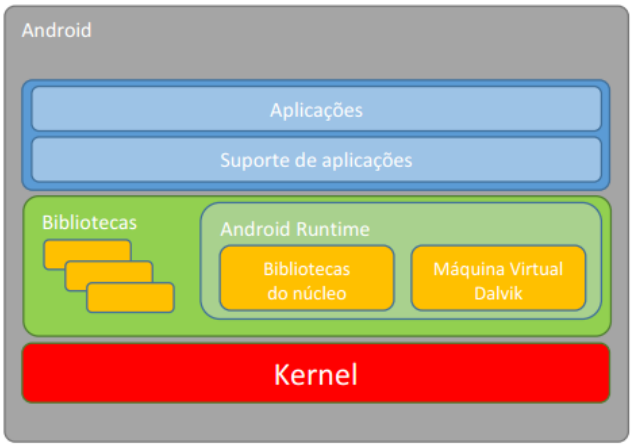
Freire (2013), destaca que a arquitetura Android é dividida em três camadas, como demonstra a figura 3:

- A primeira camada é de aplicações, ela subdivide-se em aplicações e suporte de aplicações, portanto, nesta camada encontram-se todos os programas nativos;
- *Middleware* é a camada intermediária, nela se encontram bibliotecas básicas em C/C++ (*bionic libc*, biblioteca de funções, biblioteca de servidores nativos e biblioteca de abstração de hardware). Dentre as funções contidas na biblioteca estão as de acesso ao banco de dados

SQLite, além disso, esta camada possui também, um Android *Runtime* (bibliotecas nativas de Java e a Máquina Virtual Dalvik).

- A última divisão é o Sistema Operacional, nesta camada estão os programas de gerenciamento de memória, configurações de segurança, gerenciamento de consumo energético, acesso ao sistema de arquivos, comunicação em rede, comunicação entre processos e os drives, contidos no Kernel.

Figura 3 - Arquitetura do Android.



Fonte: Freire (2013)

Para desenvolver aplicações no Android utiliza-se o kit de desenvolvimento Android SDK, isto é, é um conjunto de bibliotecas, documentos, utilitários e compiladores necessários para a codificação, compilação, teste e distribuição de aplicativos. Assim sendo, para a construção de programas nesta plataforma é utilizada a linguagem Java, porém a máquina virtual utilizada não é a JVM e sim a máquina virtual Dalvik que é otimizada para execuções em dispositivos móveis. Em vista disso, é gerado um código Dalvik executável com formato dex ao invés do *byte-code* de formato. class do Java. Após o arquivo .dex e outros recursos como imagens são compactados em um arquivo com extensão .apk (*Android Package File*), que é o arquivo de instalação da aplicação.

Conforme Pereira *et al.* (2009), na instalação de um novo aplicativo, é criado um usuário Linux para este programa, em que serão selecionados os

diretórios que serão utilizados apenas para este aplicativo. Desta forma, as aplicações são isoladas, ou seja, não se podem acessar as informações contidas em outras aplicações. Portanto, para que isso ocorra é necessária a criação de uma autorização do usuário, que pode aceitá-la ou não no momento da instalação da aplicação, além disso, na escolha de negar esta licença a instalação é cancelada. Ao desenvolver uma aplicação no Android é necessário verificar quais as permissões são necessárias para a execução das funções implementadas para esta aplicação. Desta maneira, cada permissão controla um conjunto de funções de API do Android ou Java, contudo, o acesso a estas funções sem a permissão adequada gera uma exceção de segurança. Quaisquer acessos a serem realizados, por exemplo, em outros aplicativos como *internet*, arquivos e etc. necessitam ser declarados nas permissões que serão apresentadas ao usuário no momento da instalação. Além disso, essas permissões são mostradas com um título, no qual é destacada sua categoria geral, um subtítulo que indica uma breve descrição de suas ações. Assim ao clicar em cima de alguma delas, o usuário pode ter um detalhamento sobre a permissão e visualizar exemplos de como esta aplicação pode abusar de seu acesso. Ademais no momento da compactação o arquivo .apk deve conter também um manifesto em XML em que são encontradas as permissões, conforme descreve Bakar & Mahmud (2013).

2.3 Tecnologias de *Backup*

O *backup* consiste em uma cópia segura dos dados encontrados em um dispositivo. Esta reprodução dos dados é armazenada em local seguro fora do dispositivo de origem, protegendo-os contra perdas e danos. Segundo Ottaviani *et al.* (2011), pode-se classificá-los conforme o modelo de transferência de dados:

- Completo, onde todos os dados são transferidos;

- Incremental, onde apenas os dados que não estão no repositório são transferidos.

A cópia dos dados a ser realizada pode abordar dados armazenados em arquivos ou fisicamente guardados no disco rígido do computador, assim sendo, quando baseado em arquivos, torna-se mais lento, porém é mais fácil de gerenciar. Além disso, pode-se também transferir dados on-line e off-line, se for *off-line* é necessário conectar o dispositivo a uma mídia (HD, discos removíveis, CD, DVD, etc.). Já quando *on-line* o dispositivo deve estar conectado a uma rede e os dados podem ser armazenados remotamente (Armazenamento em Nuvem, E-mail, etc.).

O *backup* envolve vários fatores importantes para sua realização, como compressão dos dados, envio e armazenamento. É necessário que cada passo executado na sua realização possa ser trabalhado na forma invertida no momento da restauração, evitando assim a perda de informações na realização destas etapas.

A Restauração de dados é a recuperação da cópia de segurança dos dados, quando o armazenamento é realizado *off-line*, apenas faz-se a cópia da mídia novamente para o dispositivo. Já quando efetuado *on-line* é necessário que o dispositivo a receber os dados esteja conectado a uma rede e tenha acesso à conta onde os dados foram gravados.

2.3.1 Técnicas para Compressão de Dados

Como destaca Dapper (2013), a compressão de dados é utilizada na realização de transferência de dados, pois ao ser realizada diminui o tamanho das informações, permitindo assim que a transferência ocorra de forma rápida e que utilize menos espaço de armazenamento. A compressão de dados sem perda consiste em converter um conjunto de dados com um tamanho inferior.

Isto é possível através da redução da redundância presente na sequência de dados a ser comprimida. Existem três tipos de algoritmos deste tipo, são eles:

- Algoritmo de Comprimento – Busca de sequências de símbolos substituindo-as por códigos que indicam o número de vezes que aquele símbolo ocorreu. O algoritmo RLE (*Run-Length Encoding*) é o que mais se destaca neste tipo de aplicação.
- Algoritmo Estatístico – Verifica a redundância do conjunto de dados através da atribuição de código de tamanho dependente da frequência do símbolo, tendo os mais frequentes com um tamanho menor. O algoritmo de Huffman é o mais eficiente neste tipo de aplicação, com base em Knuth (1985), o algoritmo de Huffman consiste na construção de uma árvore binária em que todos os números à direita de qualquer nó são 1 e os à esquerda são 0. Assim, constrói-se esta árvore unindo-se os dois conjuntos de símbolos que se repete menos. Além disso, soma-se então, o peso dos dois e esse se torna um novo peso de repetição. Depois se une os próximos dois menores, ficando as sequências de símbolos que se repetem com uma maior freqüência, assim representa com um número menor de bits.
- Algoritmo baseado em dicionário – Segundo Nelson & Gailly (1996), neste modelo as sequências de símbolos são substituídas por códigos pertencentes a um dicionário. A aplicação criada por Jacop Ziv e Abraham Lembel, que utiliza dicionários adaptáveis. Sendo que o LZ77 destaca-se entre os algoritmos deste tipo, ele possui uma janela deslizante que possui uma buffer de busca em que ficam os dados a serem compactados e um dicionário dos símbolos que já estão codificados. Assim, no momento da compressão os dados ficam à direita na janela deslizante e conforme vão se repetindo são colocados na buffer com um código que possui a posição, o tamanho e o próximo símbolo da sequência, os dados já compactados vão sendo excluídos da janela deslizante, restando ao fim apenas o dicionário e o código.

Existe um padrão de algoritmo que utiliza o algoritmo LZ77 e aplica a árvore de Huffman nos campos do código gerado, além do mais é denominado

algoritmo Deflate que é encontrado nos arquivos do padrão ZIP e é amplamente utilizado devido a sua compatibilidade com a linguagem java.

2.3.2 Meios de Armazenamento

Existem diversos meios de armazenamento de dados, no caso de transferência *off-line,* faz-se o uso de uma mídia, ou seja, conecta-se ao aparelho no qual os dados estão depositados a um dispositivo de armazenamento como, por exemplo, *pen drives*, HD externo ou CD's. Após, é realizada a transferência dos dados para este dispositivo. Já no *backup on-line* o local mais utilizado atualmente é a computação em nuvem.

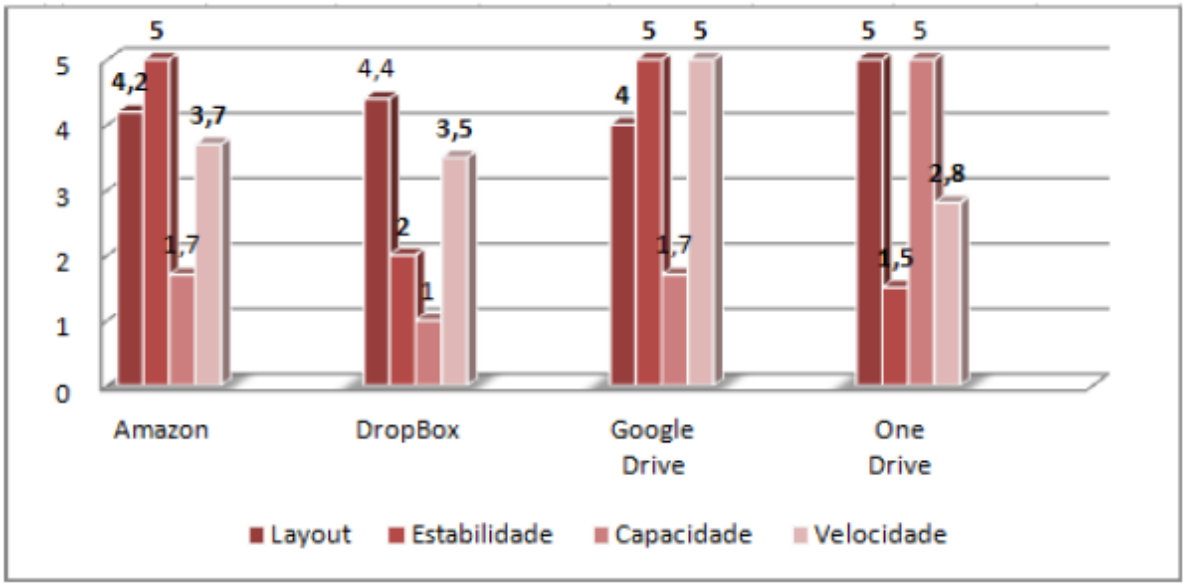
Conforme Ruchel *et al.* (2010), a computação em nuvem é baseada nos serviços e produtos de tecnologia da informação sob demanda (*Unility Computing*). Tem como princípio base o fornecimento de componentes básicos, como o armazenamento através de provedores especializados com um baixo custo unitário. A capacidade de armazenamento fornecida é ampla e é possível ler e gravar dados. Assim como em caso de falhas é de responsabilidade do provedor substituir componentes e tornar os dados disponíveis em tempo hábil.

A arquitetura da computação em nuvem divide-se em três camadas abstratas, a camada mais baixa é a infraestrutura, por meio desta disponibiliza-se os serviços de rede e armazenamento. A segunda camada é a de plataforma, ela fornece os serviços para que as aplicações possam ser desenvolvidas, testadas, implementadas e mantidas no ambiente de nuvem pelos prestadores de serviço. A camada do nível mais alto é de aplicação, ela disponibiliza diversas aplicações, como serviço para os usuários, em que são disponibilizados os serviços de armazenamento.

A pesquisa de Bortolin *et al.* (2014), investigou algumas das principais aplicações de armazenamento em nuvem, gratuitos, disponíveis no mercado (*Google Drive, Amazon Cloud Drive, DropBox* e *One Drive*) e qual tem o melhor desempenho. Esta avaliação foi feita através das seguintes variáveis, *layout,*

estabilidade, capacidade de armazenamento e velocidade. Como resultado da pesquisa os autores apontaram os softwares com melhor desempenho, como demonstra a figura 4.

Figura 4 - Desempenho das aplicações de armazenamento em nuvem.



Fonte: Bortolin *et al.* (2014)

Através desta analise verificou-se que o armazenamento em nuvem com o melhor desempenho é o Google Drive disponibilizado pela Google. Através dos percentuais agregados a cada um dos pontos da medida de desempenho, chamado Likert, desenvolvido por Bortolin *et al.* (2014), esta plataforma ficou com pontuação 4,2 e em segundo ficou a Amazon com 3,6

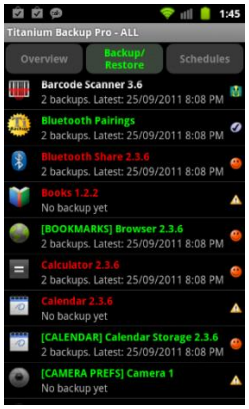
2.3.3 Soluções de *Backup*para Android

Existem muitos aplicativos para a cópia segura de dados, disponíveis na Google Play[3], todos tendem a mostrar uma plataforma simples e o armazenamento em nuvem, dentre eles os mais utilizados são:

[3] Loja virtual do Android em que estão todas as aplicações destinadas a esta plataforma.

- **Titanium:** Segundo Gaikar (2012), este aplicativo, desenvolvido para salvar aplicações, é um dos dez aplicativos essenciais do Android. É possível realizar o *backup*, restauração ou congelamento (na versão paga PRO) de aplicativos, com os dados e o link do mercado. Ele inclui também, proteção de aplicações e pode-se agendar a transferência, os dados são movidos para um cartão SD conectado ao aparelho e foi desenvolvido pela Titanium Track. Para utilizar esta aplicação é necessário realizar root do aparelho, para isso é necessário a instalação de outro aplicativo, ou seja, ele solicita então para ter acesso ao armazenamento. Enquanto ele faz a busca pelos aplicativos para *backup* é possível utilizar o aparelho normalmente, porém, não possui uma interface dinâmica, a figura 5 demonstra a interface do usuário desta aplicação.

Figura 5 - Titanium *backup*.



Fonte: Titanium *Backup* (2011)

- **CM *Backup***: Desenvolvido pela *Cheetah Mobile*[4], consiste em um armazenamento em nuvem, ou seja, no qual seleciona-se quais dados devem ser transferidos. Além disso, possui 5GB de capacidade de armazenamento e o *backup* pode ser agendado. Ao instalar o aparelho é necessário entrar com uma conta do Facebook, Google+ ou se registrar.

[4]https://www.cmcm.com/en-us/cm-backup/

Além do mais, oferece *backup* dos contatos, SMS, registro de chamadas, calendário, alarmes e favoritos. Ao realizar o *backup* são mostrados quantos itens foram transferidos e se desejar pode-se excluir itens do aparelho ou até mesmo no momento da restauração mostra a mesma lista, podendo-se restaurar apenas um tipo de item. A figura 6 mostra a tela inicial da aplicação que possui uma interface fácil e dinâmica.

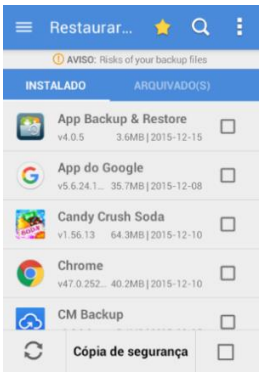
Figura 6 - CM *backup*.

Fonte: Android Headlines (2014)

- **App *Backup*& Restore**: Criado pela Infolife[5], seu fabricante destaca que realiza a cópia da apk. dos aplicativos instalados no dispositivo, podendo gravar estes dados no cartão SD ou enviar ao e-mail disponibilizado pelo usuário. Além de possuir uma interface simples é possível selecionar também quais aplicativos deseja-se realizar *backup*. As mensagens e avisos mostrados ao usuário estão em inglês. A figura 7 mostra como a aparência do programa.

[5]http://infolife-llc.android.informer.com/

Figura 7 - App *Backup*& Restore.



Fonte: Google Play.

Alguns aplicativos podem oferecer a opção de agendamento, isto é, é realizada a transferência na data fornecida pelo utilizador. As aplicações apresentadas diferem do modelo proposto nesta pesquisa, pois, o usuário não fica ciente do momento da transferência, além disso, esta não inicia no momento em que o aparelho tem pouco uso, a interface de alguns deles não é simples e é de difícil entendimento. Além disto, é oferecida uma lista para que o usuário escolha os itens que devem ser copiados, no Charge Backup o usuário seleciona os dados através do sistema de arquivos.

2.4 Trabalhos Correlatos

Algumas pesquisas que estão sendo realizadas por outros autores na área de segurança de informação relacionam-se com a desenvolvida nesta pesquisa, porém este trabalho possui uma nova perspectiva. Três trabalhos são destacados neste capítulo demonstrando a soma de outras pesquisas para esta área e vislumbrando os novos componentes a serem realizados.

A pesquisa de Shriwas *et al.* (2013), relata problemas enfrentados pelos usuários ao realizar *backup* e restauração, o autor expõe uma nova abordagem de *backup* e restauração de dados em Smartphone Android. Além do mais, como ele usa a técnica de compressão RLE, ele economiza tempo, espaço para armazenar e melhorar o desempenho. Ainda, destaca também novas

formas de trabalhar com compressão RLE, portanto, a diferença entre a pesquisa desenvolvida por Shriwas e a realizada neste trabalho, é que não utilizaremos a compressão RLE. Apesar de esta compressão ser muito eficiente em dados com alta repetição de padrões, como em imagens, ela se mostra pouco eficiente em dados que não possuem um padrão repetitivo, podendo até aumentar o tamanho do arquivo ao invés de diminuí-lo.

Já Barbera *et* al. (2013), demonstra em sua pesquisa as peculiaridades da computação em nuvem como uma associada aos sistemas móveis. Portanto, estuda-se o *software fmobile* / *backup seasibility* de dados de ambos *off loading* computação móvel e *software* móvel, *backup* de dados em situações da vida real. Assumindo uma arquitetura na qual cada dispositivo real está associado a um clone software na nuvem, além disso, é considerado dois tipos de clones: O *off-clone*, cujo objetivo é apoiar a computação *offloading*, e o *back-clone*, que vem a ser usado quando uma restauração de dados e aplicativos do usuário é necessária. Apesar de neste estudo utilizarmos a computação em nuvem para o armazenamento dos dados, não será utilizado um clone do software na nuvem.

Ottaviani *et al.* (2011), apresentam em sua pesquisa um novo método de *backup* e restauração de dados para dispositivos móveis, esta abordagem é independente de plataforma em particular. São apresentados dois protótipos baseados em dois diferentes sistemas operacionais móveis: Google Android e Symbian S60. Outra característica desta abordagem reside na capacidade de compartilhar informações em dispositivos móveis entre um grupo de pessoas selecionadas. Esta pesquisa difere-se da de Ottaviani *et al.* (2011), pois destaca apenas o *backup* no sistema operacional Android e não serão feitas abordagens a compartilhamento de dados.

Nenhuma das pesquisas correlatas citadas realiza um *backup* transparente, o qual viabilize ao usuário uma transferência fácil e eficaz. Ou seja, os estudos apresentados visam à utilização de *backups* com outros fatores, como o clone em nuvem, compartilhamento de informações e a compactação RLE, não focando na eficiência propriamente dita. Portanto,

nesta pesquisa o objetivo principal é a transparência e eficiência de *backup* destacando-se assim das relatadas anteriormente.

3 METODOLOGIA

Segundo Cervo *et al.* (2007), o método científico visa a realidade dos fatos, no qual se utiliza observação e a coleta de todos os dados possíveis, isto é, hipótese para explicar as observações, a experimentação, a indução e a teoria. Empregou-se neste projeto, portanto, o método científico, ou seja, utilizando as técnicas de observação sistemática, descrição das técnicas observadas e comparação dos diversos métodos de salvaguarda empregados atualmente. Desta forma, constituiu-se em uma pesquisa bibliográfica sobre o sistema operacional Android, verificando-se suas características e as técnicas de salvaguarda de informação, a fim de entender seu funcionamento, através de artigos científicos e livros, assim visou-se a realidade e não empregabilidade de suposições. Assim, foram analisadas as diversas técnicas de salvaguardar dados, abordadas em pesquisas e as mais utilizadas. Caracterizou-se também como uma pesquisa experimental, pois os dados bibliográficos pesquisados foram utilizados para a criação de um modelo de *backup* que foi implementado.

Analisou-se as pesquisas realizadas, por outros pesquisadores na área de segurança da informação, a fim de obter um parâmetro para realizar uma implementação segura e simples de transferência de dados**,** possibilitando que qualquer usuário possa utilizar a ferramenta. além do mais que a utlização ocorra de forma transparente e de forma automática, facilitando assim a realização do *backup*, estimulando os usuários a fazê-lo com mais frequência.

Figura 8 - Organograma das Etapas da Metodologia.

Fonte: Elaborada pelo autor, 2016.

A figura 8 apresenta o organograma com a representação de cada etapa que foi abordada nesta pesquisa, tendo se iniciado por um vasto estudo dos métodos e técnicas de salvaguarda de informações. Entre eles as técnicas de compactação de dados e tecnologias de armazenamento, que vem sendo abordado por outros pesquisadores. Além disto, testou-se as principais aplicações de backup gratuito, pode-se então fazer o uso dos pontos positivos, assim como evitando os negativos no modelo desenvolvido. Através desta avaliação foi proposto um método para a exportação de dados, verificando-se meios de armazenamento bem como um modelo de *backup*, assim este modelo foi então implementado e documentado. Ademais, executou-se uma pesquisa sobre teste de software e foi criado um roteiro de testes que foram realizados no programa. Por fim, realizou-se a validação da aplicação, instalando o sistema em outros dispositivos. Os resultados, portanto, foram então analisados e as conclusões elaboradas.

4 CHARGER BACKUP

Neste capitulo será abordado o software Charge Backup, iniciando com a explicação do modelo proposto para backup e o levantamento dos requisitos, até sua implementação e validação. Assim, na seção 4.1 é explicado o modelo e seu funcionamento. Já na seção 4.2 é demonstrado a análise e o projeto. Na seção 4.3 é verificado o desenvolvimento do software implementado. E finalmente na seção 4.4 são abordados os testes do sistema.

4.1 Proposta de um modelo para salvaguarda de dados

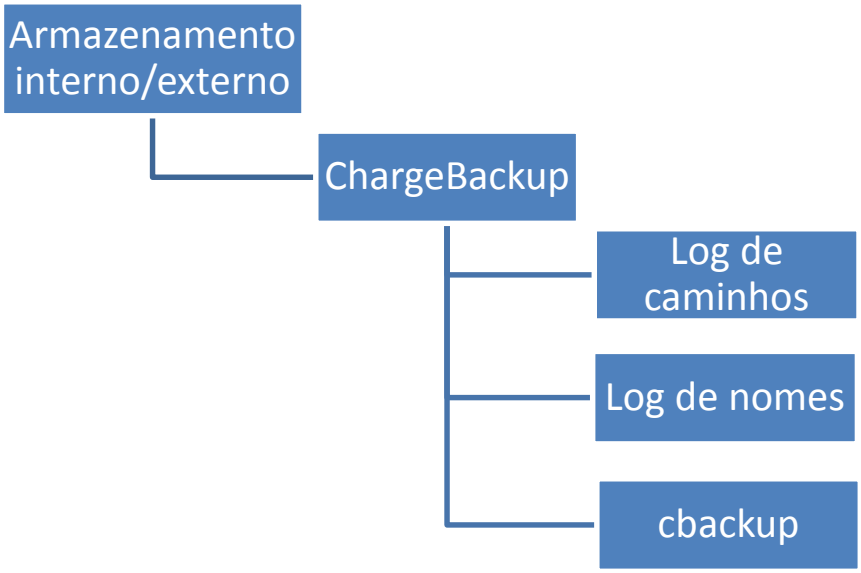
Entendendo a importância do processo de *backup*, realizou-se um estudo sobre as ações e funcionalidades necessárias a uma salvaguarda de dados efetiva, em dispositivos móveis com a plataforma Android. Para isto, o modelo propõe que a transferência deve ocorrer quando o aparelho entrar em momento de *charge,* ou seja, quando é colocado para carregar ou quando conectado via USB. Além disto, o backup deve ser transparente, possibilitando que o aparelho possa realizar outras tarefas enquanto a transferência de dados é realizada seja na instalação ou ao clicar no ícone do programa, assim o usuário poderá realizar quatro ações:

- Selecionar as pastas com os arquivos que deseja enviar para nuvem;
- Restaurar os dados;
- Realizar o *backup;*
- Definir o local de *restore* como armazenamento externo (Cartão de memória) ou armazenamento interno (memória do próprio dispositivo).

O modelo proposto define que no momento da instalação, aparecerá uma lista com todas as contas da Google vinculadas ao aparelho, o usuário

deverá escolher ou adicionar uma conta em que os dados irão ser salvos. Após a seleção do usuário, é criado um log com o caminho de cada arquivo contido nos diretórios selecionados. Este arquivo de texto é salvo em uma pasta chamada ChargeBackup que é criada pelo sistema. Assim, ao iniciar o *backup* o log de caminhos é lido e é realizada uma cópia dos dados contidos neles dentro da pasta cbackup, esta pasta é então compactada em um arquivo com extensão .zip que utiliza a compactação Deflate discutida na seção 2.3.1. O diretório cbackup é criado dentro da pasta ChargeBackup, que fica no armazenamento escolhido (interno ou externo). Após o envio, o diretório cbackup é apagado automaticamente. A figura 9 mostra a árvore de armazenamento criada pelo sistema Charge Backup dentro do diretório selecionado pelo usuário.

Figura 9 - Arvore de Armazenamento



Fonte: Elaborada pelo autor, 2016.

Um arquivo de texto com o nome das pastas selecionadas também é criado e é armazenado na pasta ChargeBackup. Portanto, tem como objetivo demonstrar ao usuário quais pastas já foram escolhidas, assim, ao abrir o

programa é feita uma leitura deste log e é mostrado na tela da aplicação. Desta forma, o usuário pode selecionar mais dados ou apagá-los, pois ele poderá verificar a todo momento quais as informações estão sendo salvas. A figura 10 demonstra o funcionamento do modelo desenvolvido.

Figura 10 - Modelo da Aplicação.

Fonte: Elaborada pelo autor, 2016.

Através da figura 10, pode-se observar que ao conectar o dispositivo móvel a uma fonte de energia (momento de *charge*) o sistema gera uma mensagem *broadcast*. A aplicação recebe esta mensagem e inicia um *service* que é uma classe padrão android utilizada para realizar o *backup* em *background*, ou seja, para realizar a transferência de modo transparente. Assim, o usuário pode usar seu dispositivo normalmente enquanto a transferência é realizada. Após o primeiro envio, é aguardado 5 momentos de *charge* para realizar o próximo.

Para realizar a restauração, a aplicação acessa a conta Google Drive, fornecida pelo usuário e, então busca os arquivos cbackup.zip. É mostrada uma lista, ao usuário, com todas as versões realizadas até o momento, ou seja,

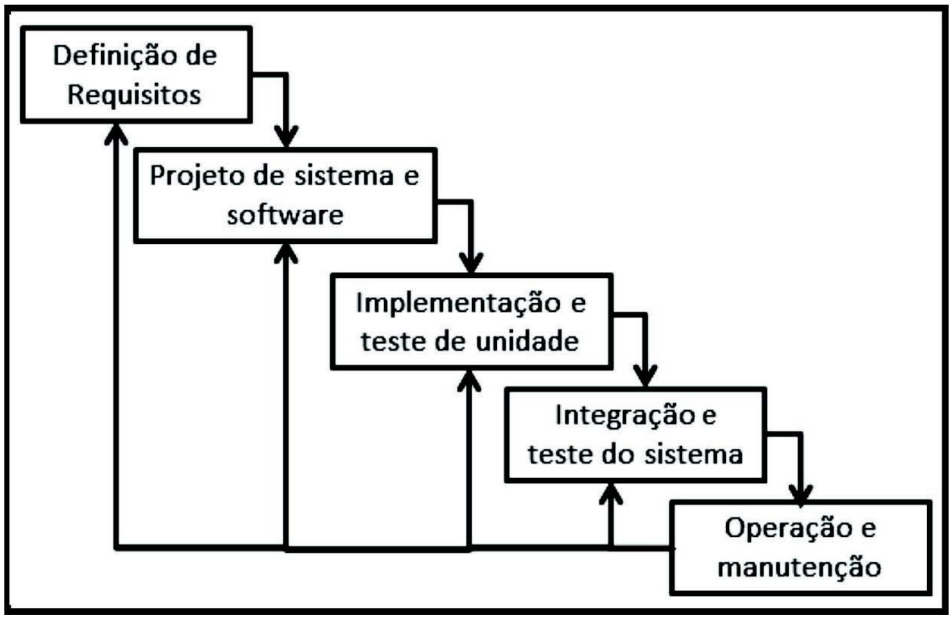
com suas datas de modificação. Assim sendo, pode-se então escolher a versão desejada, o arquivo selecionado é salvo na pasta ChargeBackup e, então é descompactado.

O fato de o programa iniciar no momento de carga da bateria, é favorável, por não serem realizadas muitas tarefas neste momento, mesmo a transferência sendo realizada em *background* pode acabar deixando o aparelho lento. Desta maneira, o momento de *charge* facilita assim que os dados sejam frequentemente salvos, em uma plataforma fora do aparelho, deixando-os seguros.

4.2 Análise e Projeto

Neste projeto foi utilizado o modelo de desenvolvimento em cascata, Lessa *et al.* (2009), destaca que este é um modelo linear e sequencial, isto é, cada fase do desenvolvimento segue uma ordem, sem qualquer sobreposição ou interação. Este modelo foi escolhido já que tem as suas fases de desenvolvimento bem delimitadas, assim foi possível criar um planejamento, bem como definir os prazos para a finalização de cada fase. A figura 11 mostra as fases deste modelo.

Figura 11 - Modelo Cascata.



Fonte: Elaborada pelo autor, 2016.

As seções que seguem demonstram as fases deste desenvolvimento, desse modo na seção 4.2.1 serão definidos os requisitos do sistema. Já a seção 4.2.2 mostrará o projeto de sistema de software. Logo, a implementação por ser mais extensa será abordada na seção 4.3 e os testes assim como a manutenção serão descritos na seção 4.4.

4.2.1 Definição de Requisitos

Os requisitos de software como descreve Paula Filho (2009), são as características que definem os critérios de aceitação de um produto. Consequentemente, quando a documentação é bem elaborada, os requisitos documentados têm maior chance de ser corretamente entendidos, logo, para o levantamento dos requisitos é necessário realizar uma análise para verificar quais as especificações que o programa deve ter, seguindo assim uma linha lógica e objetiva de pensamento.

Os requisitos funcionais definem as funcionalidades e serviços do sistema, portanto, o objetivo principal do aplicativo é permitir o *backup* de dados no momento de *charge* (momento de carregamento da bateria do aparelho). Para isto, o sistema deve ser capaz de executar as seguintes funcionalidades:

- Selecionar dados: Deve permitir que o usuário selecione os dados para *backup*;
- Apagar dados: O Usuário deve ser capaz de apagar os dados selecionados;
- Escolher conta: A aplicação deve permitir que o usuário escolha a conta do Google Drive;
- Acesso de dados: A aplicação deve possibilitar o acesso aos dados escolhidos pelo usuário;
- Copiar dados: A aplicação deve ser capaz de copiar os dados dentro de uma pasta no dispositivo;
- Compactar os dados: Deve ser capaz de compactar os dados;
- Enviar os dados: Deve ser capaz de enviar os dados para a plataforma em nuvem Google Drive;
- Realizar restauração: Deve ser capaz de buscar o arquivo compactado no Google Drive;
- Descompactar: A aplicação deve ser capaz de descompactar o arquivo restaurado.

Ao contrário dos requisitos funcionais, os requisitos não funcionais não expressam nenhuma função a ser realizada pelo software. Ao invés disto, expressam comportamentos e restrições que este software deve satisfazer, como destaca Cysneiro *et al.* (1997), assim os requisitos não funcionais da aplicação são os seguintes:

- Sistema Operacional: O dispositivo deve ter o sistema operacional Android com versão superior a 4.4 Kitkat;
- Acesso à *Internet*: O dispositivo deve ter acesso à *internet*;
- Possuir Google Play Service instalada no dispositivo;
- Desempenho: Deve apresentar um bom desempenho;
- Simplicidade: Deve possuir uma interface de fácil utilização;
- Transparência: Deve realizar o *backup* de forma transparente, ou seja, que não atrapalhe a utilização do dispositivo pelo usuário.

4.2.2 Projeto do Sistema e Software

Com a finalidade de organizar a implementação da aplicação para o *backup* de dados em dispositivos móveis, foi adotado um padrão de modelagem UML (*Unified Modeling Language*), com o intuito de auxiliar na visualização do sistema a ser implementado. Segundo De Almeida *et al.*(2011), a UML é uma técnica que apresenta diagramas que auxiliam o processo de desenvolvimento de *software*.

Para a geração dos diagramas, foi utilizada a ferramenta *Astah Community*[6]*.* Neste capítulo, portanto, é apresentada a análise do projeto. Para descrever o sistema serão utilizados os seguintes diagramas UML:

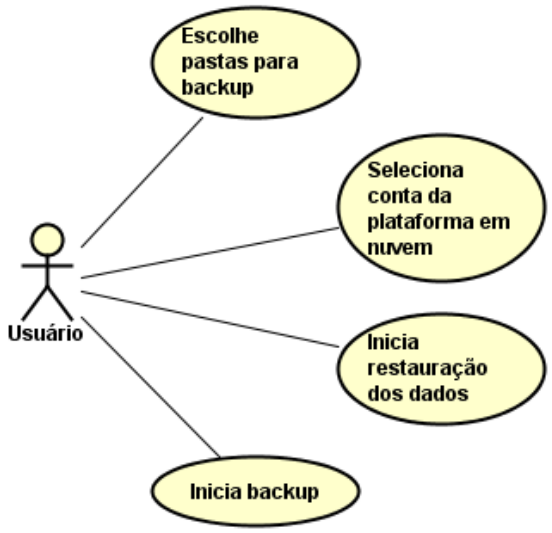
- Casos de uso: que são utilizados para a análise dos requisitos do sistema, com o detalhamento do comportamento em alto nível, ou seja, o sistema deve executar para um determinado autor.
- Diagramas de sequência: Descreve a ordem temporal em que as mensagens são trocadas entre os objetos (FONSECA, 2015).

Segundo Sommerville (2011), os diagramas de casos de uso, utilizam técnicas baseadas em cenários para a aquisição de requisitos do sistema.

[6]http://astah.net/editions/community

Representando todas as ações que foram descritas nos requisitos funcionais do sistema, desta maneira, a figura 12 representa os casos de uso do usuário.

Figura 12 - Diagrama de Casos de Uso do Usuário.



Fonte: Elaborada pelo autor, 2016.

A partir do diagrama exibido na figura 12, foram criadas tabelas com a descrição dos casos de uso. Nestas tabelas é apresentado o ator principal, isto é, a descrição, a pré-condição e a pós-condições.

O caso de uso Pasta para *backup* possibilita ao usurário escolher as pastas que deseja que sejam transferidas.

Tabela 1 – Caso de Uso Pasta para *Backup*

Nome do Caso de Uso	**Pasta para *backup***
Ator Principal	Usuário
Descrição	Usuário escolhe pastas para *backup*
Pré-Condições	
Pós-Condições	É criado um log com os caminhos dos

	arquivos

Fonte: Elaborado pelo autor, 2016.

O caso de uso Seleciona Conta, possibilita que o usuário selecione a conta da Google Drive desejada, este caso de uso disponibiliza uma lista com as contas já vinculadas ao aparelho, porém pode-se adicionar outra.

Tabela 2 – Caso de Uso Seleciona Conta

Nome do Caso de Uso	**Pasta para *backup***
Ator Principal	Usuário
Descrição	Usuário escolhe conta da Google Drive para a realização do *backup*.
Pré-Condições	O usuário deve possuir uma conta da Google.
Pós-Condições	

Fonte: Elaborado pelo autor, 2016.

O caso de uso Restauração de Dados possibilita ao usuário iniciar o processo de restauração, para isso, ele deve ter realizado o *backup* pelo menos uma vez.

Tabela 3 - Caso de Uso Restauração de Dados

Nome do Caso de Uso	**Pasta para *backup***
Ator Principal	Usuário
Descrição	Inicia a realização da restauração.
Pré-Condições	O usuário deve ter escolhido a conta

	Google Drive; O usuário deve ter realizado o *backup* pelo menos uma vez.
Pós-Condições	O usuário deve selecionar a versão do arquivo cbackup que deseja restaurar; O arquivo.zip é salvo na pasta ChargeBackup; O arquivo é descompactado.

Fonte: Elaborado pelo autor, 2016.

O caso de uso Iniciar *Backup* possibilita ao usuário iniciar o *backup,* mesmo sem o dispositivo estar em momento de carga. Para isso é necessário que se tenha selecionado uma conta na Google Drive e as pastas para *backup*.

Tabela 4 - Caso de Uso Iniciar *Backup.*

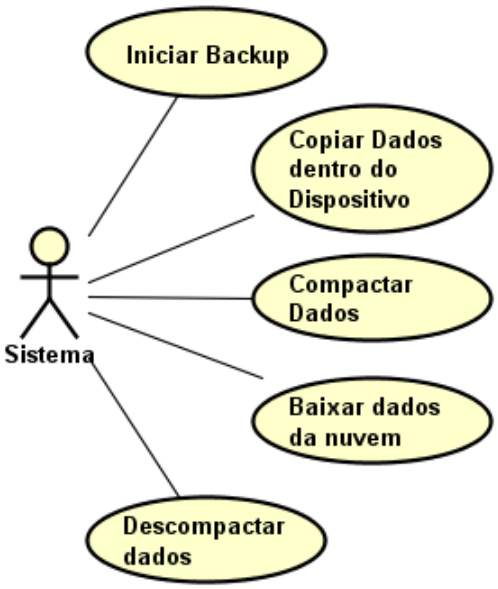
Nome do Caso de Uso	**Pasta para *backup***
Ator Principal	Usuário
Descrição	Inicia o *backup* de dados.
Pré-Condições	O usuário deve ter selecionado uma conta da Google Drive; O usuário deve ter selecionado as pastas para *backup*.
Pós-Condições	Os arquivos selecionados são copiados para a pasta cbackup dentro do diretório ChargeBackup; A pasta cbackup é compactada para

	um arquivo cbackup.zip; Este arquivo é enviado para a Google Drive.

Fonte: Elaborado pelo autor, 2016.

A figura 12 demonstrou o diagrama de casos de uso em que o usuário é o autor principal, na figura 13 é possível observar o diagrama de casos de uso no qual o sistema é o autor principal das atividades. Definindo, assim, as atividades que são realizadas automaticamente pelo sistema, não necessitando da interferência do usuário da aplicação.

Figura 13 - Diagrama de Casos de Uso do Sistema.



Fonte: Elaborado pelo Autor, 2016.

Através do diagrama mostrado na figura 13, podem-se montar as seguintes tabelas dos casos de uso:

O caso de uso Iniciar *Backup* assemelha-se ao caso de uso Iniciar *Backup* do usuário, diferencia-se apenas pela forma de iniciar a transferência,

sendo iniciado pelo sistema ao receber uma mensagem *broadcast* que informa que o dispositivo móvel entrou em momento de *charge.*

Tabela 5 - Caso de Uso Iniciar *Backup.*

Nome do Caso de Uso	**Pasta para *backup***
Ator Principal	Sistema
Descrição	Inicia o *backup* de dados.
Pré-Condições	O usuário deve ter selecionado uma conta da Google Drive; O usuário deve ter selecionado as pastas para *backup*; A variável “n” que conta os momentos de *charge* deve estar com o valor 0; O dispositivo deve estar em momento de *charge*.
Pós-Condições	Os arquivos selecionados são copiados para a pasta cbackup dentro do diretório ChargeBackup; A pasta *charge* cbackup é compactada para um arquivo cbackup.zip; Este arquivo é enviado para a Google Drive. A variável “n” é incrementada.

Fonte: Elaborado pelo Autor, 2016.

O caso de uso, cópia dados, dentro do dispositivo, acessa os dados que devem ser copiados através do log, gerado no momento em que são escolhidas as pastas.

Tabela 6 - Copia Dados Dentro do Aparelho.

Nome do Caso de Uso	**Pasta para *backup***
Ator Principal	Sistema
Descrição	Copia dados dentro do aparelho.
Pré-Condições	O usuário deve ter escolhido as pastas para *backup.*
Pós-Condições	A pasta onde os arquivos são salvos é compacta.

Fonte: Elaborado pelo Autor, 2016.

O caso de uso compactar dados é realizado automaticamente pelo sistema, ou seja, ele compacta a pasta cbackup em que encontra-se a cópia dos arquivos selecionados pelo usuário.

Tabela 7 - Caso de Uso Compactar Dados.

Nome do Caso de Uso	**Pasta para *backup***
Ator Principal	Sistema
Descrição	Compacta dados
Pré-Condições	As cópias dos arquivos selecionados devem estar na pasta cbackup.
Pós-Condições	O arquivo cbackup.zip é enviado para

	a nuvem.

Fonte: Elaborado pelo Autor, 2016.

O caso de uso baixar dados da nuvem é realizado após o usuário escolher a versão do cbackup.zip que deseja restaurar e salva este arquivo na pasta ChargeBackup.

Tabela 8- Baixar Dados da Nuvem.

Nome do Caso de Uso	**Pasta para *backup***
Ator Principal	Sistema
Descrição	Baixar dados da nuvem.
Pré-Condições	O usuário deve ter escolhido a versão que deseja realizar restauração.
Pós-Condições	O arquivo cbackup.zip é salvo na pasta ChargeBackup.

Fonte: Elaborado pelo Autor, 2016.

O caso de uso descompactar dados, descompacta o arquivo cbackup.zip salvo na pasta ChargeBackup.

Tabela 9 - Caso de Uso Descompactar

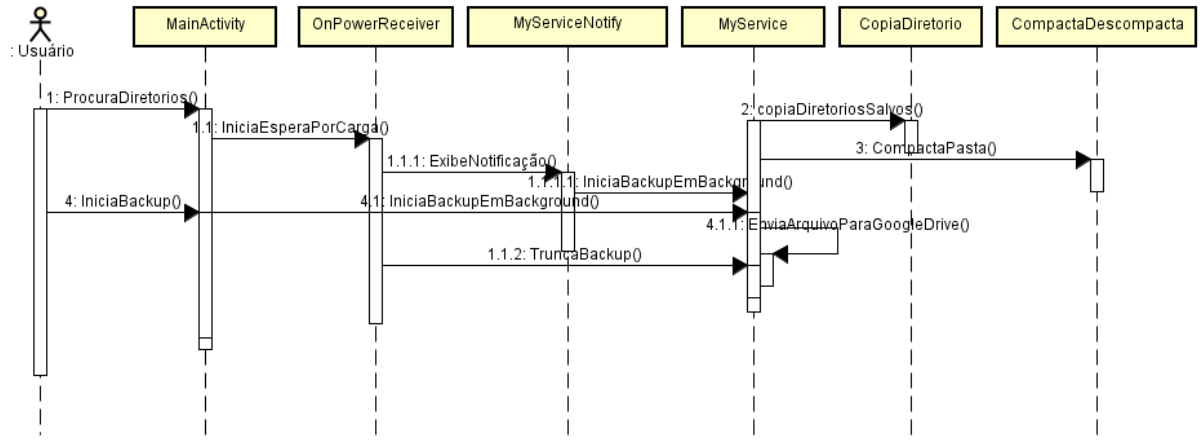
Nome do Caso de Uso	**Pasta para *backup***
Ator Principal	Sistema
Descrição	Descompacta dados.

Pré-Condições	O arquivo cbackup.zip deve ter sido restaurado da nuvem e salvo na pasta ChargeBackup.
Pós-Condições	

Fonte: Elaborado pelo Autor, 2016.

Após analisar cada caso de uso da aplicação é necessário definir a interação entre as classes para a execução destes casos de uso, para isto, foram criados diagramas de sequência. Fowler *et al.* (2005), destaca que o diagrama de sequência é um tipo de diagrama de interação que demonstra como um grupo de objetos colaboram em algum comportamento. Isto é, captura o comportamento em um único cenário, mostrando vários objetos e mensagens trocadas entre eles. Na figura 14 é demonstrado o diagrama de sequência para a ação de *backup*, destacando a interação entre as classes do sistema.

Figura 14 - Diagrama de Sequência do *Backup*.



Fonte: Elaborada pelo autor, 2016.

As interações mostradas na figura 14 destacam troca de informações que cada classe realiza no momento de backup. Inicialmente o usuário deve selecionar as pastas, isto ocorre na classe principal MainActivity, logo após, é

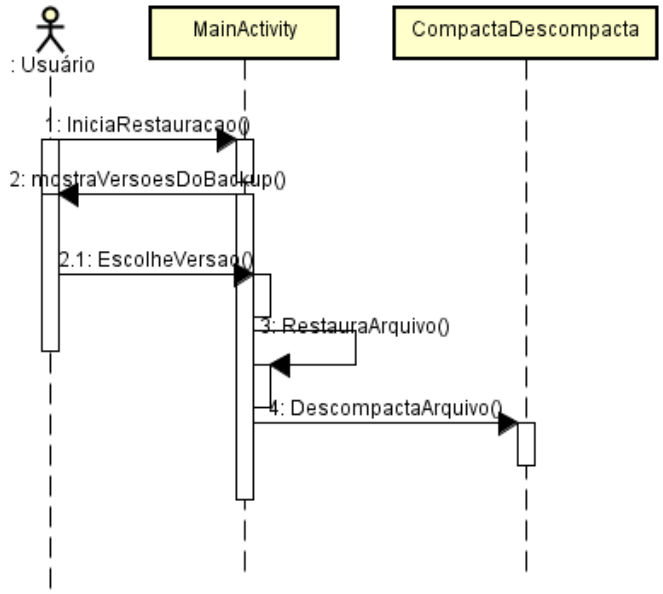
iniciada a classe OnPowerReceive que é uma classe BroadcastReceiver, assim ela aguarda por uma mensagem *broadcast* informando que o aparelho está em momento de charge. Assim sendo, quando isto ocorre, a classe verifica sua variável “n” para ver se ela é igual a 0, esta variável é incrementada a cada momento de charge, até 5 e depois é zerada novamente, desta forma, o *backup* é realizado a cada cinco momentos de *charge*.

Se “n” for igual a 0 e estiver em momento de *charge,* é enviada uma solicitação a classe MyServiceNotify, esta classe cria uma notificação que é mostrada ao usuário e ele pode decidir se a transferência de dados deve ser realizada neste momento ou não. Caso seja decidido que não, será solicitado novamente ao usuário no próximo momento de *charge*, já se a resposta for sim, inicia-se o processo de *backup* chamando a classe MyService.

A classe MyService pode ser iniciada por MyServiceNotify ou pela classe MainActivity, isto é, quando solicitado pelo usuário. Essa classe inicia a cópia dos dados dentro do aparelho através da classe CopiaDiretorio. A compactação da pasta cbackup que contém os dados copiados, também é iniciada por essa classe, chamando a classe CompactaDescompacta. Desta maneira, inicia-se então a transferência para a Google Drive, assim, a classe MyService pode ser interrompida a qualquer momento pela classe OnPowerReceiver, no momento em que o aparelho é desconectado da energia. Caso haja interrupção no momento da transferência, o programa desconecta o cliente Google. Portanto, a própria Google Drive Api é responsável por salvar localmente, os dados que não foram enviados, assim como enviar no próximo momento de *charge*, esta interrupção é chamada de truncamento do envio.

Através da figura 14 pode-se observar a ação de *backup* realizada pelo sistema. A figura 15 mostra o diagrama de sequência para a ação de restauração.

Figura 15 - Diagrama de Sequência da Restauração



Fonte: Elaborada pelo autor, 2016.

A figura 15 evidencia que ao realizar a restauração dos dados, apenas duas classes são utilizadas. Assim, no momento em que o usuário solicita o início do *backup* a classe MainActivity acessa a conta Google Drive e mostra uma lista de todas as versões de *backup* realizadas até o momento. Assim sendo, o usuário poderá escolher uma, ou seja, será então realizado um *download* deste arquivo, além do mais o arquivo será salvo na pasta ChargeBackup e então é solicitado a classe CompactaDescompacta que descompacte o arquivo.

4.3 Implementação

A partir da modelagem do sistema, foi necessário definir as ferramentas para a implementação do software, na programação para Android pode-se utilizar diferentes IDE (ambiente de desenvolvimento integrado). Assim sendo para o desenvolvimento desta aplicação optou-se pelo uso do Android Studio, pois é a ferramenta oficial da Google e possui o pacote SDK, que é um pacote para desenvolver *software*. Além do mais, para a implementação deste projeto, foi elencado a linguagem Java, visto que esta é a linguagem utilizada para o

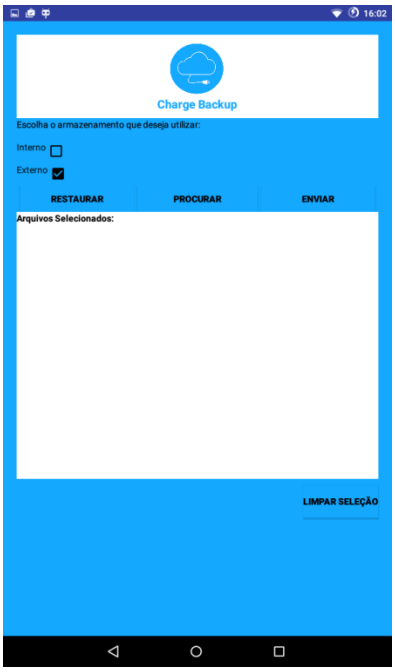
desenvolvimento sobre a plataforma, isto é, a solução utiliza classes e bibliotecas padrão do Android.

A fim de facilitar a compreensão deste projeto, o presente capítulo foi subdivido em seções que seguem a ordem de funcionamento do sistema. Sendo assim, na seção 4.3.1 é mostrada a tela inicial da aplicação e suas funções, na seção 4.3.2 é abordada a classe *DocumentsProvider* que é utilizada para a escolha de diretórios, já na seção 4.3.3 é destacada a cópia dos arquivos dentro do próprio dispositivo. Ainda, na seção 4.3.4 é verificado como foi realizada a compactação e descompactação dos dados. Consequentemente, na seção 4.3.5 é mostrada a classe *BroadcastReceiver*, na seção 4.3.6 é mostrada a Classe *Service*. Além disso, na seção 4.3.7 é demonstrado a Google Drive API Android, que é responsável pelo envio e busca do arquivo na nuvem. Finalmente, na seção 4.3.8 são verificadas as permissões necessárias para a utilização do sistema.

4.3.1 Tela Inicial da Aplicação

Ao criar uma atividade no Android Studio é gerado também um arquivo activity_main que é desenvolvido em XML, este arquivo é responsável pela tela da aplicação principal. Ao iniciar a implementação foi modificado esse arquivo e criada a tela mostrada na figura 16. Logo, Lima Junior *et al.* (2016), destaca que a interface de uma aplicação deve ser fácil de usar e entender, possibilitando que o usuário possa realizar suas tarefas de forma intuitiva. Assim, a interface desta aplicação foi moldada seguindo este pensamento.

Figura 16- Tela da Aplicação



Fonte: Elaborado pelo Autor, 2016.

A tela inicial da aplicação, que é mostrada na figura 16, apresenta as seguintes ações que podem ser realizadas pelo usuário:

- **Escolher Armazenamento:** Utilizou-se dois elementos checkedTextView. Os quais permitem que o usuário escolha em qual armazenamento (Interno ou Externo), os arquivos de texto, de configuração do sistema e os arquivos gerados na restauração e *backup* serão salvos;
- **Restaurar:** Ao clicar neste botão é realizada a restauração dos dados automaticamente, para isso o programa busca na conta selecionada as versões do arquivo cbackup.zip. Após, o usuário selecionar qual versão deseja, o programa restaura este arquivo na pasta ChargeBackup que é criada pela aplicação no armazenamento escolhido e o descompacta;
- **Procurar:** Neste botão é realizada a procura pelos arquivos que o usuário deseja salvar, na forma de uma cópia na nuvem. Para a realização desta procura foi implementada a classe *DocumentsProvider* que é usada para mostrar o sistema de pastas do aparelho. Portanto, a seleção é realizada

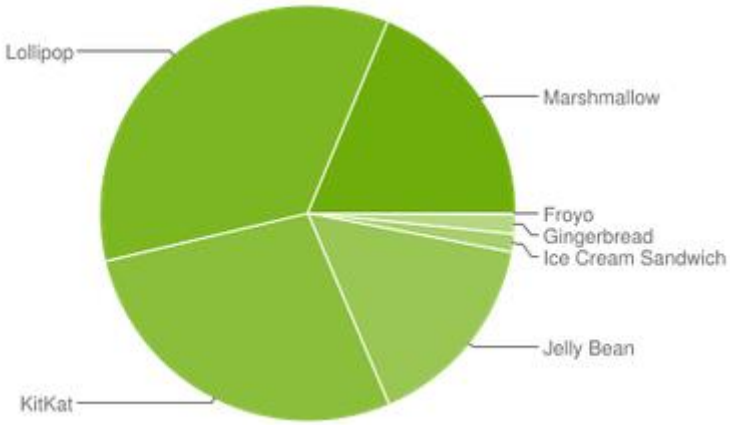
abrindo-se a pasta desejada e apertando o botão selecionar na parte inferior da tela;

- **Enviar:** O usuário poderá também realizar o *backup* quando desejar, ou seja, não precisando aguardar o momento de *charge*. Para isso foi disponibilizado no *layout* da aplicação um botão enviar que realiza a busca dos arquivos, compacta-os e os envia para a conta Google Drive, isto é, disponibilizada pelo usuário;
- **Limpar Seleção:** Utilizado para apagar a seleção de arquivos, ao clicar neste botão é necessário realizar a procura por pastas novamente.

4.3.2 D*ocumentsProvider*

No momento em que o usuário clica no botão procura, é demonstrado a ele o sistema de arquivos do aparelho, desta forma, ele poderá escolher quantos arquivos achar necessário para a realização do *backup.* Para implementar esta demonstração e escolha foi utilizada a classe *DocumentsProvider*, esta classe é aceita em sistemas Android com a versão a partir da 4.4 Kitkat. Isto é, limitando o programa os dispositivos com esta versão ou com versões mais recentes. Porém, segundo o site oficial de desenvolvimento para Android da Google -1(2016), 81,4% dos dispositivos como sistema operacional Android utilizados atualmente, possuem as versões Kitkat, Lollipop e Marshmallow. Portanto, esta classe limita a aplicação a apenas 18,6% dos aparelhos. À vista disso, a figura 17 mostra o gráfico da utilização das versões do Android.

Figura 17 - Utilização das Versões do Android.



Fonte: https://developer.android.com/about/dashboards/index.html

Apesar da limitação, é favorável utilizar esta classe, pois ela apresenta os arquivos do sistema de uma forma que o usuário é acostumado a visualizar e percorrer. Além disto, ela facilita a implementação, utilizando apenas um método para realizar a demonstração, este método foi chamado de performFileSearch e é mostrado na figura 18.

Figura 18 - Implementação do Método performFileSearch().

```
public void performFileSearch() {

    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT_TREE);

    startActivityForResult(intent,0);
}
```

Fonte: Elaborado pelo Autor, 2016.

Através da figura 18 pode-se observar que por meio do método foi criada uma *Intent*. Conforme Lancheta (2013), a *Intent* é utilizada através da classe padrão android. content. Intent e representa uma mensagem da aplicação para o sistema operacional, solicitando que algo seja realizado. Portanto, neste método ela solicita a demonstração do sistema de arquivos através da ACTION_OPEN_DOCUMENT_TREE. Após, o usuário fazer a escolha, deve ser realizada uma ação, para isto, utiliza-se startActivityForResult. Assim, ela

inicia a onActivityResult, este método gera um inteiro RESULT_OK, quando não ocorreu nenhum erro na execução da atividade, é necessário enviar outro inteiro. Neste caso é “0” que destaca que ação deve ser tomada, pois o onActivityResult é chamado por outros métodos. A figura 19 demonstra as ações tomadas no método onActivityResult para a seleção de arquivos.

Figura 19 – **onActivityResut para** a escolha de diretórios.

```
case (0):

    final TextView ver = (TextView)findViewById(R.id.verArquivo);

    if (resultCode == RESULT_OK) {

        Uri treeUri = data.getData();
        String path = FileUtil.getFullPathFromTreeUri(treeUri,this);


        String nome = pickedDir.getName();

        ver.setText(ver.getText() + nome + "\n");
        File file2=new File(path);
        diretorio(file2,path);

        try {
            CriaTxtPath();//CRIA TXT PARA ArqPath
            CriaTxtNome();//
        } catch (IOException e) {
            e.printStackTrace();
        }
```

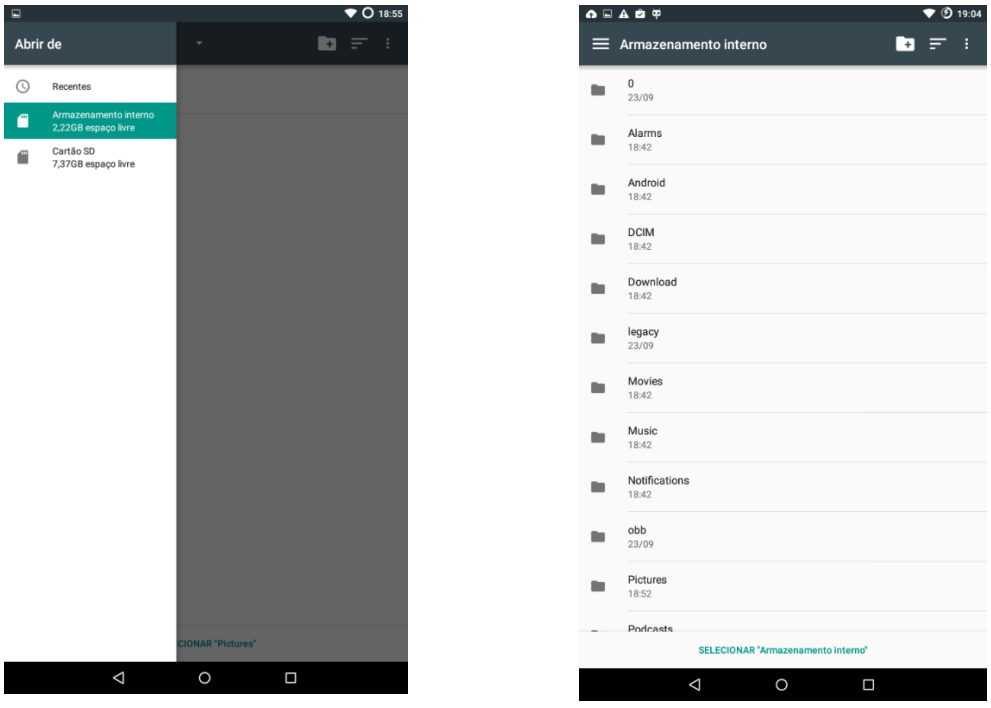
Fonte: Elaborado pelo autor, 2016.

A figura 19 demonstra que é gerado um Uri (Identificador de recurso uniforme), ou seja, para o documento escolhido pelo usuário, através deste identificador é possível acessar o caminho das pastas selecionadas. Assim sendo, criou-se uma classe FileUtil que gera o verdadeiro caminho do arquivo, pois ao usar get Path (treeUri) que é normalmente usado para descobrir um diretório, o caminho retornado é o caminho da árvore de arquivos e não o real. Portanto, no momento de realizar a cópia dos arquivos é necessário o caminho real. O método diretórios é utilizado para definir o caminho de todos os arquivos presentes nas pastas selecionadas, este método salva estes arquivos

em um vetor global. Além do mais, cria-se um arquivo de texto de caminhos que recebe os valores deste vetor e será utilizado para verificar quais dados devem ser transferidos no *backup.* Isto é, é criado também um log com os nomes das pastas selecionadas que será mostrado na tela da aplicação através do TextView, desta forma o usuário fica ciente de quais arquivos já foram selecionados.

A figura 20 mostra como o *DocumentsProvider* é apresentado ao usuário no momento em que é clicado no botão Procurar.

Figura 20- Seleção de pastas.



(a) (b)

Fonte: Elaborado pelo autor, 2016.

A figura 20 (a) é a primeira tela apresentada na qual estão às opções de Armazenamento interno, Armazenamento externo (Cartão SD) e recentes. E a figura 20 (b), mostra o armazenamento interno do dispositivo e o botão de selecionar arquivo na parte inferior da tela.

4.3.3 Cópia de Arquivos

No momento em que é iniciado o *backup* é realizada a cópia de todos os arquivos contidos no log de caminhos. Estes arquivos são salvos dentro da pasta cbackup que fica localizada dentro da pasta ChargeBackup, ou seja, no armazenamento escolhido. Assim sendo, para realizar esta cópia foi criada uma classe CopiaDiretorio, logo, a figura 21 mostra essa classe.

Figura 21 - Classe CopiaDiretorio.

```
public class CopiaDiretorio {

    public static void copyFileUsingStream(File source, File dest) throws IOException {
        //InputStream is = null;
        OutputStream os = new FileOutputStream(dest);
        try {
            InputStream is = new FileInputStream(source);
            //os = new FileOutputStream(dest);
            int tamanho= 4096;
            byte[] buffer = new byte[tamanho];
            int length=-1;
            while ((length = is.read(buffer, 0, tamanho)) != -1) {
                os.write(buffer, 0, length);
                os.flush();
            }
            assert is != null;
            is.close();
            os.close();
        }catch (IOException e) {
            Log.e(TAG, e.getMessage());
        }
    }
}
```

Fonte: Elaborado pelo autor, 2016.

Na figura 21 pode-se observar que ao chamar o método CopyFileUsingStream é necessário passar dois arquivos, um é o arquivo que deve ser copiado chamado de “source” e o outro é o destino chamado de “dest”. Assim, é criado então um *OutputStream* para escrever o “dest” e um *Input Stream* para ler “source”. Desta forma, feito um *while* que percorre os bytes do *Input Stream* e os lê. Esta leitura é salva no *OutputStream*, após

terminar, ambos são fechados. Esse processo é realizado para cada arquivo contido nas pastas selecionadas pelo usuário.

4.3.4 Compactação e Descompactação

Após a cópia ser realizada é necessário compactar os dados contidos na pasta cbackup. Como destacado na seção 2.3.1 a compressão Deflate que é a utilizada em arquivos com extensão .zip, é compatível com a linguagem Java, possuindo uma biblioteca para sua utilização. Como esta é a compactação mais indicada e facilita a implementação foi a utilizada neste trabalho. A figura 22 mostra como foi implementado método addToZip, que compacta arquivos.

Figura 22 - Compactar.

```
public static void addToZip(File directoryToZip, File file, ZipOutputStream zos) throws FileNotFoundException,
        IOException {

    FileInputStream fis = new FileInputStream(file);

    String zipFilePath = file.getCanonicalPath().substring(directoryToZip.getCanonicalPath().length() + 1,
            file.getCanonicalPath().length());
    System.out.println("Writing '" + zipFilePath + "' to zip file");
    ZipEntry zipEntry = new ZipEntry(zipFilePath);
    zos.putNextEntry(zipEntry);

    byte[] bytes = new byte[1024];
    int length;
    while ((length = fis.read(bytes)) >= 0) {
        zos.write(bytes, 0, length);
    }

    zos.closeEntry();
    fis.close();
}
```

Fonte: Elaborado pelo autor, 2016.

Foi criado um *FileInputStream* para a leitura do arquivo não compactado. Um *ZipOutputStream* é recebido como entrada deste método, este arquivo salva a leitura do*FileInputStream*. O outro parâmetro directoryToZip é o arquivo final .zip através dele é criado o ZipEntry, cada arquivo novo compactado é adicionado a esse arquivo através da linha zos.putNextEntry. Assim, o

FileInputStream é compactado no *ZipOutputStream* e este é agregado ao arquivo final directoryToZip.

Para descompactar os arquivos foi criado um método chamado de unZip. Este método é mostrado na figura 23.

Figura 23 - Descompactar.

```
public static void unZip(File file) {
    int tamanho=4096;
    byte[] buffer = new byte[4096];

    try {

        // Cria o input do arquivo ZIP
        ZipInputStream zinstream = new ZipInputStream(new FileInputStream(file));

        ZipEntry zentry=zinstream.getNextEntry();
        System.out.println("Estraindo...: " + file.getName());
        System.out.println("em: " + file.getAbsolutePath());

        while (zentry != null) {
            // Pega o nome da entrada
            String entryName = zentry.getName();
            System.out.println("File: " + entryName);
            FileOutputStream outstream = new FileOutputStream(new File(file.getAbsolutePath().replace(file.getName(), entryName)));
            int n=-1;
            while ((n = zinstream.read(buffer, 0, tamanho)) != -1) {
                outstream.write(buffer, 0, n);
            }
            outstream.close();

            zinstream.closeEntry();
            zentry = zinstream.getNextEntry();
        }
```

Fonte: Elaborado pelo autor, 2016.

É criado um *ZipInputStream* para a leitura do arquivo compactado. Cada arquivo contido neste zip é lido chamando-se getNextEntry. O FileOutputStream é criado no mesmo diretório onde se encontra o arquivo compactado, faz-se então a leitura do arquivo compactado e escrita no FileOutputStream.

4.3.5 *BroadcastReceiver*

Esta classe é executada em segundo plano, o *BroadcastReceiver* é utilizado para que a aplicação reaja a algum evento, além do mais possibilita que a aplicação possa receber uma mensagem e processá-la sem que o usuário perceba (LENCHETA, 2013). Nesta aplicação foi utilizado o

BroadcastReceiver para verificar se o sistema está em momento de *charge*. A figura 24 mostra como este processo foi implementado.

Figura 24 - Recebimento de Mensagem de Power Connected.

```
public class OnPowerReceiver extends BroadcastReceiver {
    int n;
    SharedPreferences pref;


    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        //con = context;
        pref = context.getSharedPreferences("Charges", 0);
        n = pref.getInt("Num", 0);
        Log.i("NumeroCharges",""+n);

        if (action.equals(Intent.ACTION_POWER_CONNECTED) && n==0) {

            Toast.makeText(context, "Iniciando Broadcast" + n, Toast.LENGTH_LONG).show();
            Log.d("onActivityResult", "Aparelho Carregando");
            n++;

            SharedPreferences.Editor editor = pref.edit();
            editor.putInt("Num", n);
            editor.commit();
            NotificationUtil.notify(context);
```

Fonte: Elaborado pelo autor, 2016.

A figura 24 demonstra a classe OnPowerReceiver, no momento em que o aparelho entra em *charge* essa classe é iniciada, um componente muito importante empregado é o SharedPreferences. Este elemento é utilizado para salvar dados do sistema sem ser necessário implementar um banco de dados, assim armazena dados em pares chave-valor. Além disso, ele foi utilizado para salvar a variável “n” com a chave “Num”, esta variável realiza uma contagem dos números de *charges*, isto é, se estiver em momento de *charge* e “n” for igual a 0 é iniciada uma notificação. A notificação é uma mensagem que é mostrada ao usuário para que este defina se o *backup* deve iniciar ou não.

A figura 25 demonstra como foi feito o truncamento do sistema. Ou seja, no momento em que o dispositivo é desconectado da energia, o *backup* é cancelado. Isso é feito terminando o *service* de *backup.*

Figura 25 – Recebimento de Mensagem Power Disconnected.

```
else {
  if (action.equals(Intent.ACTION_POWER_DISCONNECTED)) {

      Intent it = new Intent(context, MyService.class);
      context.stopService(it);
```

Fonte: Elaborado pelo autor, 2016.

Para que o sistema possa saber quais mensagens de *broadcast* deve iniciar a classe é importante definir isto no arquivo *AndroidManifest*. A figura 26 mostra que na aplicação foi necessário apenas definir as mensagens de *Power_Connected* e *Power_Disconnected*.

Figura 26 - *Receiver* no *AndroidManifest.*

```
<receiver
    android:name=".OnPowerReceiver"
    android:label="OnPowerReceiver"
    android:permission="android.permission.BATTERY_STATS">
    <intent-filter>
        <action android:name="android.intent.action.ACTION_POWER_CONNECTED" />
        <action android:name="android.intent.action.ACTION_POWER_DISCONNECTED" />
    </intent-filter>
</receiver>
```

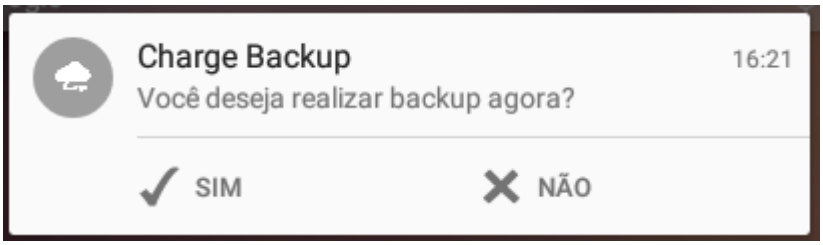
Fonte: Elaborado pelo autor, 2016.

Para definir o *BroadcastReceiver* no *Manifest* é necessário passar o nome da classe que vai implementá-lo, a permissão necessária para o seu funcionamento e em *intent-filter* deve-se destacar quais as mensagens deseja-se filtrar. Observa-se na figura 26 que as mensagens filtradas por este *receiver* são as de: *Power Connect* (Conectado a energia) e *Power Disconnect* (Desconectado da energia).

4.3.6 *Service*

A classe *service* é utilizada para executar atividades em segundo plano. Normalmente é vinculada a processo com um tempo indeterminado de execução, que pode consumir recursos, memória e CPU (LECHETA,2013). A aplicação possui dois *services*, um chamado de MyServiceNotify é gerado pela classe OnPowerConnect no momento em que o aparelho entra em momento de *charge* e a variável “n” for igual a zero. Isto é, sua função é gerar uma notificação para que o usuário possa aceitar o *backup* ou não. A figura 27 mostra como esta notificação é apresentada.

Figura 27 - Notificação de solicitação de *backup*



Fonte: Elaborado pelo autor, 2016.

O segundo *service* foi chamado MyService, iniciado pelo MyServiceNotify, no momento em que o usuário aceita a realização da transferência de dados. Este *service* é responsável pela realização do *backup*. Para isso foi utilizada a Google Drive API Android, que é destacada na próxima seção.

4.3.7 Google Drive API Android

Para a realização do *backup* e restauração de dados foi utilizada a Google Drive Api Android. Segundo Site oficial de desenvolvimento para Android da Google -2 (2016), esta API nativa fornece abstrações para gerenciar conteúdo e metadados de arquivos. Assim, nesta API os arquivos

são representados pela interface do DriveFile e as pastas pela interface do DriveFolder, ambos compartilham uma superinterface comum, DriveResource, que contém um DriveId. Este é o identificador exclusivo para todos os arquivos e pasta, a API também inclui um pacote para consulta de arquivos com base em atributos de metadados, como o título.

Para utilizar esta API foi necessário acessar o site de desenvolvedor Google e criar um credenciameto para a aplicação, para isso foi necessário fornecer o nome do pacote da aplicação e a chave SHA1, que é obtida através de um terminal, utilizando-se o seguinte comando.

keytool -exportcert -alias androiddebugkey -keystore C:\Users\Leticia\.android\debug.keystore -list –v

Logo após fornecer estes dados foi necessário ativar a aplicação na plataforma de desenvolvedor da Google e o programa ficou apto a utilizar esta API. No arquivo build.grandle, que é criado automaticamente pelo Android Studio, foi necessário adicionar alguns compiladores. Como mostra a figura 28.

Figura 28 - Compiladores da Google Play

```
compile 'com.android.support:appcompat-v7:24.2.0'
compile 'com.google.android.gms:play-services:9.4.0'
compile 'com.android.support:multidex:1.0.0'
```

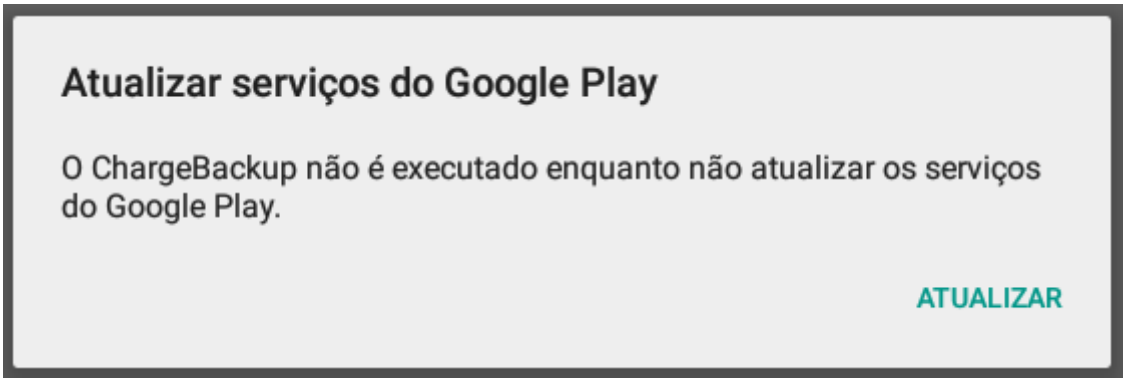
Fonte: Elaborado pelo autor, 2016.

O primeiro compilador é padrão do Android, o segundo é o compilador da Google Play, necessario para a utilização da Google Drive API. E o terceiro é utilizado para gerar o APK da aplicação, pois ao utilizar a API da Google a aplicação ficou muito grande e foi necessário utililizar um multidex que separa o arquivo .dex criado em varias partes e os junta no arquivo .apk.

Esta API possui muitas vantagens como facilitar a implementação de envio e busca de arquivos na Google Drive, possui suporte para truncamento,

maior gerencia de arquivos e metadados, entre outras. Ou seja, fazendo com que seja muito favorável sua utilização neste projeto, no entanto, ela limita o sistema a versão kitkat do Android ou versões mais recentes, o que já estava definido no momento de utilizar a classe *DocumentsProvider*. Desta forma, para a utilização desta API é necessário também ter instalado no aparelho a Google Play Service, assim se o dispositivo não possuir atualização deste serviço, será solicitada sua instalação ao abrir a aplicação, como mostra a figura 29.

Figura 29 - Solicitação de serviços da Google Play.



Fonte: Elaborado pelo autor, 2016.

Com a instalação deste serviço é possível utilizar o programa normalmente. Como diversas aplicações da Google necessitam deste serviço, a maioria dos dispositivos com sistema operacional Android, já possuem esta atualização.

Ao implementar esta API foi necessário, primeiramente, criar um GoogleApiClient, que é a solicitação de conta ao usuário, para isso, foi necessário criar um *builder* (construtor). Como mostra a figura 30.

Figura 30 - Criando um GoogleApiClient

```
protected void onResume() {
    super.onResume();
    if (mGoogleApiClient==null) {
            mGoogleApiClient = new GoogleApiClient.Builder(this)
                    .addApi(Drive.API)
                    .addScope(Drive.SCOPE_FILE)
                    .addConnectionCallbacks(this)
                    .addOnConnectionFailedListener(this)
                    .build();
    }
    mGoogleApiClient.connect();

}
```

Fonte: Elaborado pelo autor, 2016.

Este *builder* foi construído dentro do método on Resume, desta forma no momento em que é iniciado o programa é solicitada a conta ao usuário. Pode-se verificar, portanto, que ele só é chamado quando o m GoogleApiClient não for nulo, ou seja, ele é chamado apenas uma vez. Então, a própria API é responsável por manter a conta salva, a figura 31 mostra como as contas são mostradas ao usuário.

Figura 31 - Escolha de Conta Google.

Fonte: Elaborado pelo autor, 2016.

Todas as contas salvas no dispositivo são mostradas na forma de uma lista, para que o usuário escolha uma ou adicionar uma conta nova. Após,

definir o GoogleApiClient pode-se realizar o *backup* ou a restauração dos dados.

Para a realização do *backup* foi criado um processo semelhante à cópia de dados dentro do dispositivo, mostrado na seção 4.3.3. Porém, o *OutputStream* é criado através de um DriveContents, que é uma referência ao conteúdo de um arquivo do Drive, como destacado no site oficial de desenvolvimento para Android da Google -2 (2016). O *InputStream* é o arquivo cbackup compactado, após realizar a leitura e escrita é necessário criar o arquivo no Google Drive. A figura 30 destaca como foi realizado este processo.

Figura 32 - *Backup* para Google Drive.

```
MetadataChangeSet changeSet2 = new MetadataChangeSet.Builder()
        .setTitle(Arq.getName())
        .setMimeType("application/zip")
        .setStarred(true).build();

// Salva no arquivo root do Drive
Drive.DriveApi.getRootFolder(mGoogleApiClient)
        .createFile(mGoogleApiClient, changeSet2, driveContents)
        .setResultCallback(fileCallback);

Log.v("Arquivo Enviado ", Arq.getName());
```

Fonte: Elaborado pelo autor, 2016.

Primeiramente, foram definidos os Meta dados do arquivo a ser criado, para isso selecionou-se o título, como o mesmo do arquivo que está sendo copiado (cbackup) e o tipo para zip. Então, o arquivo é criado na pasta principal da Google Drive, para isso é necessário o GoogleApiClient, os meta dados setados e o DriveContent que possui a cópia do arquivo, é então criado um log para mostrar que o arquivo foi enviado.

Para a realização da restauração deve-se mostrar ao usuário as versões do cbackup existentes, para isto foi criada uma *Intent*, ela busca todos os arquivos da aplicação que forem zipados. Pode-se, portanto, verificar a implementação desta *Intent* na figura 33.

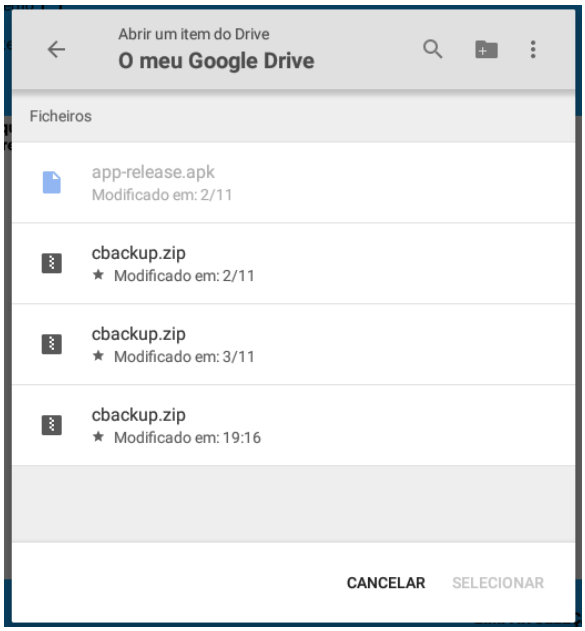
Figura 33 - Implementação da escolha de versão para restauração.

```
IntentSender intentSender = Drive.DriveApi
        .newOpenFileActivityBuilder()
        .setMimeType(new String[] { "application/zip" })
.build(mGoogleApiClient);
```

Fonte: Elaborado pelo autor, 2016.

Desta forma, são mostrados todos os arquivos da aplicação ao usuário, porém só os zipados estão habilitados para escolha, pode-se visualizar também, a data de modificação dos arquivos, assim é possível saber qual a data em que o *backup* foi realizado. Esta lista dos arquivos é observada na figura 34.

Figura 34 - Escolha da Versão do Arquivo cbackup no Drive.



Fonte: Elaborado pelo autor, 2016.

Esta seleção gera um *DriveId* que é o identificador do arquivo no drive, através deste identificador é possível criar um *Drive Contents* que na restauração gera um *input Stream* e é lido dentro da pasta ChargeBackup.

O *backup* é realizado em um *service*, assim quando o aparelho é desconectado da energia é gerado um *stopservice* que é utilizado para parar o serviço. Neste momento é desconectado o GoogleApiClient, desta forma, a própria API é responsável por realizar o truncamento do sistema. A API salva os dados localmente, desse modo, quando o aparelho entrar novamente em momento de *charge* os dados concluirão seu envio.

4.3.8 Permissões

Deve-se definir as permissões a serem aceitas pelo usuário, para isso acessa-se o *Manifest.permission* no site oficial do Android para utilizar as permissões existentes. Portanto, é possível criar novas, mas isto não é necessário para uma aplicação de *backup* simples, as utilizadas nesta aplicação são mostradas na figura 35.

Figura 35 - Permissões do Sistema

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.STORAGE" />
<uses-permission android:name="android.permission.BATTERY_STATS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

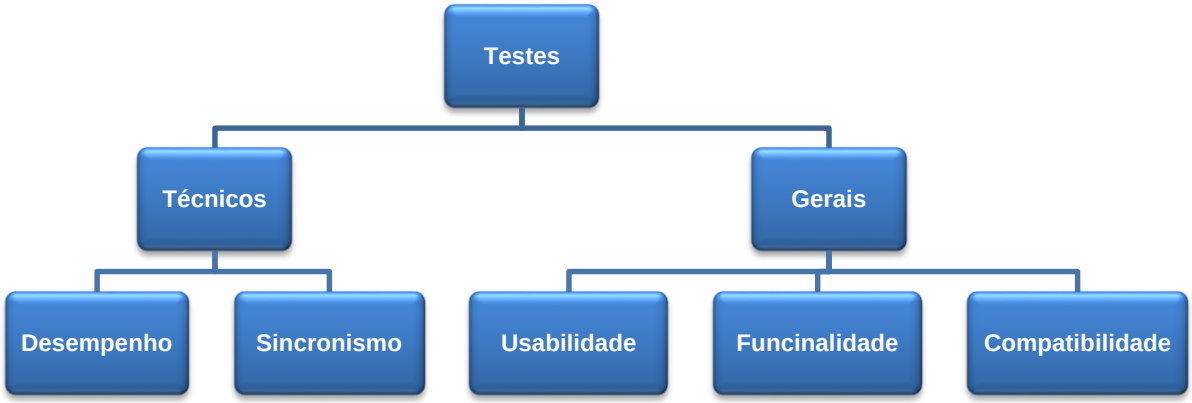
Fonte: Elaborado pelo autor, 2016.

Estas permissões são de extrema importância para a execução de aplicações no Android, pois sem elas não é possível acessar nenhuma funcionalidade do dispositivo. Permissões como *Internet*, *storage*, *write* e *read external storage* são fundamentais para o funcionamento desta aplicação, pois permitem acesso ao armazenamento e a *internet*, sem elas não seria possível uma aplicação de *backup*.

4.4 Testes– Charge Backup

Segundo Pressman *et al.*(2016), a realização de testes de *software* têm o objetivo de encontrar erros, sendo que um bom plano de testes é aquele que tem maior probabilidade de encontrar falhas. Estes processos são importantes para a validação do sistema, neste projeto os testes de *software* foram organizados em duas categorias: Técnicos e Gerais. Assim aa figura 36 demonstra a finalidade dos testes empregados.

Figura 36 - Diagrama de Testes.



Fonte: Elaborado pelo autor, 2016.

Os testes foram subdivididos, desta forma para que nos gerais se pudesse criar um Formulário de pesquisa, simples e prático. Sendo os técnicos mais complexos ao usuário comum, através deste plano de testes pode-se verificar a validação do sistema. A seção 4.4.1 irá destacar os testes técnicos realizados já na seção 4.4.2 serão mostrados os gerais.

4.4.1 Testes Técnicos

Estes testes tiveram como principal objetivo verificar o desempenho e o sincronismos dos processos de *backup* e restauração de dados. Para isto foi utilizado apena um aparelho, demonstrado na tabela 10.

Tabela 10 - Aparelho Utilizado nos Testes Técnicos.

Modelo	Marca	Android
Tablet Galaxy tab 2	Sansung	5.1.1 Lollipop

Fonte: Elaborado pelo autor, 2016.

O teste de desempenho, conforme Matos *et al.* (2012), é realizado para verificar a lentidão do sistema e desta forma, eliminar gargalos de desempenho, à vista disso, foi realizado medindo-se o tempo de *backup* e o tempo de restauração. Para isto, utilizou-se medições de tempo de própria IDE, esta medição foi efetuada utilizando uma rede WiFi e uma 3g. A rede WiFi é disponibilizado pelo ISP (*Internet Service Provider*) ZapTche[7] e tem uma velocidade de 2Mbps, a rede 3g é do ISP Vivo[8] e possui velocidade de 500kbps. Com isso, foram montados 4 cenários para medição do desempenho como mostra a tabela 11, nos dois ISP's transferiu-se um arquivo compactado com o tamanho de 17,46 MB. Sendo assim, foi também testado o truncamento da aplicação, ou seja, no momento da transferência ao desconectar o aparelho da energia, é parada a transferência e iniciada novamente no próximo momento de *charge*.

[7]http://zaptche.com.br/

[8]http://www.vivo.com.br/portalweb/appmanager/env/web

Tabela 11 - Cenários do Teste de Desempenho

AÇÃO	REDE
Restauração	WiFi
Restauração	3g
Backup	WiFi
Backup	3g

Fonte: Elaborado pelo autor, 2016.

O teste de sincronismo teve como objetivo verificar se há perdas de pacotes no momento da transferência de dados. Para comprovar isto, foi necessário implementar um experimento para a análise da transferência de dados, ou seja, transferiu-se três pacotes contendo diferentes dados (pacote1= 5,95MB, pacote2= 942KB, pacote3= 17,46MB) e se verificou se eles estavam completos após a transferência. Para a realização deste teste estes pacotes foram transferidos para a nuvem, então, utilizou-se o programa Zip Extrator[9], fornecido pela Google Drive, para descompactar os arquivos na nuvem. Desta forma, é possível verificar erros no momento de envio, após, foi realizada a restauração deste arquivo, se verificou se ocorreram erros no momento de *download.*

O formulário criado para realizar os testes gerais também aborda questões de desempenho e sincronismo, foi questionado, portanto, se o aparelho apresenta lentidão no momento da transferência e se foi possível utilizar outras aplicações enquanto o backup era realizado, validando assim o desempenho. Foi abordado também se os dados selecionados para backup são os mesmos obtidos na restauração, verificando-se assim o sincronismo, porém, os testes técnicos apresentam maior precisão de resultados como tempo de transferência e em qual método ocorreu perda de pacote.

[9]https://zip-extractor.appspot.com/

4.4.2 Testes Gerais

Para a realização dos testes gerais, foi criado um Formulário de Pesquisa, demonstrado no apêndice A. O Formulário tem como intuito testar a compatibilidade, usabilidade e funcionalidade da aplicação, por outras pessoas, em aparelhos variados. Foi disponibilizada o arquivo Charge Backup.apk, pacote para instalação da aplicação, e o formulário que foi testado em 7 aparelhos, listados na tabela 12.

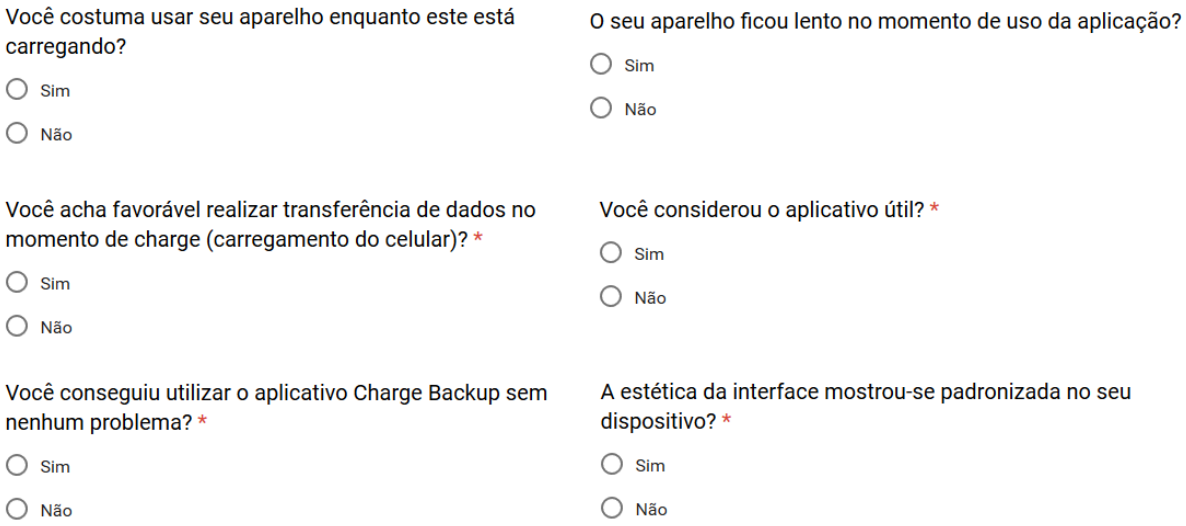
Tabela 12 - Aparelhos Utilizados para Testes Gerais.

Modelo	Marca	Android
Tablet Galaxy tab 2	Sansung	5.1.1 Lollipop
Redmi 2	Xiaomi	4.4.4 Kitkat
Tablet V700	LG	4.4.2 Kitkat
Galaxy Pocket 2	Sansung	4.4.2 Kitkat
Galaxy S5	Sansung	6.0.1 Marshmallow
Galaxy J3	Sansung	5.1.1 Lollipop
Galaxy Core 2	Sansung	4.4.2 Kitkat

Fonte: Elaborado pelo autor, 2016.

Com os testes gerais realizaram-se os testes de compatibilidade, usabilidade e funcionalidade, a fim de comprovar a validade do sistema através da utilização do usuário. Os sete aparelhos listados na tabela 12 foram utilizados por 6 usuários diferentes que responderam perguntas específicas para cada tipo de teste através do formulário em um período de uma semana. A figura 37 mostra algumas das perguntas mais relevantes para a execução destes testes.

Figura 37- Formulário disponibilizado para os usuários



Fonte: Elaborado pelo autor, 2016.

O teste de compatibilidade tem por objetivo verificar se a aplicação Charge Backup é compatível com diversos tipos de aparelhos. Segundo Matos *et al.*(2012), é necessário verificar a compatibilidade em dispositivos móveis, pois cada fabricante desenvolve seu próprio *hardware* e as aplicações devem se adaptar à eles. Por isso, foi necessário instalar a aplicação em dispositivos de diferentes marcas e modelos. Este teste foi realizado, verificando-se se a interface da aplicação encontra-se padronizada e se foi possível conectar-se à rede para realizar a transferência.

O teste de usabilidade verifica principalmente a interface da aplicação, bem como sua compreensão e facilidade, os testes de usabilidade são direcionados aos usuários. Além disso, são realizados verificando se pessoas diferentes conseguiram utilizar a ferramenta sem problemas, além do mais, foi verificado se os usuários aprovaram a ferramenta e se a consideraram útil. Verificou-se também se a realização da transferência em modo de *charge* é aprovada e se o aparelho é utilizado neste momento.

Segundo Maldonado *et al.*(2012), os testes funcionais verificam as saídas geradas pelo software, ignorando o comportamento interno. Assim, este teste foi realizado em três etapas:

- Primeira etapa: Foram escolhidos os arquivos e realizou-se o backup manualmente, clicando-se no botão da interface, desta forma, pode-se verificar a funcionalidade do processo*;*
- Segunda etapa: Foram escolhidos os arquivos e aguardou-se para realizar o *backup* no momento de *charge*, podendo-se, portanto, verificar a criação da notificação de solicitação de transferência e o início do processo em momento de *charge;*
- Terceira etapa: Realização da restauração dos dados assim como se verifica se são os mesmos escolhidos para o *backup* e se estão armazenados na memória escolhida.

Através deste plano de testes, pode-se discutir a integridade da aplicação, ou seja, cada processo testado tem como objetivo a validação do sistema e a aprovação dos diversos usuários em relação ao Charge Backup e, desta forma, verificar se a aplicação segue os requisitos definidos para o sistema. Os resultados obtidos e as discussões sobre os testes aplicados serão abordados no capítulo 5.

5 RESULTADOS E DISCUSSÕES

Através dos testes técnicos de desempenho que transferiu um arquivo compactado com 17,46 MB podem-se obter os resultados demonstrados na tabela13.

Tabela 13 – Resultados dos Testes de Desempenho.

	WiFi	**3g**	**Truncamento**
Backup	1,23s	5,21s	Sim
Restauração	1,19s	4,87s	Sim

Fonte: Elaborado pelo autor, 2016.

Através da tabela 13 pode-se observar que a velocidade de transferência dos arquivos tanto em *backup* quanto na restauração, são satisfatórias, levando em consideração a velocidade da rede. Segundo Bindal *et al.*(2006), para estimar a velocidade de download é necessário dividir o tamanho do pacote pela velocidade da *internet,* atentando-se as unidades. Isto é, como o pacote transferido era de 17,46MB para uma velocidade de conexão de 2Mbps (0,25MBps) seria igual à:

$$Tempo = \frac{17{,}46MB}{\frac{0{,}25MB}{60s}} = 1{,}16s$$

A velocidade obtida foi de 1,23s, portanto, obteve-se um aumento de menos 6% da velocidade estimada quando realizado o backup e ainda menor para a restauração. Para a conexão 3g, que possuía uma velocidade de 500kbps (0,061MBps) o tempo estimado foi de:

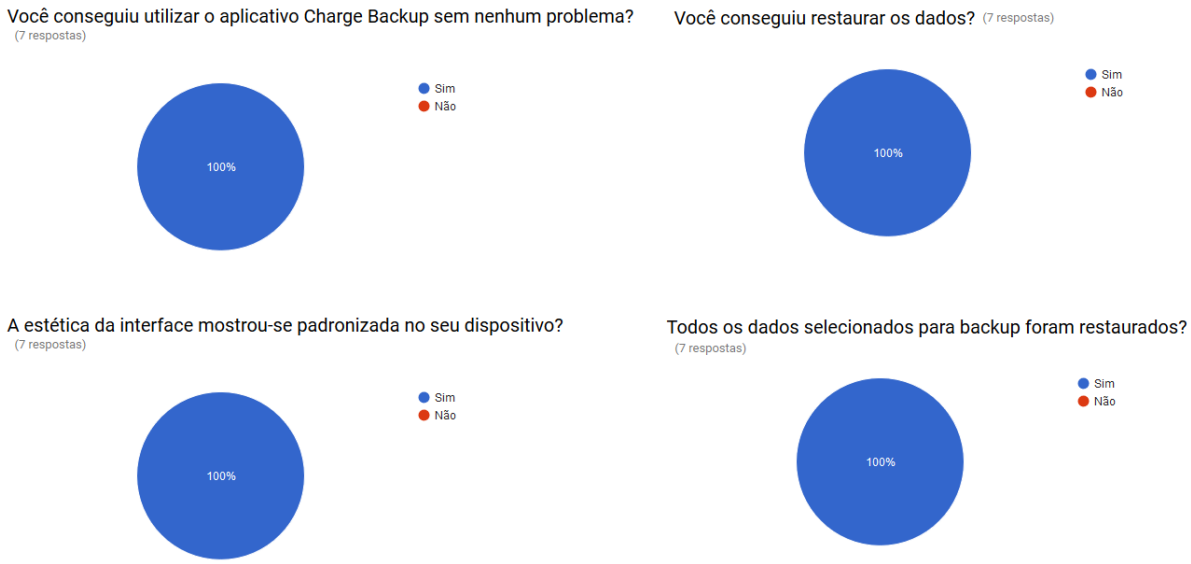
$$Tempo = \frac{17{,}46MB}{\frac{0{,}061MB}{60s}} = 4{,}77s$$

Obteve-se um aumento de apenas 8,44% no tempo estimado e menos que isso na operação de restauração, através destes resultados pode-se verificar que o tempo de transferência é muito semelhante ao estimado, as diferenças encontradas devem-se a Internet não manter uma velocidade constante, podendo sofrer variações. Todas as pessoas que testaram a aplicação, afirmaram que o dispositivo não ficou lento no momento da transferência e que foi possível utilizar outros aplicativos enquanto o *backup* era realizado. Além disso, foi verificado também que o truncamento ocorre de maneira correta nos dois momentos de transferência da aplicação.

Para o teste de sincronismo obteve-se os resultados esperados, tanto no momento em que os dados foram descompactados na nuvem como na restauração, as informações foram às mesmas das escolhidas para o *backup*, ou seja, não ocorrendo perdas de pacotes durante nenhuma das transferências e em nenhum momento da execução do programa.

Nos testes gerais de compatibilidade todos os dispositivos testados conseguiram rodar a aplicação normalmente. Em todos os modelos, o *layout* se mostrou padronizado e conseguiram enviar e restaurar dados, mostrando que foi possível acessar a *internet* e a Google Drive. A figura 38 mostra que ao responder o formulário todos os usuários conseguiram utilizar o programa sem problemas.

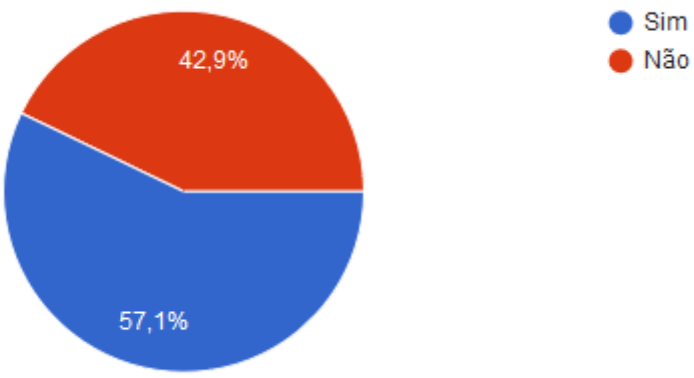
Figura 38 – Respostas obtidas nos testes de compatibilidade



Fonte: Elaborado pelo autor, 2016.

Por meio dos testes de usabilidade obteve-se a opinião dos usuários, os quais conseguiram manipular a aplicação com sucesso, validando a tela da aplicação como intuitiva e simples como Lima Junior *et al.* (2016), afirma que deve ser. Além do mais, cada um que utilizou a aplicação e respondeu ao formulário considerou a aplicação útil, ou seja, nenhum deixou alguma crítica ou sugestão de melhoria. Ademais, quando perguntado se o aparelho é utilizado no momento de *charge*, 42,9% responderam que sim, como mostra o gráfico da figura 39.

Figura 39 - Usuários que utilizam o aparelho em momento de *charge*.



Fonte: Elaborado pelo autor, 2016.

Porém, quando foi perguntado se acham útil realizar a transferência dos dados no momento de *charge,* 100% das pessoas responderam positivamente.

No teste de funcionalidade, todos os usuários conseguiram escolher os arquivos e realizar o *backup* manualmente, logo, validando a primeira etapa da funcionalidade. Consequentemente, as respostas foram favoráveis também na segunda etapa em que o backup ocorre automaticamente no momento de charge e na terceira quando é realizada a restauração dos dados. Igualmente, todos os utilizadores da aplicação conseguiram visualizar a notificação e iniciar o *backup* em momento de *charge,* assim como responderam que os dados selecionados foram restaurados com sucesso. À vista disso, esse processo permite verificar os testes de integração e aceitação abordados por Rocha *et al.* (2001) que possibilita avaliar o software em busca de falhas, por meio da utilização do mesmo e realizar operações de rotina do sistema, de modo a verificar se seu comportamento está de acordo com o solicitado.

Através dos testes realizados, pode-se obter validação da aplicação, além do mais como os testes se mostraram favoráveis, não foi necessária nenhuma manutenção do sistema. Além disto, observou-se que a maioria dos usuários não utiliza o dispositivo em momento de *charge*, mesmo os que utilizam acharam favorável a transferência neste momento, portanto, todos os usuários conseguiram utilizar a aplicação sem problemas, revelando que a plataforma é simples e de fácil utilização. Estes resultados comprovam que esta aplicação é funcional e soluciona o problema de pesquisa apresentado.

6 CONCLUSÕES

Este trabalho apresentou uma pesquisa sobre meios de salvaguarda de dados, que teve como intuito criar e implementar um modelo de *backup.* Assim, foi possível verificar que a maioria dos utilizadores de dispositivos móveis tem preocupação com os dados contidos em seus aparelhos. Porém, devido à complexidade da maioria das aplicações encontradas hoje em dia, poucos são os que têm algum software de salvaguarda de dados. Assim sendo, faz-se necessário a implementação de um sistema que facilite a utilização pelo usuário, isto é, que seja transparente, funcional e de fácil compreensão.

Com base nisto, concluiu-se que foi possível criar e implementar um modelo para salvaguarda da informação, simples de manipular, sendo totalmente intuitivo ao usuário o seu funcionamento, além do mais transparente na realização do *backup*, possibilitando a utilização de outros aplicativos enquanto os dados são transferidos. Consistindo assim em um sistema confiável em que todos os dados transferidos no *backup* podem ser restaurados.

A partir do formulário criado foi possível realizar os testes de desempenho, sincronismo, compatibilidade, usabilidade e funcionalidade. Após, a realização dos testes pode-se confirmar a validação do Charge Backup, além do mais a solução atendeu aos requisitos necessários para solução do problema proposto. Portanto, o fato da transferência ser realizada no momento de *charge* possibilita que o usuário possa utilizar outras aplicações enquanto este é realizado, facilitando assim que a cópia dos dados em uma plataforma em nuvem seja realizada frequentemente, isto é, sem atrapalhar a rotina de quem o utiliza. Com isso, pode-se analisar as necessidades dos usuários e verificar que o aplicativo desenvolvido funcionou da maneira correta.

6.1 Trabalhos Futuros

Como trabalhos futuros pretende-se implementar uma funcionalidade para que no momento da restauração os dados sejam copiados para seus diretórios de origem. Visando assim, um melhor funcionamento para o usuário, não sendo necessário que os arquivos sejam organizados manualmente na estrutura de arquivos, facilitando, desta forma, que outras aplicações possam manipulá-los em seus locais de origem. Outra funcionalidade a ser implementada é a solicitação na tela da aplicação na rede o usuário deseja que seja realizada a transferência, podendo-se permitir o início do *backup* apenas quando conectado a uma rede WiFi, 3g ou 4g. Isto é, não permitindo que uma transferência seja realizada em uma rede inconveniente ao usuário. Esta aplicação entrará em processo de envio ao Play Store da Google, após o término da escrita deste trabalho. Assim, pretende-se seguir dando-lhe manutenção e criando novas versões após o envio.

REFERÊNCIAS

AFAQ, Khan H. *et al.*(2009) **4G as a Next Generation Wireless Network.** 2009 International Conference on Future Computer and Communication.

AMORIM, Fabrício; AMARAL, Sérgio Tibiriçá. (2015) **Evolução Histórica da Liberdade de Imprensa no Plano Internacional**. ETIC-ENCONTRO DE INICIAÇÃO CIENTÍFICA-ISSN 21-76-8498.

BAKAR, Normi S. A. A; MAHMUD, Iqram (2013) **Empirical Analysis of Android Apps Permissions.**2013 International Conference on Advanced Computer Science Applications and Technologies.

BARBERA, Marcos *et al.* (2013) **To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing**. 2013 Proceedings IEEE INFOCOM.

BINDAL, Ruchir et al. (2006)**Improving traffic locality in BitTorrent via biased neighbor selection.** In: 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06). IEEE. p. 66-66.

BORTOLIN, Diogo *et al.* (2014) **Como Avaliar Serviços de Armazenamento em Nuvem? Do Estudo a Prática.**

CERVO, Amado L; BERVIAN, Pedro A; DA SILVA, Roberto (2007) **Metodologia Cientifica**. 6ª ed. Pearson Education do Brasil.

CYSNEIROS, Luiz Mareio; LEITE, J. C. S. P. (1997) **Definindo requisitos não funcionais.** XI Simpósio Brasileiro de Engenharia de Software. Fortaleza, CE, p. 33, 1997.

DAPPER, Roque Eduardo (2013) **Desenvolvimento e Implementação de Algoritmos de Compressão Aplicados à Qualidade da Energia Elétrica**.

DE ALMEIDA, andre aparecido leal; CITRO, elisangela. (2011) **Aplicação da modelagem UML na fase de análise de um projeto de software para agendamento de uso de veículos internos de uma empresa.**

DEUTSCH, L. Peter (1996) **DEFLATE compressed data format specification version 1.3.**

FERNANDES, João Carlos f. (2010) **Tecnologias de rede: aplicabilidade e tendências mercadológicas para redes sem fio e a utilização do 3G, WiMAX e LTE**. FaSCi-Tech 1.2

FIALHO JR, Mozart *et al.* (2007)**Guia essencial do backup**. Universo dos Livros Editora.

FONSECA FILHO, Cléuzio (2007) **História da computação: O Caminho do Pensamento e da Tecnologia**. EDIPUCRS.

FOWER, Martin(2005) **UML Essencial: Umbreve guia para a linguagem-padrão de modelagem de objetos.** 3ª ed. Bookman

FREIRE, Júlio G. da Fonte (2013) **Análise de Desempenho de Plataformas para Desenvolvimento com o Sistema Operacional Android.**

GAIKAR, Vishal (2012) **Top 10 Essential Android Apps.**

HANZO, Lajos *et al.* (2012) **Wireless Myths, Realities, and Futures: From 3G/4G to Optical and Quantum Wireless.** Proceedings of the IEEE 100.Special Centennial Issue (2012): 1853-1888.

HENNESSY, John. PATTERSON, David (2012) **Computer Architecture: A Quantitative Appoach.** Fifth Edition, Elsevier.

HU, Wen-Chen. *et al.* (2009) **Mobile Data Protection Using Handheld Usage Context Matching**. In 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware

KHAN, Afaq H. *et al*. (2009)**4G as a next generation wireless network. In:**

Future Computer and Communication. ICFCC 2009. International

Conference on. IEEE, 2009. p. 334-338.

KHOMH, F., YUAN, H. and ZOU, Y. (2012) **Adapting Linux for Mobile Platforms: An Empirical Study of Android**. In: 28th IEEE International Conference on Software Maintenance (ICSM).

KNUTH, Donald E. (1985) **Dynamic huffman coding. Journal of algorithms**, v. 6, n. 2, p. 163-180.

LENCHETA, Ricardo (2013) **Google Android: Aprenda a criar aplicações para dispositivos móveis com Android SDK.**Terceira edição, Novatec.

LESSA, Rafael Orivaldo; LESSA JUNIOR, Edson Orivaldo(2009)**Modelos de Processos de Engenharia de Software.** Link para o PDF: http://xps-project. googlecode. com/svn-history/r43/trunk/outros/02_Artigo. pdf, 2009.

LIMA JUNIOR, Guaratã Alencar F; SILVA, Rodrigo Cezario (2016)**Guia de Boas Práticas para Desenvolvimento de Interface e Interação para Desenvolvedores da Plataforma Android**. III Workshop de Iniciação Científica em Sistemas de Informação, Florianópolis, SC

MALDONADO, José Carlos et al. "**Introdução ao teste de software"**. São Carlos, 2004.

MATOS, Tiago; PIRES, Gustavo (2012) **A utilização de testes em aplicativos móveis.** Engenharia de Software, Uniffacs.

MUSLUKHOV, Ildar *et al.* (2012) **Understanding users' requirements for data protection in smartphones**. In ICDE Workshop on Secure Data Management on Smartphones and Mobiles.

NELSON, Mark; GAILLY, Jean-Loup (1996)**The data compression book**. New York: M&T Books.

OLIVEIRA, Hugo; DE AZEVEDO PINTO, Maria Manuela Gomes (2014) **A Gestão da Produção Informacional: o formato PDF e a comunicação via email.** Páginas a&b, p. 3-48.

OTTAVIANI, Vittorio *et al.* (2011) **Shared backup & restore: Save, recover and share personal information into closed groups of smartphones**, in Proc. of IFIP NTMS 2011.

PAULA FILHO, Wilson (2009) **Engenharia de Software: Fundaments, metodos e padrões**,

PEREIRA, Luciano; SILVA, Michel (2009) **Android para desenvolvedores**. Primeira edição, Brasport Livros e Multimidia Ltda.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software**(2016) 8ª Edição. McGraw Hill Brasil.

ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C. *et al.*(2001), "Qualidade de software – Teoria e prática", Prentice Hall, São Paulo.

RUSCHEL, Henrique *et al. (*2010) **Computação em Nuvem. Especialização em Redes e Segurança de Sistemas**. Pontifícia Universidade Católica do Paraná.

SHRIWAS, Medhavi S. *et al.*(2013) **Effeicient Method for Backup and Restore Data in Android.** International Conference on Communication Systems and Network Technologies.

SIMONSON, Peter *et al.* (2013) **The handbook of communication history.** Routledge.

WAHID, Abdul *et al.*(2014) **Anti-Theft Cloud Apps for Android Operating System.**2014 Sixth International Conference on Computational Intelligence and Communication Networks.

Cisco. **Tráfego global de dados móveis crescerá quase 10 vezes entre 2014 e 2019.**Disponivel em: http://www.cisco.com/web/PT/press/articles/2015/20150203.html. Acessado em 05/11/2015

Canaltech. **Quatro em cada cinco smartphones no mundo rodam Android, revela estudo.** Disponivel em: http://corporate.canaltech.com.br/noticia/smartphones/Quatro-em-cada-cinco-smartphones-no-mundo-rodam-Android-revela-estudo/. Acessado em 05/11/2015

Ferramentas PC: http://www.ferramentaspc.com.br/2015/02/como-descobrir-versao-do-android.htmlAcessado em 15/10/2015

Titanium backup (2011): http://www.titaniumtrack.com/titanium-backup.html Acessado em 25/11/2015

Android Headlines (2014): http://www.androidheadlines.com/2014/12/sponsored-app-review-cm-backup.html Acessado em 25/11/2015

Google Play: https://play.google.com/store/apps/details?id=mobi.infolife.appbackup Acessado em 25/11/2015

Fonseca, Gabi. Os principais diagramas da UML – Resumo Rápido. Disponível em: http://www.profissionaisti.com.br/2011/07/os-principais-diagramas-da-uml-resumo-rapido/. Acessado em: 25/11/2015

Site oficial de desenvolvimento para Android da Google -1 (2016) https://developer.android.com/about/dashboards/index.html Acessado em 02/11/2016

Site oficial de desenvolvimento para Android da Google -2 (2016)https://developers.google.com/drive/android/intro?hl=pt Acessado em 02/11/2016

Site oficial de desenvolvimento para Android da Google -3 (2016) https://developers.google.com/android/reference/com/google/android/gms/drive/DriveContents Acessado em 02/11/2016

APÊNDICE A – Formulário de Pesquisa

FORMULÁRIO DE VALIDAÇÃO DO PROTÓTIPO DE BACKUP

Questões Gerais

1. Você utiliza alguma aplicação de *backup* em seu dispositivo?

   ( ) Sim
   ( ) Não

   Em caso positivo, qual?

   _____

2. Você tem preocupação com a segurança dos dados armazenados em seu dispositivo?

   ( ) Sim
   ( ) Não

3. Qual o dispositivo você vai usar para testa a aplicação?

   ( )Tablet
   ( )Smartphone

   Qual a marca e modelo?

   _____

   Qual a versão do Android?

   _____

4. Você costuma utilizar seu aparelho enquanto este está carregando?

   ( ) Sim
   ( ) Não

5. Você acha favorável realizar transferência de dados no momento de *charge* (carregamento do celular)?

   ( ) Sim
   ( ) Não

Questões Específicas

6. Você conseguiu utilizar o aplicativo Charge Backup sem nenhum problema?

   ( ) Sim
   ( ) Não

   Em caso negativo, relate o problema.
   _____
   _____
   _____

7. A pasta ChargeBackup foi gerada no armazenamento escolhido (Interno ou Externo)?

   ( ) Sim
   ( ) Não

8. A estética da interface mostrou-se padronizada no seu dispositivo?

( ) Sim
( ) Não

9. Você conseguiu acessar e selecionar os arquivos para *backup*?

( ) Sim
( ) Não

10. Foi solicitada a conta da Google Drive?

( ) Sim
( ) Não

11. Foi gerado o arquivo cbackup na sua conta da Google Drive?

( ) Sim
( ) Não

12. Você conseguiu restaurar os dados?

( ) Sim
( ) Não

13. Você conseguiu escolher a versão do arquivo cbackup.zip a ser restaurada?

( ) Sim
( ) Não

14. Os dados restaurados foram descompactados?

( ) Sim
( ) Não

15. Todos os dados selecionados para *backup* foram restaurados?

( )Sim
( )Não

16. O seu aparelho ficou lento no momento de uso da aplicação?

( ) Sim
( ) Não

17. Você conseguiu utilizar outra aplicação enquanto estava realizando o *backup* ou restauração de dados?

( ) Sim
( ) Não

18. Você considerou o aplicativo útil?

( ) Sim
( ) Não

Em caso negativo, explique por que.

_____
_____
_____

19. Sugestões de melhorias:

_____
_____
_____