CENTRO UNIVERSITÁRIO UNIFACVEST

CIÊNCIA DA COMPUTAÇÃO

MARINA MOTTA DA COSTA SILVA

SMARTEYES – APLICATIVO ANDROID

LAGES-SC

2015

MARINA MOTTA DA COSTA SILVA

SMARTEYES – APLICATIVO ANDROID

Trabalho de Conclusão de Curso de Ciência da Computação submetido ao Centro Universitário Catarinense – UNIFACVEST, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação. Sob orientação do Professor MSc Márcio José Sembay.

LAGES - SC

2015

MARINA MOTTA DA COSTA SILVA

SMARTEYES – APLICATIVO ANDROID

Trabalho de Conclusão de Curso de Ciência da Computação submetido ao Centro Universitário Catarinense – UNIFACVEST, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação. Sob orientação do Professor MSc Márcio José Sembay.

LAGES, SC __/__/2015. Nota _____

_____

LAGES
2015

Dedico este trabalho ao meu pai, Alexandre Pagani Silva, exemplo de perseverança e honestidade. Sempre me guiando no caminho correto e me ensinando a não desistir dos meus sonhos. É graças a ti tudo o que sou hoje. Eu amo você!

RESUMO

O estudo realizado investigou o uso de Tecnologias Assistivas para tornar independente a vida de pessoas portadoras de algum tipo de deficiência visual que os impeça de realizar atividades ou escolhas sem o auxílio presencial de outra pessoa. Uma das maiores dificuldades encontradas foi a situação em que não é possível "substituir os olhos", ou seja, é realmente necessário possuir a visão saudável para tomar uma decisão, podendo tomar como exemplo encontrar uma peça de roupa de determinada cor em meio a uma pilha de roupas. Baseado nesse tipo específico de problema, foi desenvolvido um estudo das tecnologias capazes de serem utilizadas para criar um aplicativo Android com objetivo de realizar a interação da dificuldade encontrada pelo portador da deficiência e a solução do problema, por uma pessoa visualmente saudável, sem que esta esteja presencialmente no local do problema, tendo sido desenvolvida uma arquitetura computacional capaz de suportar trocas de informações (dados) em tempo real. A metodologia para execução deste trabalho deu-se partindo de pesquisas sobre as deficiências visuais e suas limitações na independência de vida do indivíduo que a possui e também estudos para encontrar as melhores tecnologias capazes de suprir a necessidade, ou boa parte dela, de uma visão saudável para solucionar questões cotidianas.

Palavras-chave: Tecnologia Assistiva; Aplicativos Android; Deficiência Visual;

ABSTRACT

The study investigated the use of Assistive Technologies for independent make life of people with some kind of visual disability that prevents you from performing activities or choices without the presence assistance of another person. One of the biggest difficulties was the situation where you can not "replace the eyes", that is, it really necessary to have healthy vision to make a decision, and may take as an example find a particular color of clothing amid a pile of clothes. Based on this specific type of problem, we developed a study of technologies that can be used to create an Android application in order to perform the interaction of difficulty encountered by the bearer of disability and the solution of the problem, for a visually healthy person without that in person is in trouble site, has been developed a computer architecture capable of supporting the exchange of information (data) in real time. The methodology for this assignment gave breaking research on visual impairment and its limitations in the individual's life of independence that owns and also studies to find the best technologies that can meet the need, or much of it, a vision healthy to solve everyday issues.

Keywords: Assistive Technology; Android applications; Visual Impairment;

RESUMEN

El estudio investigó el uso de tecnologías de asistencia para la vida marca independiente de las personas con algún tipo de discapacidad visual que le impide realizar actividades o decisiones sin la asistencia presencia de otra persona. Una de las mayores dificultades fue la situación en la que no se puede "sustituir a los ojos", es decir, lo que realmente es necesario tener una visión saludable para tomar una decisión, y puede tomar como ejemplo encontrar un color particular de la ropa en medio de una pila de ropa. En base a este tipo específico de problema, se desarrolló un estudio de las tecnologías que se puede utilizar para crear una aplicación Android con el fin de realizar la interacción de la dificultad encontrada por el portador de la discapacidad y la solución del problema, para una persona visualmente saludable sin que en persona se encuentra en el sitio problema, se ha desarrollado una arquitectura de computadora capaz de apoyar el intercambio de información (datos) en tiempo real. La metodología para esta tarea dio investigación rompiendo en la discapacidad visual y sus limitaciones en la vida del individuo de la independencia que posee y también estudia para encontrar las mejores tecnologías que pueden satisfacer la necesidad, o gran parte de ella, una visión saludable para resolver problemas cotidianos.

Palabras clave: Tecnologías de Apoyo; Aplicaciones de Android; Discapacidad Visual;

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE DIAGRAMAS

LISTA DE ABREVIATURAS E SIGLAS

ADT – *Android Developer Tools*

API – *Application Program Interface*

ART – *Android Run Time*

AV – Acuidade Visual

CDC – *Connected Device Configuration*

CDMA – *Code Division Multiple Access*

CEJA – *Centro de Educação de Jovens e Adultos*

CLDC – *Connected Limited Device Configuration*

CPU – *Central Processing Unit*

CVM – *Compact Virtual Machine*

DFD – Diagrama de Fluxo de Dados

EDGE – *Enhanced Date Rates For GSM Evolution*

FTP – *File Transfer Protocol*

GUI – *Graphical User Interface*

GSM – *Global System for Mobile Communication*

HD – *High Definition*

HTML – *Hyper Text Markup Language*

HTTP – *Hyper Text Transfer Protocol*

IBGE – Instituto Brasileiro de Geografia e Estatística

IBM – *Institute Business Machines*

IDE – *Integrated Development Enviroment*

J2ME – *Java 2 Micro Edition*

JIT – *Just In Time*

JVM – *Java Virtual Machine*

KVM – *Kilo Virtual Machine*

LTE – *Long Term Evolution*

NFC – *Near Field Communication*

OHA	–	*Open Handset Alliance*
OMS	–	Organização Mundial da Saúde
PDA	**–**	*Personal Digital Assistant*
ROM	–	*Ready Only Memory*
SGBD	–	Sistema de Gerenciamento de Banco de Dados
SMS	–	*Short Message Service*
SQL	–	*Structured Query Language*
TCP/IP	–	*Transmission Control Protocol/Internet Protocol*
UI	**–**	*User Interface*
UMTS	**–**	*Universal Mobile Telecommunications*
URI	–	*Uniform Resource Identifier*
URL	**–**	*Uniform Resource Locator*
XML	**–**	*eXtensible Markup Language*
WI-FI	–	*Wireless Fidelity*

SUMÁRIO

1. INTRODUÇÃO

O problema da deficiência visual tem aumentado e criando, com o passar do tempo, grandes dificuldades no quesito acesso às informações, ocasionando uma exclusão do meio virtual social e profissional de pessoas portadoras de deficiência visual.

Com a exploração de novas tecnologias, atualmente, o uso de aplicativos digitais e softwares são vistos como um forte elemento de desenvolvimento humano, capaz de proporcionar aos deficientes visuais um acesso ao mundo digital, permitindo que estes usufruam com o maior proveito possível, de acordo com suas necessidades e motivações.

É neste contexto, de inclusão, não somente digital, mas principalmente social, que este projeto foi desenvolvido, visando proporcionar aos portadores de deficiência visual, seja ela cegueira ou baixa visão, meios e recursos necessários, de maneira que estes sejam capazes de realizar escolhas ou sanar dúvidas quase que independentemente do auxílio de uma outra pessoa.

1.1 PROBLEMA

Independente da origem ser congênita ou adquirida, a população mundial estimada que possui algum tipo de deficiência visual é de 285 milhões de pessoas (65% da população), destes 246 milhões possuem baixa visão e 39 milhões são considerados cegos e 82% da população de cegos possuem idade superior a 50 anos (OMS, 2011). Com base na população brasileira, pelo censo de 2010, 3,5% da população possui algum tipo de deficiência visual (IBGE, 2010).

Sabe-se muito bem que existem diversos recursos para auxiliar na qualidade de vida e independência das pessoas que possuem algum tipo de deficiência visual. Entretanto essa população ainda enfrenta muitas dificuldades no dia a dia, muitas vezes dificuldades que passam despercebidas por pessoas sem deficiência. Coisas comum como escolher um cinto para combinar com uma blusa ou identificar a validade de um produto que não contém instrução em braille (método de leitura e escrita em relevo com base em 64 símbolos resultantes da combinação de 6 pontos e dispostos em colunas de 3 pontos (Instituto Benjamin Constant, 2005)) pode ser uma tarefa muito difícil, senão impossível para uma pessoa cega que esteja sozinha.

Baseando-se nessas informações podemos definir a pergunta que originou este estudo: “Como elaborar uma arquitetura voltada à população brasileira de deficientes visuais que, juntamente com a tecnologia da informação, seja capaz de sanar as dificuldades cotidianas que atualmente só podem ter resultado com o auxílio presencial de uma outra pessoa que não possua deficiência?”

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

Propor uma arquitetura computacional com base no desenvolvimento *mobile* capaz de suportar trocas de informações (dados) em tempo real, visando auxiliar na independência e qualidade de vida da população com deficiência visual.

1.2.2 OBJETIVOS ESPECÍFICOS

Para que o objetivo principal seja atingido, faz-se necessário que alguns objetivos mais específicos sejam alcançados previamente, sendo eles:

a) Ser capaz de realizar uma varredura no servidor em busca de usuários cadastrados que estejam conectados à internet, diferenciando-os em grupos de usuários portadores de deficiência e não portadores;
b) Enviar uma notificação, a ser mostrada na barra de tarefas do *smartphone* e enviar um sinal sonoro, cada vez que houver um novo questionamento ou uma nova resposta;
c) Não permitir que um mesmo questionamento seja respondido mais de uma vez, pois isto pode causar confusão ao questionador;
d) Permitir que o usuário tenha o poder de decisão para que o aplicativo seja ou não ativado automaticamente ao conectar-se à internet;
e) Permitir que o usuário decida entre ativar ou não os comandos sonoros do aplicativo, permitindo assim a utilização do mesmo por usuários portadores de cegueira ou baixa visão.

2. REFERENCIAL TEÓRICO

A seguir serão apresentados conceitos e estudos que fundamentam e justificam a realização deste trabalho.

2.1 DEFICIÊNCIA VISUAL E BAIXA VISÃO

2.1.1 CONCEITOS

De acordo com Conde (2012) aquele que apresente desde alguma percepção luminosa capaz de determinar formas a curtíssimas distâncias até a ausência total da visão, são considerados cegos ou de visão subnormal, baixa visão.

> A cegueira engloba prejuízos da aptidão para o exercício de tarefas rotineiras exercidas de forma convencional, através do olhar, só permitindo sua realização de formas alternativas. A cegueira total ou simplesmente AMAUROSE, pressupõe completa perda de visão. A visão é nula, isto é, nem a percepção luminosa está presente. No jargão oftalmológico, usa-se a expressão 'visão zero'. (CONDE, pág. 01, 2012)

Cegueira parcial pode ser definida pelo indivíduo que é apenas capaz de contar dedos a curta distância e também aquele que só percebe vultos, este indivíduo é capaz de identificar de onde provém a luz. Ainda considerados integrantes do grupo da cegueira parcial, porém muito próximos da cegueira total, estão os indivíduos que só tem percepção de projeções luminosas, apenas fazem distinção entre o claro e o escuro. (CONDE, 2012)

Entretanto, Conde (2012) ainda nos explica que pedagogicamente, são considerados cegos todos os indivíduos que necessitam da utilização do Braille ou softwares especializados de leitura de textos, até mesmo os indivíduos que possuem baixa visão. Também são considerados cegos os indivíduos que utilizam para leitura textos ampliados ou fazem uso de potentes recursos ópticos.

Tratando-se de medicina, Conde (2012) afirma que o indivíduo pode ser considerado cego se enquadra-se em uma das seguintes opções:

a) A visão corrigida do melhor dos seus olhos é de 20/200 ou menos. Isto significa que ele pode ver a 20 pés (6 metros) o que um indivíduo com visão normal enxerga a 200 pés (60 metros);
b) Se o diâmetro mais largo do seu campo visual subentende um arco não maior de 20 graus, mesmo que sua acuidade visual neste campo seja superior a 20/200;

E ainda tratando-se de medicina, são considerados indivíduos com baixa visão aqueles que possuem acuidade visual de 6/60 e 18/60 e/ou com campo visual entre 20º e 50º. (CONDE, 2012)

2.1.2 ACUIDADE VISUAL

Acuidade visual (AV) é o grau de aptidão do olho, a capacidade de perceber a forma e contorno dos objetos. Essa capacidade é atribuída pelos cones (células fotossensíveis da retina). O aparelho óptico do olho é muito complexo. Para uma boa visão, a luz tem de atravessar uma córnea não distorcida, um cristalino normal e o corpo vítreo, antes de atingir uma retina saudável, que está ligada ao cérebro pela via óptica. (LEAL, 2014)

Classificação	Acuidade Visual de Snellen	Acuidade Visual Decimal	Auxílios
Visão Normal	20/12 a 20/25	1,5 a 0,8	▪ Bifocais comuns
Próximo do normal	20/30 a 20/60	0,6 a 0,3	▪ Bifocais mais fortes ▪ Lupas de baixo poder
Baixa visão moderada	20/80 a 20/150	0,25 a 0,12	▪ Lentes esferopris-máticas ▪ Lupas mais fortes
Baixa visão profunda	20/500 a 20/1000	0,04 a 0,02	▪ Lupa montada telescópio ▪ Magnificação vídeo ▪ Bengala ▪ Treinamento Orientação/Mobilidade
Próximo à cegueira	20/1200 a 20/2500	0,015 a 0,008	▪ Magnificação vídeo livros falados, Braille ▪ Aparelhos de saída de voz ▪ Softwares com sintetizadores de voz ▪ Bengala ▪ Treinamento Orientação/Mobilidade
Cegueira total	Sem projeção de luz	Sem projeção de luz	▪ Aparelhos de saída de voz ▪ Softwares com sintetizadores de voz ▪ Bengala ▪ Treinamento Orientação/Mobilidade

Tabela 1: Classificação de Acuidade Visual
Fonte: Classificação ICD – 9- CM (WHO/ICO)

A tabela de Snellen é o sistema padrão universal para avaliar a visão. Este teste consiste em ler linhas de letras cujo tamanho vai diminuindo e as quais são estão a uma distância padronizada. Por convenção, a visão pode ser medida ou na distância de 20 pés (6 metros), ou ainda mais perto, a 14 polegadas de distância. Para fins de diagnóstico, a distância da acuidade é o padrão para comparação, sendo sempre testado cada olho separadamente. (LEAL, 2014)

2.1.3 CAMPO VISUAL

Refere-se a área que é visível com os olhos fixados em determinado ponto. Ou seja, é a área passível de ser vista para a frente, para as laterais, para cima e para baixo, quando o indivíduo está imóvel, com o olhar em um ponto fixo, em linha reta horizontal paralela ao solo. (VEJAM, 2014)

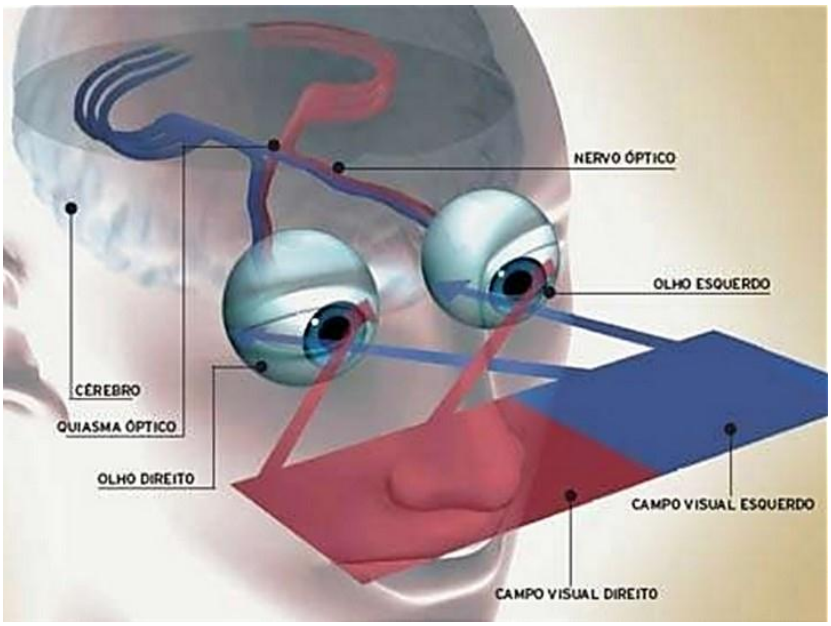

Figura 1: Representação do Campo Visual
Fonte: Loureiro, 2014

2.1.4 CAUSAS

Amaral (2012) afirma que entre as mais frequentes causas da cegueira infantil estão a rubéola, sarampo, toxoplasmose congênita, entre outras. Já em adultos, os fatores mais frequentes que levam a cegueira são a catarata, diabetes, descolamento de retina, glaucoma, entre outras. A OMS recomenda tratamentos precoces de várias doenças oculares, as quais são preveníveis ou tratáveis com a devida intervenção oftalmológica. Dependendo de fatores

como o tempo de início do tratamento, sua qualidade e regularidade podem definir a saúde do olho, podem levar a pessoa à cegueira, à baixa visão, ou a cura.

2.1.5 PREVENÇÃO

Principalmente na infância, é essencial que se tenha uma atenção especial com a saúde dos olhos. A Fundação Dorina Nowill (2014) cita os seguintes cuidados que devem ser tomados:

a) Seguir corretamente o pré-natal, pois várias doenças podem causar cegueira ou baixa visão no feto;
b) Realizar exame oftalmológico no recém-nascido sempre que for observada qualquer alteração ocular, como: olhos muito grandes, lacrimejamento intenso, mancha branca na menina dos olhos;
c) Vacinar periodicamente a criança para evitar doenças que possam causar problemas visuais, como sarampo, rubéola, meningite, varíola, entre outras;
d) Usar medicações e colírios somente com indicação médica;
e) Deixar fora do alcance das crianças produtos de limpeza, objetos pontiagudos (facas, arame, tesoura), fogos de artifício e plantas tóxicas;
f) Procurar um médico ao entrar ciscos ou fagulhas nos olhos. Não esfregar nem retirar com a ajuda de objeto caseiro;
g) Usar cinto de segurança no trânsito e colocar crianças no banco traseiro;
h) Fazer aconselhamento genético em caso de casamento consanguíneo.

2.1.6 TECNOLOGIA ASSISTIVA

Tecnologia assistiva é o termo utilizado para conceituar um conjunto de serviços e recursos que contribuem para proporcionar ou ampliar habilidades funcionais de pessoas com deficiência, promovendo vida independente e inclusão. (BERSH; SARTORETTO, 2014)

Existe no Brasil o Comitê de Ajudas Técnicas – CAT – e propõe o seguinte conceito: "Tecnologia Assistiva é uma área do conhecimento, de característica interdisciplinar, que engloba produtos, recursos, metodologias, estratégias, práticas e serviços que objetivam promover a funcionalidade, relacionada à atividade e participação de pessoas com deficiência, incapacidades ou mobilidade reduzida, visando sua autonomia, independência, qualidade de vida e inclusão social" (ATA VII - Comitê de Ajudas Técnicas (CAT) - Coordenadoria

Nacional para Integração da Pessoa Portadora de Deficiência (CORDE) - Secretaria Especial dos Direitos Humanos - Presidência da República, 2014).

Segundo Bersh e Sartoretto (2014), são considerados recursos qualquer item, equipamento ou parte dele, produto ou sistema fabricado em série ou sob medida, utilizado para aumentar, manter ou melhorar as capacidades funcionais das pessoas com deficiência. E os serviços seriam definidos como sistemas que auxiliam a pessoa com deficiência a selecionar, comprar ou utilizar os recursos.

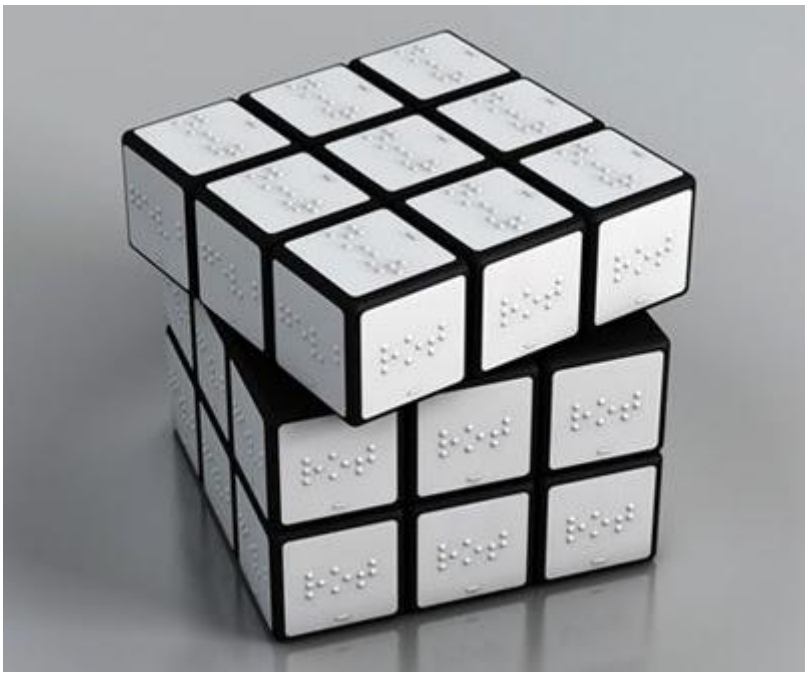

Figura 2: Tecnologia Assistiva para cegos: Cubo Mágico em Braille
Fonte: Débora, 2010

2.1.7 BRAILLE

2.1.7.1 LOUIS BRAILLE

Louis Braille nasceu no início do século XIX em Coupyray, munícipio localizado próximo de Paris. Quando tinha apenas três anos de idade, Louis sofreu um acidente ao brincar na oficina de seu pai, o que o fez perder completamente a visão. Com o passar dos anos, os pais de Louis o enviaram para estudar em uma escola especial para cegos. (SOUSA, 2008)

Nesta escola ele conheceu a jovem Teresa von Paradise, também cega, que o ensinou o gosto pela música. Juntos eles utilizavam um aparelho para a leitura e composição de partituras para piano. Louis sempre fora muito dedicado e estudioso, com o tempo aperfeiçoou-se na música e passou a realizar concertos. Em uma de suas apresentações conheceu Alphonse Thibaud, conselheiro comercial do Estado Francês, que sugeriu que Louis criasse um método para permitir a leitura e escrita para os cegos. (SOUSA, 2008)

Sentindo-se desafiado, passou a estudar e realizar diversos experimentos. Após três anos, Loius Braille estabeleceu um novo sistema de escrita e leitura especialmente para cegos. E foi no ano de 1829 que ele publicou este código no livro: “Processo para escrever as palavras, a música e o canto-chão, por meio de pontos, para uso dos cegos e dispostos para eles”. Infelizmente Loius faleceu antes de ver seu trabalho ser reconhecido, no ano de 1852. (SOUSA, 2008)

2.1.7.2 CÓDIGO BRAILLE

Costa (2009), afirma que o código Braille baseia-se em 64 símbolos em relevo, resultantes da combinação de até 6 pontos dispostos em duas colunas de três pontos cada. Com estas combinações é possível realizar a representação de letras, algarismos e sinais de pontuação, e sua leitura é feita da esquerda para a direita, ao toque de uma ou duas mãos ao mesmo tempo.

> O Brasil conhece o sistema desde 1854, data da inauguração do Instituto Benjamin Constant, no Rio de Janeiro, chamado, à época, Imperial Instituto dos Meninos Cegos. Fundado por D. Pedro II, o instituto já tinha como missão a educação e profissionalização das pessoas com deficiência visual. "O Brasil foi o primeiro país da América Latina a adotar o sistema, trazido por José Álvares de Azevedo, jovem cego que teve contato com o Braille em Paris", conta a pedagoga Maria Cristina Nassif, especialista no ensino para deficiente visual da Fundação Dorina Nowill. (COSTA, pág 01, 2009)

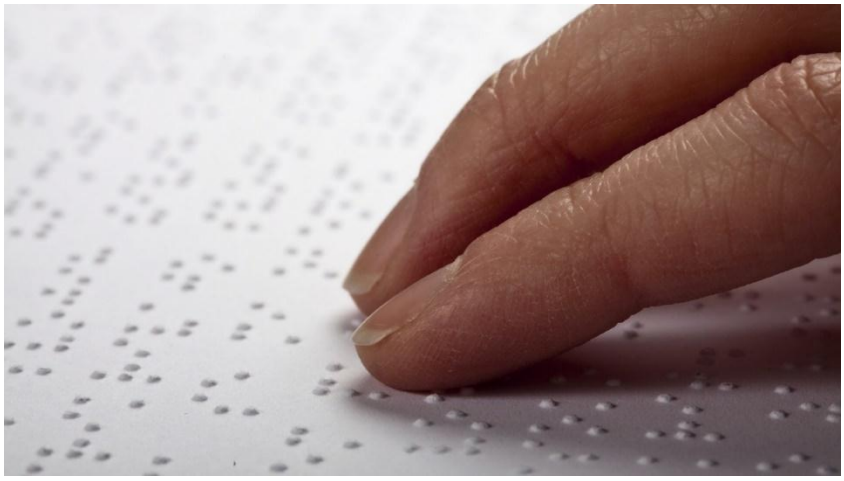

Figura 3: Sistema Braille
Fonte: Rewordit, 2015

2.2 JAVA

2.2.1 CONTEXTO HISTÓRICO DA LINGUAGEM

Em 1991, a Sun Microsystems, percebendo o impacto profundo que os microprocessadores têm em dispositivos eletrônicos inteligentes de consumo popular, financiou uma pesquisa corporativa interna que resultou em uma linguagem de programação baseada no C++, a qual foi batizada pelo seu criador, James Gosling, de Oak. Este nome foi escolhido pois da janela de James na Sun, ele enxergava uma árvore de carvalho. Depois foi descoberto que já existia uma linguagem com esse nome, e então quando a equipe da Sun visitou uma cafeteria, Java (nome de uma cidade de origem de um tipo de café importado) foi sugerido e ficou. (DEITEL, 2005)

Figura 4: Java, linguagem de programação
Fonte: Farah, 2014

> A contribuição mais importante da revolução do microprocessador até essa data (1991) é que ele tornou possível o desenvolvimento de computadores pessoais, que agora contam com mais de um bilhão em todo o mundo. Os computadores pessoais afetaram profundamente vida das pessoas e a maneira que as organizações conduzem e gerenciam seus negócios. (DEITEL, pág. 22, 2005)

O mercado para dispositivos eletrônicos inteligentes não desenvolveu da maneira que a Sun esperava, mas por sorte a Web deu um salto de popularidade em 1993 e a Sun resolveu utilizar o Java para criar conteúdo dinâmico às páginas Web. Em 1995, a Sun apresentou formalmente o Java, e este chamou atenção justamente pela interação na Web. (DEITEL, 2005)

Com a crescente utilização do Java, empresas como a IBM anunciaram suporte ao Java. Estudos relatam que a tecnologia Java foi que cresceu mais rapidamente na história da informática. Menos de 10 anos após o seu lançamento oficial, a linguagem já contava com 4 milhões de desenvolvedores. E o objetivo inicial do projeto, que propunha uma linguagem capaz de ser utilizada em diferentes dispositivos foi finalmente alcançada. (PACIEVITCH, 2014)

2.2.2 FUNCIONAMENTO

2.2.2.1 CARACTERISTICAS

A linguagem de programação Java foi projetada para atender os seguintes objetivos (DEITEL, 2005):

a) Orientação à Objetos – baseando-se no modelo de Smaltalk e Simula67
b) Portabilidade – Independência de plataforma – “write once, run anywhere”
c) Recursos de Rede – Possuir extensa biblioteca de rotinas que facilitem a cooperação com protocolos TCP/IP, como HTTP e FTP
d) Segurança – Pode executar programas via rede com restrições de execução

E algumas de suas vantagens:

a) Sintaxe similar a Linguagem C/C++.
b) Facilidades de Internacionalização – Suporta nativamente caracteres Unicode.
c) Simplicidade na especificação, tanto da linguagem como do “ambiente” de execução (JVM).
d) É distribuída com um vasto conjunto de bibliotecas (ou APIs)

e) Possui facilidades para criação de programas distribuídos e multitarefa (múltiplas linhas de execução num mesmo programa)
f) Desalocação de memória automática por processo de coletor de lixo (G*arbage Collector*)
g) Carga Dinâmica de Código - Programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

2.2.2.2 JAVA VIRTUAL MACHINE (JVM)

Diferente de outras linguagens de programação, onde o programa é traduzido para linguagem de máquina e comunica-se diretamente com o sistema operacional, no Java isso ocorre de forma diferente, seus programas são traduzidos em *ByteCode* e comunicam-se com a JVM, *Java Virtual Machine*. (FLAVIO, 2010)

A JVM que é responsável por traduzir os *ByteCodes* em linguagem de máquina, dessa maneira pode-se atingir o objetivo da portabilidade, que diz que os sistemas Java rodam em qualquer sistema operacional/dispositivo. (FLAVIO, 2010)

> Os processos de execução de um software Java foram aperfeiçoados ao longo dos tempos, pois no início, a Virtual Machine interpretava apenas um ByteCode por vez. Hoje em dia, a JVM possui sistemas de compilação JIT (Just - In - Time) misturados com a interpretação do código. Essa técnica cria os chamados “Hot-Spots”, que nada mais são que áreas de código executadas com maior frequência. Isso ocorre com a análise dos ByteCodes à medida que eles são interpretados pela Virtual Machine. (ROMANATO, pág. 01, 2014)

Segundo a definição da Sun, a principal responsável pela criação da linguagem Java, a JVM pode ser entendida como "uma máquina imaginária implementada via software ou hardware que executa instruções vindas de bytecodes". (ALECRIM, 2005)

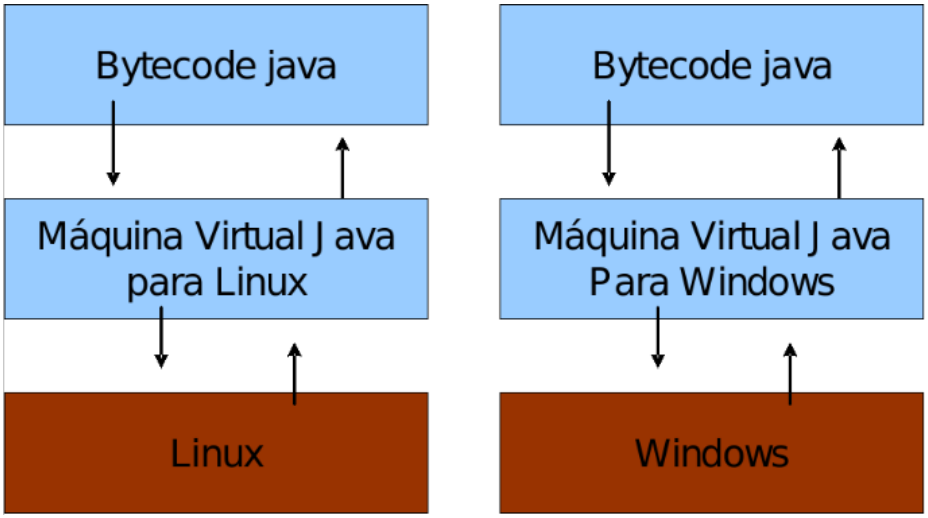

Figura 5: Exemplificação da Java Virtual Machine
Fonte: Caelum

2.2.2.3 FRAMEWORKS

Segundo Deitel (2005), entre os frameworks mais utilizados estão:

a) Hibernate – Framework para o mapeamento objeto-relacional;
b) Spring – Ferramenta que auxilia a implementação de injeção de dependência e inversão de controle;
c) Log4j – Ferramenta para auxiliar na criação de logs da aplicação;
d) JUnit – Ferramenta para auxiliar na criação de testes unitários;
e) Struts – Auxilia na criação de aplicações para Web.

2.2.3 JAVA PLATAFORM MICRO EDITION (J2ME)

Java 2 Micro Edition (J2ME) é a edição da linguagem Java para ser utilizada em dispositivos móveis, que possuem baixa capacidade de processamento e pouca memória disponível, alimentação elétrica por baterias, e limitados e variados métodos de entrada e saída. Alguns exemplos destes dispositivos seriam os telefones celulares, *pagers*, PDAs(Assistentes Digitais Pessoais), *Palms*. (DIAS; FONTES, 2003)

> No passado, todo dispositivo era oferecido com um conjunto fixo de funcionalidades cuja programação era realizada exclusivamente pelo fabricante, sobre uma tecnologia altamente proprietária. Através de J2ME, torna-se possível desenvolver, atualizar e instalar novas aplicações segundo as necessidades particulares de cada usuário. (DIAS; FONTES. Pág. 37. 2003)

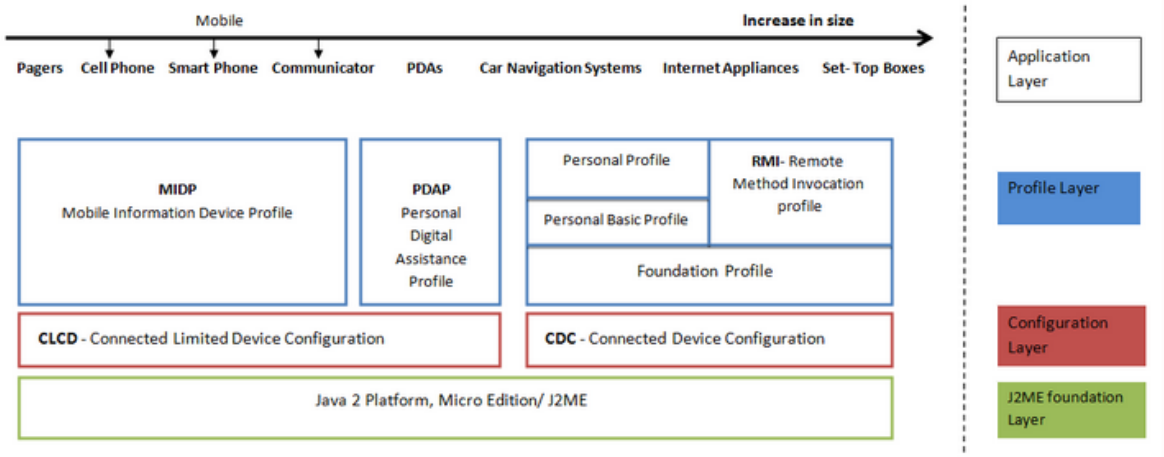

Figura 6: Arquitetura J2ME
Fonte: R4R

Segundo Dias e Fontes (2003), as vantagens de utilizar Java no desenvolvimento de aplicativos para dispositivos móveis são listadas a seguir:

a) Dinamismo: novas aplicações podem ser baixadas da rede e instaladas a qualquer momento;
b) Segurança: a forma de programação da linguagem garante a proteção das informações carregadas pelo dispositivo. Dados de uma aplicação não são acessíveis por outras aplicações;
c) Portabilidade: aplicações podem ser executadas em dispositivos de diferentes fabricantes e diferentes tipos;

2.2.3.1 MÁQUINAS VIRTUAIS J2ME

Assim como nas versões *Standard* (J2SE) e *Enterprise* (J2EE), a versão J2ME também necessita de uma máquina virtual para converter o *bytecode* em código binário. Entretanto, por estar executando em um dispositivo diferente das outras duas versões, que foram projetadas para aplicações para *desktop* e servidores respectivamente, fez-se necessário criar uma nova JVM especialmente para o *Java Micro Edition*. Foram então projetadas duas máquinas virtuais: *Kilo Virtual Machine* (KVM) – para ser utilizada na configuração CLDC e a *Compact Virtual Machine* (CVM) – para ser utilizada na configuração CDC. (DIAS; FONTES, 2003)

2.2.3.1.1 KILO VIRTUAL MACHINE (KVM)

Possui cerca de 40 a 80K de memória tornando-se assim muito apropriada para dispositivos como celulares, *pagers*, entre outros. Quando foi projetada, a KVM tinha por objetivo criar uma máquina virtual compacta que não fosse inferior em qualidade às outras máquinas virtuais da linguagem e que fosse capaz de executar em um dispositivo com recursos limitados, originado assim o seu nome Kilo com origem em *kilobytes*. (DIAS; FONTES, 2003)

Segundo Dias e Fontes (2003), ela foi implementada na linguagem C e é capaz de executar em qualquer sistema que possua um processador de 16 ou 32 bits e um total de 160 a 512K. Dentre as suas especificações, a KVM foi projetada para ser:

a) Pequena e com baixo requisito de memória;
b) Enxuta e portável;
c) Modular e customizável;
d) Completa e rápida como suas "versões anteriores"

2.2.3.1.2 COMPACT VIRTUAL MACHINE (CVM)

Pelo fato de a KVM não suportar dispositivos de uma segunda categoria e com o objetivo de adicionar mais funcionalidades, surgiu a CVM – *Compact Virtual Machine*. De uma maneira mais otimizada, essa máquina virtual é capaz de atender todas as características de uma máquina virtual Java convencional, feito que a KVM não conseguiu. Dentre suas características, pode-se citar as principais (DIAS; FONTES, 2003):

a) Melhor desempenho em aplicações real time;
b) Coleta automática de lixo da memória (*Garbage Collector*);
c) Mapeamento direto de *threads* Java para *threads* nativos;
d) Execução de classes diretas da memória ROM;
e) Portabilidade;
f) Sincronização com um reduzido número de instruções.

2.3 INTEGRATED DEVELOPMENT ENVIROMENT – IDE

Segundo Deitel (2005), IDEs – *Integrated Development Enviroment* – ou em português Ambiente Integrado de Desenvolvimento – são ferramentas que suportam o processo de desenvolvimento de *software*, incluindo editores e depuradores para encontrar erros de lógica e em tempo de compilação encontrar erros de sintaxe.

> Um ambiente de desenvolvimento integrado (IDE) é um ambiente de programação que foi empacotado como um programa de aplicação, geralmente constituídos por um editor de código, um compilador, um depurador e uma interface gráfica do usuário (GUI) construtor. O IDE pode ser um aplicativo independente ou pode ser incluído como parte de uma ou mais aplicações existentes e compatíveis. – traduzido (ROUSE, pág. 01, 2007)

2.3.1 ECLIPSE

Gastando mais de 40 milhões de dólares, a IBM (*International Business Machines*), em 2001, desenvolveu o projeto Eclipse e o entregou para a comunidade como um software livre (open source) (SANTANA, 2011). “Hoje o ecossistema Eclipse congrega 170 empresas, 270 projetos e 50 milhões de linhas de código e é um dos maiores sucessos da história das comunidades *open source*.” (BLEWITT, 2011).

Segundo Blewitt (2011), antes do ano 2000 já existiam diversos ambientes para o desenvolvimento em Java, entretanto estes ambientes eram focados na linguagem e não estavam acompanhando os avanços dos *servlets* e páginas HTML. Em junho de 1998 a IBM lançou o *WebSphere* que consistia em basicamente um repositório de *servlets*, surge então a necessidade de complementar o ambiente de desenvolvimento Java e em setembro daquele mesmo ano a empresa lança o *WebSphere Studio* que três anos depois evoluiria para tornar-se o Eclipse 1.0.

Em 5 de novembro de 2011 a IBM apresenta oficialmente o Eclipse:

“O *WebSphere Studio* é a primeira ferramenta disponível comercialmente a ser baseada em um software de código aberto, de codinome Eclipse, que está sendo doado pela IBM a uma nova comunidade de fornecedores de ferramentas de software.” (BLEWITT, 2011)

Nesta época a empresa notou uma diminuição no número de usuário do *VisualAge* (ambiente de desenvolvimento Java da própria IBM, antecessor ao *WebSphere*) e o surgimento de novas ferramentas comerciais que ajudavam a criar sites dinâmicas. Abrindo o código de núcleo do *WebSphere* não se sabe ao certo se a IBM pretendia aumentar a

familiaridade dos desenvolvedores com seu produto ou possuía a ideia de permitir uma maior colaboração entre seus usuários. A verdade é que o Eclipse tornou-se um verdadeiro IDE Java e levou a criação de uma organização e uma cadeia de ferramentas suportada por uma grande quantidade de empresas. (BLEWITT, 2011)

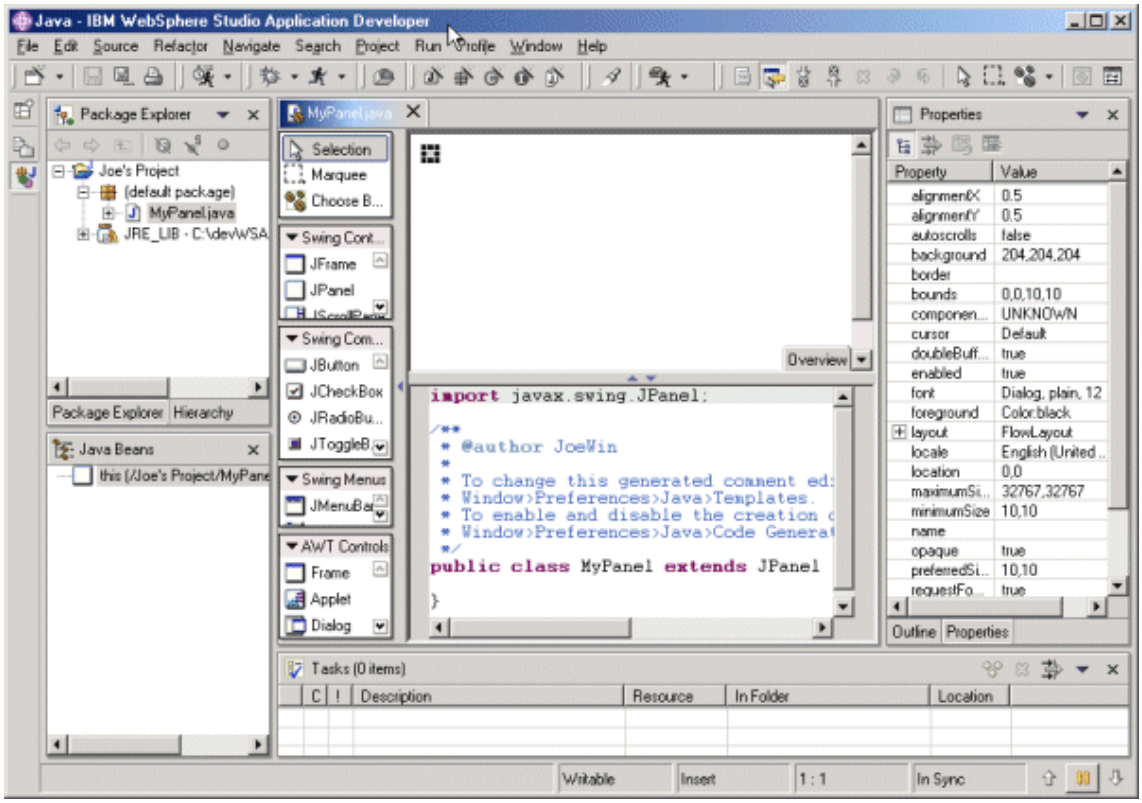

Figura 7: WebSphere Studio.
Fonte: WINCHESTER, 2002

2.3.1.1 CONSÓRCIO E FUNDAÇÃO

Fundado em 29 de novembro de 2001, o Consórcio Eclipse era formado inicialmente por Borland, IBM, Merant, QNX, Rational, RedHat, SuSe e TogheterSoft. Somente no primeiro mês, possuindo o apoio de grandes e empresas e tradicionais desenvolvedores Java, o Eclipse atingiu uma média de 4 mil *downloads* por dia. (BLEWITT, 2011)

Já em 2 de fevereiro de 2004 foi lançada a Fundação Eclipse, que trouxe uma corporação sem fins lucrativos, com um corpo de diretores e um ecossistema independentes, para suportar parceiros estratégicos e fornecedores de extensões/*plugins*. (BLEWITT, 2011)

2.3.1.2 RELEASE TRAINS

São os lançamentos consolidados e sincronizados de vários subprojetos do Eclipe, iniciados em 2006 com o Eclipse Callisto. Antes disso era necessário que os *plugins* fossem baixados um a um. (BLEWITT, 2011)

Inicialmente os lançamentos recebiam os nomes dos satélites de Jupiter, e a partir de 2009 começaram a seguir a ordem alfabética. (BLEWITT, 2011)

Segundo a organização Eclipse, estas são suas versões oficiais com suas respectivas datas de lançamento:

a) Eclipse 1.0 - 7 de novembro de 2001 (Win32/Linux32 Motif)
b) Eclipse 2.0 - 27 de junho de 2002 (Linux32 Motif + GTK, e Solaris/QNX/AIX)
c) Eclipse 2.1 - 27 de março de 2003 (primeira versão para OSX)
d) Eclipse 3.0 - 25 de junho de 2004 (primeira versão OSGi)
e) Eclipse 3.1 - 27 de junho de 2005
f) Eclipse 3.2 - 29 de junho de 2006 (Callisto)
g) Eclipse 3.3 - 25 de junho de 2007 (Europa)
h) Eclipse 3.4 - 17 de junho de 2008 (Ganymede)
i) Eclipse 3.5 - 11 de junho de 2009 (Galileo)
j) Eclipse 3.6 - 8 de junho de 2010 (Helios)
k) Eclipse 3.7 - 22 de junho de 2011 (Indigo)
l) Eclipse 4.2 – 27 de junho de 2012 (Juno)
m) Eclipse 4.3.2 - 26 de junho de 2013 (Kepler)
n) Eclipse 4.4.2 – 25 de junho de 2014 (Luna)
o) Eclipse 4.5 – 24 de junho de 2015 (Mars)

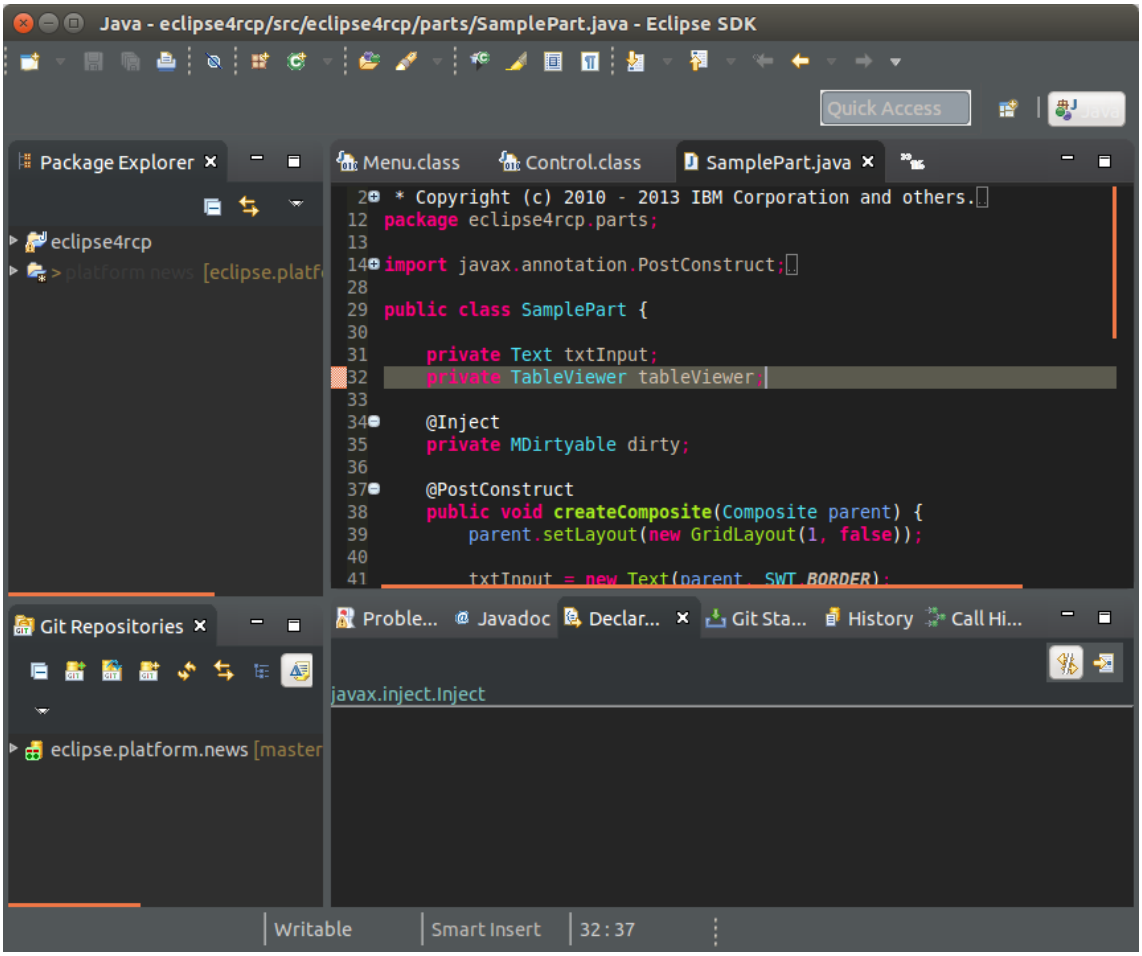

Figura 8: Eclipse Luna
Fonte: BULL, 2014.

2.3.2 NETBEANS

Em 1996, na República Tcheca, surge um projeto estudantil denominado Xelfi que tinha por objetivo escrever um IDE Java similar ao Delphi. Este foi o primeiro IDE Java escrito em Java. Quando este grupo de estudantes concluíram a graduação decidiram que poderiam comercializar o Xelfi, foi assim que foram contratados por Roman Stanek um grande empreendedor que havia se envolvido em diversas empresas que estavam surgindo na República Tcheca. (NETBEANS)

No ano de 1999 o projeto já estava bastante evoluído, com o nome de NetBeans Developer X2, nome este que surgiu da ideia de reutilização de componentes que era a base do Java. Nessa época a empresa Sun Microsystems havia desistido de sua IDE Java *Workshop* e, procurando por novas iniciativas, adquiriu o projeto NetBeans DeveloperX2 incorporando-o a sua linha de softwares. Por alguns meses a Sun mudou o nome do projeto para *Forte for Java* e o manteve por um bom tempo como software proprietário, porém, em junho de 2000 a

Sun disponibilizou o código fonte do IDE NetBeans tornando-o uma plataforma *OpenSource*. (JACOME, 2013)

2.3.2.1 RELAESES

Segundo o próprio site do NetBeans (2015), estas são as suas versões e suas respectivas datas de lançamento:

a) NetBeans 3.5 – Junho de 2003
b) NetBeans 3.6 – Abril de 2004
c) NetBeans 4.0 – Dezembro de 2004
d) NetBeans 4.1 – Maio de 2005
e) NetBeans 5.0 – Janeiro de 2006
f) NetBeans 5.5 – 30 de Outubro de 2006
g) NetBeans 5.5.1 – 24 de Maio de 2007
h) NetBeans 6.0 – 3 de Dezembro de 2007
i) NetBeans 6.1 – 28 de Abril de 2008
j) NetBeans 6.5 – 20 de Novembro de 2008
k) NetBeans 6.5.1 – 16 de Março de 2009
l) NetBeans 6.7 – 29 de Junho de 2009
m) NetBeans 6.7.1 – 27 de Julho de 2009
n) NetBeans 6.8 – 10 de Dezembro de 2009
o) NetBeans 6.9 – 15 de Junho de 2010
p) NetBeans 6.9.1 – 4 de Agosto de 2010
q) NetBeans 7.0 – 19 de Abril de 2011
r) NetBeans 7.0.1 – 1 de Agosto de 2011
s) NetBeans 7.1 – 5 de Janeiro de 2012
t) NetBeans 7.1.1 – 29 de Fevereiro de 2012
u) NetBeans 7.1.2 – 26 de Abril de 2012
v) NetBeans 7.2 – 24 de Julho de 2012
w) NetBeans 7.2.1 – 22 de Outubro de 2012
x) NetBeans 7.3 – 21 de Fevereiro de 2013
y) NetBeans 7.3.1 – 12 de Junho de 2013
z) NetBeans 7.4 – 15 de Outubro de 2013
aa) NetBeans 8.0 – 18 de Março de 2014

bb) NetBeans 8.0.1 – 9 de Setembro de 2014

cc) NetBeans 8.0.2 – 28 de Novembro de 2014

dd) NetBeans 8.1 – Versão futura para lançamento de Outubro de 2015

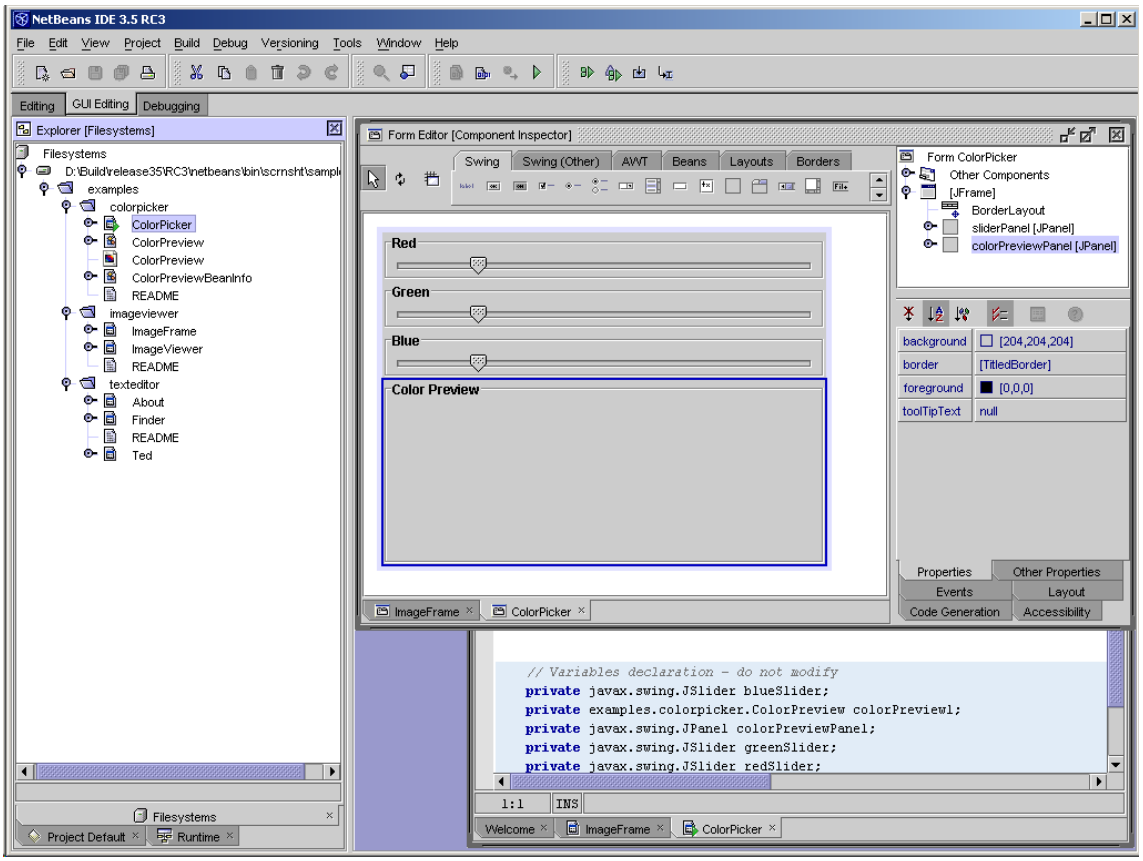

Figura 9: NetBeans 3.5
Fonte: NetBeans Web Site, 2015

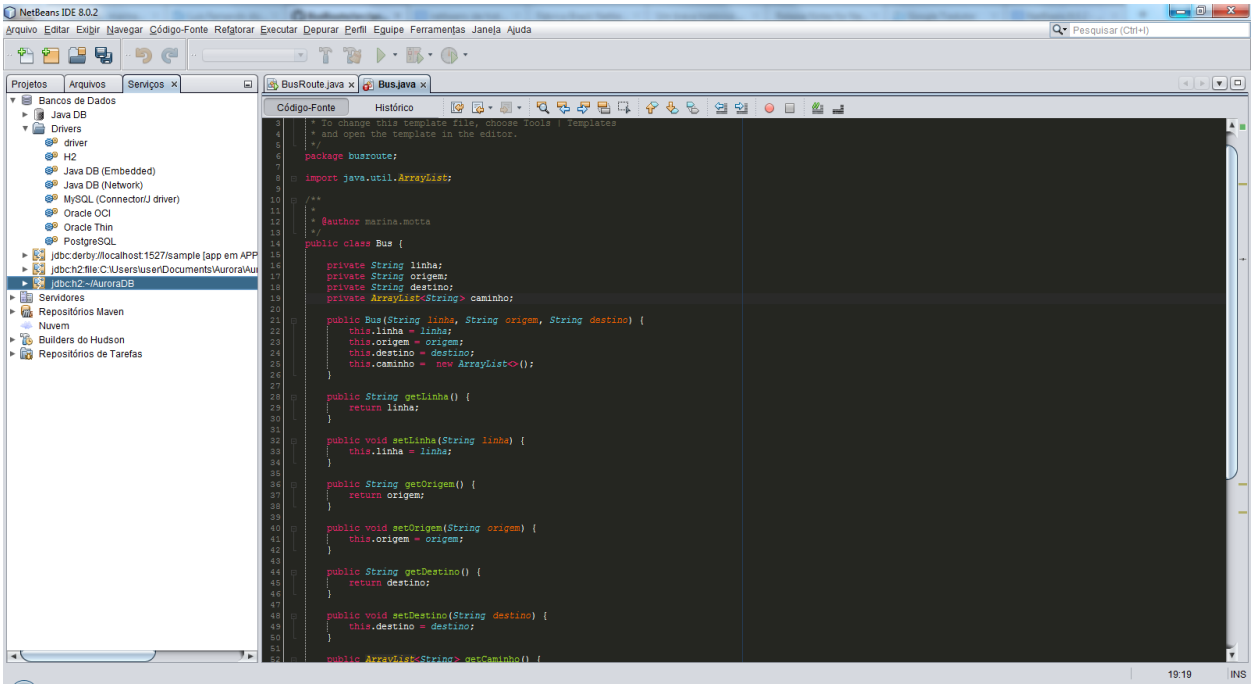

Figura 10: NetBeans 8.0.2
Fonte: O próprio autor

2.3.3 JETBRAINS INTELLIJ IDEA

A JetBrains, empresa fundadora do IntelliJ IDEA foi fundada em fevereiro de 2000 em Praga na República Tcheca. E em janeiro de 2001 foi lançada a primeira versão do IntelliJ IDEA, uma IDE para desenvolvimento Java sendo na época uma das principais IDE com recursos avançados em navegação e refatoração de códigos integrados. (FINSLAB, 2014)

Segundo informações fornecidas pela própria JetBrains (2015), estas são as versões do IntelliJ IDEA e suas respectivas datas de lançamento:

a) IntelliJ IDEA 1.0 – Janeiro de 2001
b) IntelliJ IDEA 2.0 – Julho de 2001
c) IntelliJ IDEA 3.0 – Novembro de 2002
d) IntelliJ IDEA 4.0 – 17 de Fevereiro de 2004
e) IntelliJ IDEA 4.5 – 27 de Julho de 2004
f) IntelliJ IDEA 5.0 – 01 de Agosto de 2005
g) IntelliJ IDEA 6.0 - 02 de Outubro de 2006
h) IntelliJ IDEA 7.0 – 15 de Outubro de 2007
i) IntelliJ IDEA 8.0 – 06 de Novembro de 2008

j) IntelliJ IDEA 9.0 – 08 de Dezembro de 2009 (Primeira versão de código aberto)
k) IntelliJ IDEA 10 – 2010
l) IntelliJ IDEA 10.5 – 16 de Maio de 2011
m) IntelliJ IDEA 11 – 06 de Dezembro de 2011
n) IntelliJ IDEA 11.1 – 28 de Março de 2012
o) IntelliJ IDEA 12 – 05 de Dezembro de 2012
p) IntelliJ IDEA 12.1 – 03 de Abril de 2013
q) IntelliJ IDEA 13 – 03 de Dezembro de 2013
r) IntelliJ IDEA 14 – 05 de Novembro de 2014

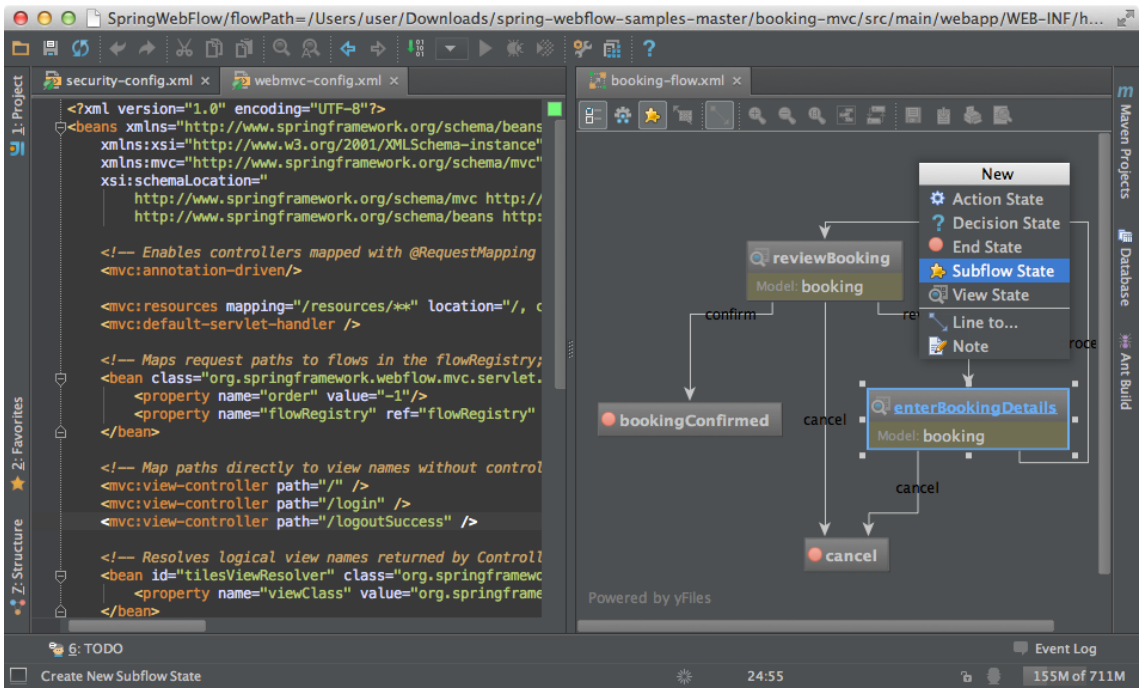

Figura 11: IntelliJ IDEA 12
Fonte: Cheptsov, 2012

2.3.4 ANDROID STUDIO

Android Studio é a IDE oficial da Google para o desenvolvimento de aplicações Android e foi baseado no IntelliJ IDEA da JetBrains. (DEVELOPERS, 2015). Sua primeira versão estável, Android Studio 1.0, foi anunciada em 2013 durante a Google I/O. Dentre as

principais funções da IDE pode-se citar a edição inteligente de código, recursos de design de interface de usuário e análise de performance. (ROCHA, 2014)

> Google I/O é uma conferência voltada principalmente para os desenvolvedores. Por isso, não existe lugar melhor para a apresentação do Android Studio, a nova interface de produção de aplicativos para o sistema operacional que promete melhorar o trabalho de quem ganha a vida construindo softwares para celulares e tablets. (DEMARTINI, pág. 01, 2013)

2.3.4.1 ANDROID STUDIO X ECLIPSE + ADT (ANDROID DEVELOPER TOOLS)

Segundo Carvalho (2013) ambos possuem o mesmo objetivo e contam com um ambiente de desenvolvimento, debug, testes e profile multiplataforma para Android. É possível trabalhar no Android Studio com projetos que foram criados no Eclipse, instalando uma atualização do ADT (versão 22.2.1), na qual está disponível uma nova funcionalidade que exporta o projeto para o formato de projeto Gradle.

> O Gradle é um sistema avançado de automatização de builds que une o melhor da flexibilidade do Ant com o gerenciamento de dependencias e as convenções do Maven. Os arquivos de build do Gradle são scripts escritos na linguagem Groovy, ao contrário dos formatos de construção do Ant e Maven que usam arquivos XML para a configuração. Por serem baseados em scripts, os arquivos do Gradle permitem que você realize tarefas de programação em seu arquivo de configuração. O Gradle ainda conta com um sistema de plugins que adicionam funcionalidades extras ao seu core. (LISBOA, pág 01, 2015)

2.3.4.1.1 VANTAGENS

De acordo com Carvalho (2013), estas são as principais vantagens do Android Studio em relação ao Eclipse:

a) Interface – O Android Studio possui uma interface muito mais atraente que o Eclipse e conta com diversos "*Look And Feel*" diferentes, além da possibilidade de customização das teclas de atalho do teclado;
b) Auto-Complete – Ele é automático, não sendo necessário apertar "Ctrl + Espaço" para que ele apareça;
c) Injection Language – Permite que *strings* de outras linguagens sejam validadas pela IDE;
d) Controle de Versão – O Android Studio se integra com os seguintes sistemas de controle de versão: Mercurial, Git e Subversion;

e) Preview de *Layouts* – Extremamente superior ao Eclipse, é possível visualizar o layout em várias telas de tamanhos diferentes simultaneamente, ainda é possível selecionar idioma, tema, versão do Android e resoluções para validar arquivos e recursos visuais;

f) Criação de Layout – Também é superior ao do Eclipse, conta com o recurso "*drag and drop*", famoso arrastar e soltar.

2.3.4.1.2 DESVANTAGENS

Carvalho (2013) ainda afirma as desvantagens que podem ser encontradas por desenvolvedores Android que utilizam a IDE Eclipse e agora migram para o Android Studio:

a) Estrutura de Projeto – Para quem está acostumado com o Eclipse, a estrutura de organização do projeto no Android Studio é bem diferente por utilizar a estrutura *Grandle*;

b) Múltiplos Projetos – Diferente do Eclipse, não é possível abrir mais de um projeto em uma mesma janela.

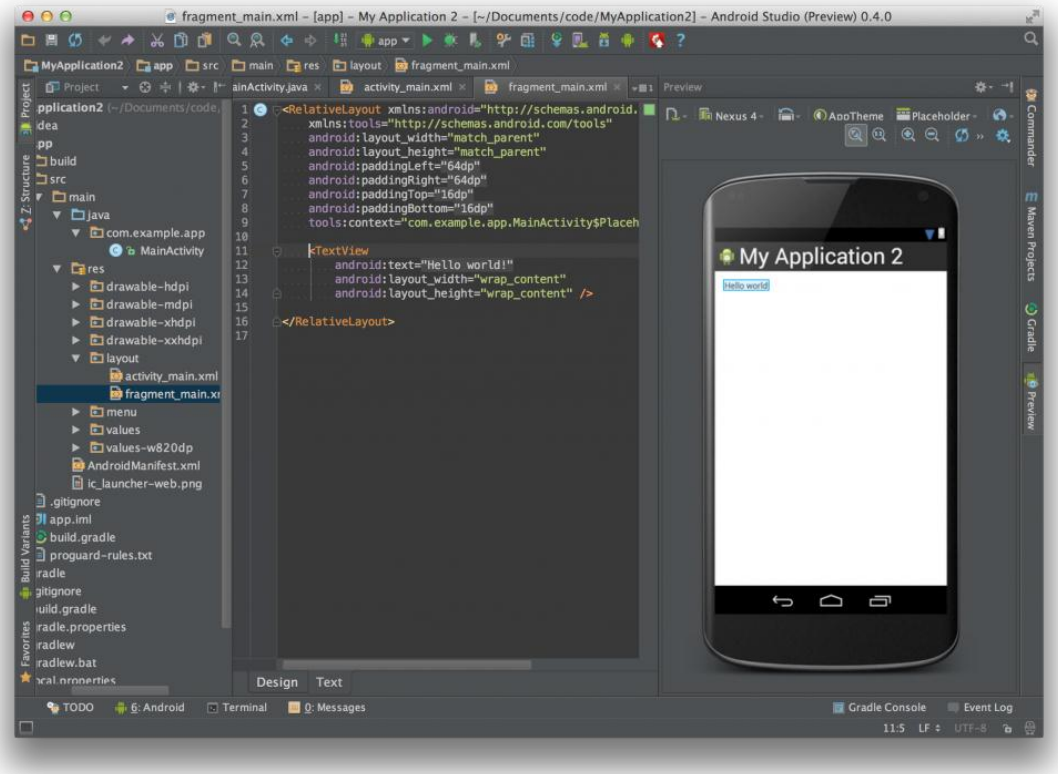

Figura 12: Android Studio IDE
Fonte: Donahue, 2014

2.4 PLATAFORMA DE DESENVOLVIMENTO ANDROID

2.4.1 SISTEMA OPERACIONAL ANDROID

2.4.1.1 CONTEXTO HISTÓRICO

Com o objetivo de desenvolver uma plataforma de celulares baseada no *kernel* do Linux, em 2003, Andy Rubin cria a startup *Android Inc*, que em 2005 seria comprada pela Google. (CARVALHO, 2014)

No ano de 2007 é criada a OHA (*Open Handset Alliance*), um grupo formado por empresas líderes mundiais em tecnologia para dispositivos móveis. Com o objetivo de criar uma plataforma livre e simples, na qual um desenvolvedor com conhecimento em linguagem Java pode tranquilamente desenvolver suas aplicações e os fabricantes de celulares que possuam hardware para suportar a plataforma podem utilizar o sistema e desfrutar de suas vantagens. (DIMARZIO, 2008)

2.4.1.2 VERSÕES

a) 2008/Setembro – Android 1.0 – Sua primeira versão comercial, já possuía alguns aplicativos como Media Player, navegador, Bluetooth e suporte a Wi-Fi, inovadores para época. (SANTINO,2013)
b) 2009/Fevereiro – Android 1.1 – A primeira atualização. Entre as novidades, estão o detalhamento e exibição de reviews de locais quando o usuário faria uma busca no Maps e melhorias na interface para realizar chamadas. (SANTINO,2013)
c) 2009/Abril – Android 1.5 – *Cupcake* – A primeira versão a receber um apelido de sobremesa, que logo tornar-se-ia um padrão da empresa. Sua principal inovação foi a inclusão de *Widgets*, que são muito utilizados até hoje, e é uma marca registrada da empresa. (SANTINO,2013)
d) 2009/Setembro – Android 1.6 – *Donut* - Trouxe suporte à resolução 800x480 e a inclusão de uma caixa de buscas já na tela inicial, para facilitar pesquisas internas e na web. Também teve melhorias em acessibilidade e a inclusão de um sistema de síntese de voz. Também trouxe mais facilidade de uso para o Google Play, possibilitando a inclusão de *screenshots* de aplicativos. (SANTINO,2013)

e) 2009/Outubro – Android 2.0 e 2.1 – *Eclair* - Trouxe uma nova interface, velocidade de *hardware* otimizada e suporte ao HTML5 no navegador. O sistema ainda apresentou a possibilidade de inclusão de várias contas no aparelho, para sincronização de contatos de várias fontes diferentes, além de trazer suporte ao protocolo de e-mail Microsoft Exchange. (SANTINO,2013)
f) 2010/Maio – Android 2.2 a 2.2.3 – *Froyo* – A principal novidade eram aplicações que rodavam quase que invisíveis para o usuário, tais como: otimização de memória, velocidade e desempenho. Também trouxe a possibilidade de transformar o celular em um *hotspot* de Wi-Fi e instalação de aplicativos em cartões de memória removíveis. (SANTINO,2013)
g) 2010/Dezembro – Android 2.3 a 2.3.7 – *Gingerbread* – Trouxe suporte à resolução HD (*high defintion*) e tecnologia NFC (*Near Field Communication*). Também passou a ter suporte nativo a sensores como barômetro e giroscópio e a aceitar múltiplas câmeras em um mesmo dispositivo. Assim, as câmeras frontais passam a se popularizar. (SANTINO,2013)
h) 2011/Fevereiro – Android 3.0 a 3.2 – *Honeycomb* – Foi o único sistema operacional desenvolvido para *tablets*. Ele trouxe melhorias de câmera e simplificação de multitarefas e suporte a processadores com múltiplos núcleos. A navegação na internet também foi melhorada, com a novidade do modo incógnito. O sistema também passou a permitir a encriptação de todos os dados do usuário. (SANTINO,2013)
i) 2011/Outubro – Android 4.0 a 4.0.4 – *Ice Cream* – Esta versão trouxe os botões virtuais (como nos *tablets*), acabando com a necessidade de teclas físicas nos aparelhos. Inclui a possibilidade de acessar aplicativos da tela de bloqueio e desbloqueio por reconhecimento facial. O navegador *Chrome* passou a aceitar navegação com abas (até 16) e o sistema também passou a possuir o editor de fotos nativo. (SANTINO,2013)
j) 2012/Julho – Android 4.1 a 4.2.2 – *Jelly Bean* – Inclui a tecnologia *Photo Sphere* para produção de imagens em 360º e trouxe a possibilidade de realizar gestos na tela de bloqueio para acessar rapidamente a câmera do celular. (SANTINO,2013)
k) 2013/Setembro – Android 4.4 – *KitKat* – Essa versão tem como sua principal novidade a otimização de recursos. Possui recursos de chamadas inteligentes

que cruzam dados da agenda, locais próximos e contas que estão adicionadas no aparelho. (JORDÃO,2013)

l) 2014/Novembro – Android 5.0 – *Lollipop* - Uma das grandes novidades é a possibilidade de fixação da tela. Nesta nova versão do Android também possível ter múltiplos usuários no celular. Possui uma versão do *Google Now* mais inteligente, é possível ativá-lo de qualquer tela do aparelho, exceto a de bloqueio, e as respostas estão mais rápidas e precisas. (RINALDI, 2014)

A seguir temos uma tabela e um gráfico comparativos das versões do Android que possuíram um maior número de distribuição.

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	6.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.9%
4.1.x	Jelly Bean	16	17.3%
4.2.x		17	19.4%
4.3		18	5.9%
4.4	KitKat	19	40.9%
5.0	Lollipop	21	3.3%

Tabela 2: *Comparativo entre as versões do Android.*
Fonte: Android Developers, 2015

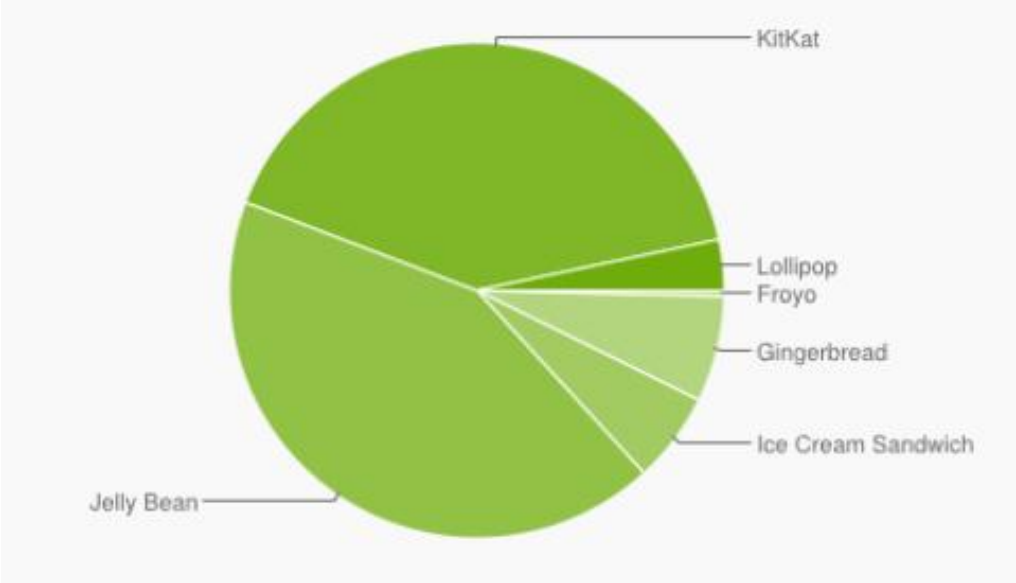

Figura 13: Gráfico Comparativo entre as versões do Android
Fonte: Android Developers, 2015

2.4.2 CARACTERÍSTICAS

Fonseca (2013) cita as seguintes características do Android como sendo as principais:

a) *Application Framework* – *Framework* responsável por permitir o reuso e substituição dos componentes;
b) Máquina Virtual Dalvik – Uma máquina virtual otimizada para dispositivos móveis;
c) Navegador Integrado – Baseado no projeto WebKit (*open source*);
d) Gráficos Otimizados – Bibliotecas gráficas 2D e gráficos 3D baseados na especificação *OpenGL* 1.0;
e) SQL Lite – Para armazenamento de dados estruturados;
f) Suporte de mídias – Áudio, vídeo, formatos de imagens (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG e GIF);
g) Mensagens – Suporta tanto SMS como MMS;
h) *Connectivity* – Suporta conexões GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, A2DP, WiFi, LTE, WiMAX, AVRCP;
i) Multi-*touch* – Suporta múltiplos toques na tela;
j) Multi-*tasking* – Suporta a execução de múltiplas tarefas;

2.4.2.1 MÁQUINA VIRTUAL DALVIK

Assim como Java necessita de uma máquina virtual para traduzir seus bytecodes em código binário, o Android (por utilizar a linguagem de programação Java) também necessita de uma máquina virtual.

O Dalvik foi criado por Dan Bornstein que posteriormente juntou-se à equipe da Google para trabalhar no projeto Android. A máquina virtual Dalvik possui então o mesmo objetivo da uma JVM já descritas neste trabalho: realizar a comunicação entre a aplicação e o *hardware* para que a aplicação seja capaz de realizar o que ela fora destinada. Os arquivos de classes são compilados em um arquivo .dex e assim podem ser interpretados pela Dalvik VM. (SOARES, 2014)

> Informações duplicadas no código-fonte são agrupados na compilação do arquivo .dex, obtendo um arquivo mais compacto tanto no tamanho quanto em sua performance. Os arquivos executáveis ainda podem ser alterados quando você instala um aplicativo, otimizando ainda mais a execução no ambiente mobile. Gerenciamento de memória, estrutura de dados e bibliotecas de funções em linha tornam os arquivos menores para execução em nossos dispositivos. (SOARES, pág. 01, 2014)

De acordo com Soares (2013), o Dalvik possui código fonte aberto (*open source*), com origem no projeto *Apache Harmony* (espécie de JVM com código fonte aberto (BLEWITT, 2011)), possui melhor desempenho que uma JVM teria em dispositivos com hardware limitado, e foi criado para que seja capaz de executar múltiplas instâncias da máquina virtual simultaneamente, possibilitando assim o recurso multi-*tasking*.

Podemos então nos perguntar qual seria o motivo de a Google não utilizar uma máquina virtual do J2ME (*Micro Edition*), já que ambos são utilizados em dispositivos móveis com opções limitadas de *hardware*? Soares (2013) nos explica o motivo, e ele é muito simples: por possuir código fonte aberto, não é necessário pagar a licença para sua utilização, ao contrário das máquinas virtuais utilizadas no J2ME.

Entretanto, para evitar possíveis transtornos com a Oracle – empresa que comprou a Sun a qual desenvolveu o projeto *Apache Harmony* – a partir da versão 4.4 KitKat do Android, a máquina virtual Dalvik não será mais utilizada, e sim a ART, desenvolvida totalmente pela Google (SOARES, 2013), a qual será explanada no próximo item.

2.4.2.2 ART (ANDROID RUN TIME)

O ART foi criado para utilizar uma técnica de compilação denominada AOT (*Ahead Of Time*), e diferentemente do Dalvik ela ocorre antes da execução do aplicativo e não durante (*Just In Time*). Assim, é esperado que a velocidade de execução aumente em até 2 vezes em relação ao Dalvik. (SOUZA, 2014)

A seguir temos a figura de um gráfico de um algoritmo de ordenação – *Quick Sort* – executado em dispositivo que utiliza Android com ART (linha vemelha), Android com Dalvik (linha azul) e Java *Native Interface* (linha laranja) – técnica utilizada para que um aplicativo Java acesse aplicações e bibliotecas escritas em linguagens de baixo nível, tais como C/C++, Assembly.

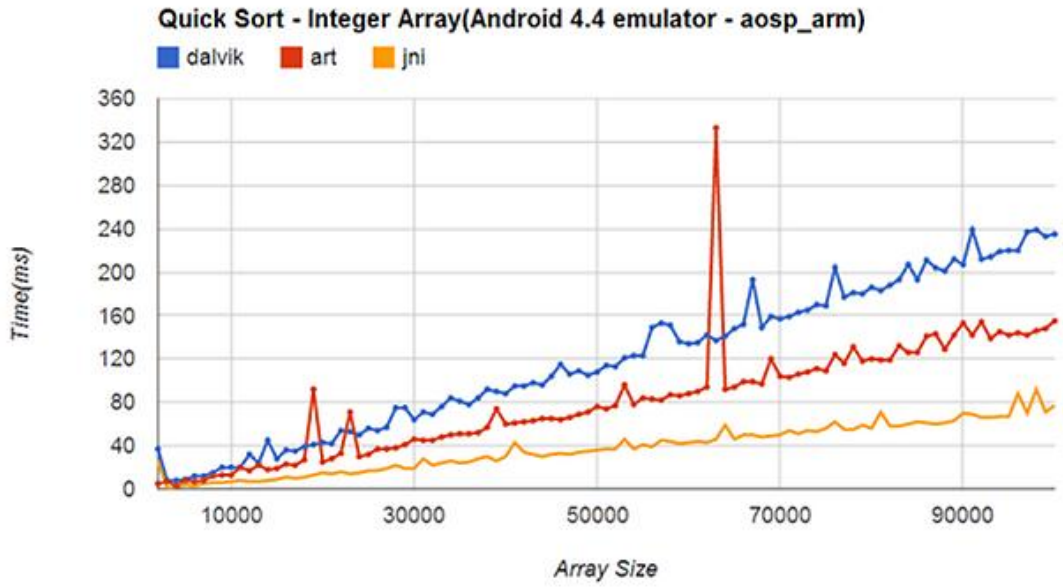

Figura 14: Comparativo entre Máquinas Virtuais
Fonte: Souza, 2014

2.4.2.3 VANTAGENS E DESVANTAGENS DAS MÁQUINAS VIRTUAIS

Souza (2014) cita as vantagens e suas consequentes desvantagens da nova tecnologia da Google. A execução dos aplicativos ficará mais rápida, porém eles ocuparão mais memória, por serem compilados no momento em que o aplicativo é instalado, este permanece na memória para uma inicialização e execução mais rápida. Outra vantagem é a economia de bateria, pois os aplicativos já estão pré-compilados e o processador precisa fazer menos esforço cada vez que um aplicativo é iniciado, entretanto ao iniciar o dispositivo pela primeira

vez ele levará um tempo maior, assim como quando um aplicativo for iniciado pela primeira vez.

2.4.3 ARQUITETURA DO ANDROID

A própria Google refere-se ao sistema operacional Android como uma pilha de *softwares*, cada camada dessa pilha possui diversos programas com funções específicas do SO. (STRICKLAND, 2014)

A seguir serão explanadas cada camada dessa pilha de acordo com os conhecimentos de Strickland (2014).

O *kernel* dessa pilha foi baseado na versão 2.6 do Linux, incluindo os programas de gerenciamento de memória, configurações de segurança, gerenciamento de energia e diversos *drivers* de *hardware*.

A bibliotecas são conjuntos de instruções que dizem ao dispositivo como lidar com cada tipo de dado. Existem bibliotecas de mídia, de aceleração tridimensional, de navegadores Web, entre tantas outras.

No mesmo nível das bibliotecas está a camada de tempo de execução (máquina virtual).

Logo após tem-se o *framework* de aplicação, programas que gerenciam funções básicas do dispositivo, como mudança entre processos e programas, localização física do dispositivo, alocação de recursos, aplicações do dispositivo. “Os desenvolvedores de aplicações têm acesso total ao *framework* de aplicações do Android. Isso possibilita que eles tirem vantagem das capacidades de processamento do Android e suportem recursos quando estão construindo uma aplicação Android.” (STRICKLAND, 2014).

E no topo da pilha temos as aplicações propriamente ditas, onde estão as funções básicas do dispositivo, tais como acessar a Internet, enviar SMS, realizar chamadas telefônicas.

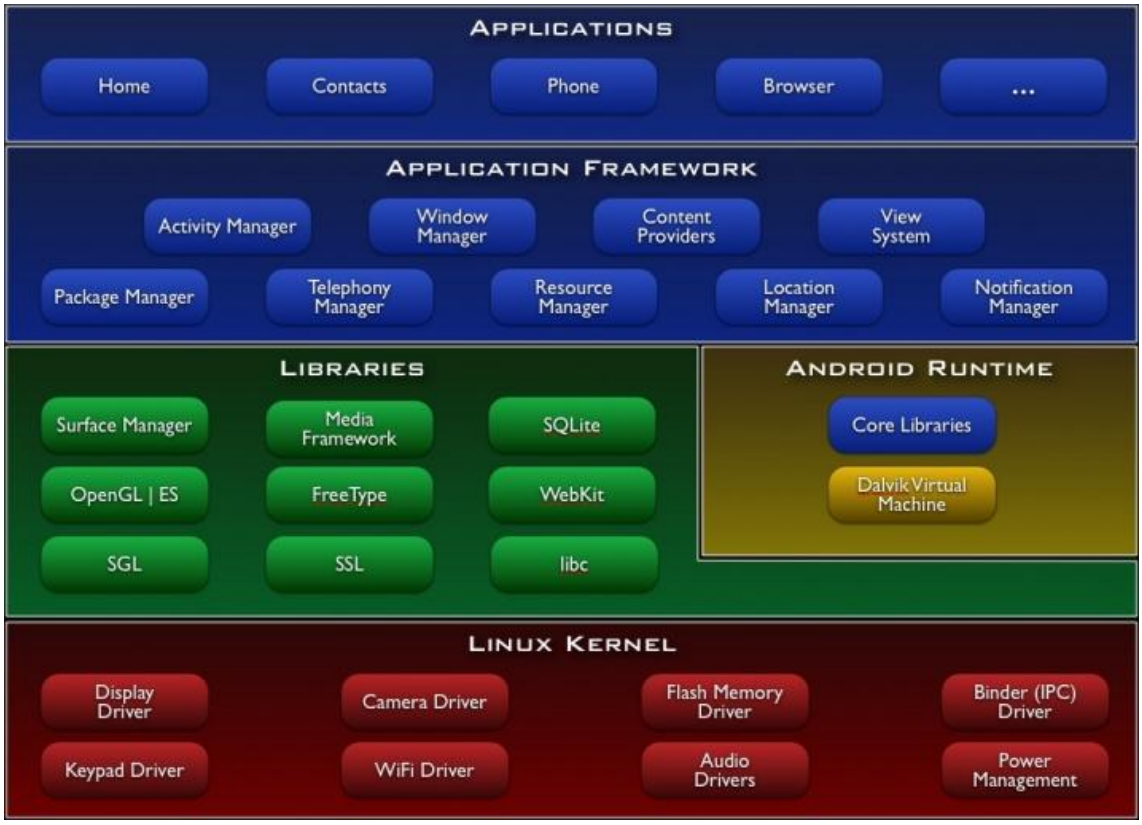

Figura 15: Arquitetura Android
Fonte: Morandi e Rodolpho, 2011

2.4.4 ANDROID MANIFEST

Macedo (2011) afirma que o androidmanifest.xml é o arquivo principal de uma aplicação Android. Nele estão inseridas todas as configurações daquela aplicação, entre elas o nome do pacote utilizado, o nome das classes de cada *activity* entre tantas outras.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tapittest_libtest"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="4"
        android:targetSdkVersion="15" />

    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>

    <!-- Optional permissions to enable ad geotargeting
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
    -->

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.tapit.adview.AdActivity"
            android:configChanges="keyboard|keyboardHidden|orientation" />
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figura 16: Android Manifest
Fonte: Bui, 2012

> Dentro da tag <manifest> é necessário declarar o pacote principal do projeto, utilizando a tag <package>. É obrigatório que cada activity do projeto esteja declarada no arquivo AndroidManifest.xml, caso contrário não é possível utiliza-la. Para declarar a activity é utilizada a tag <activity>, que recebe o nome da classe, e é sempre relativa ao pacote principal. (MACEDO, pág. 01, 2011)

2.4.5 ELEMENTOS DE UMA APLICAÇÃO ANDROID

2.4.5.1 ACTIVITY

"Uma *Activity* é basicamente uma classe gerenciadora de UI (Interface com o usuário). Todo aplicativo Android começa por uma *Activity*." (SILVEIRA, 2010) Sempre que uma aplicação é iniciada, na verdade sua *activity* principal é executada. A seguir contamos com a figura de um diagrama exemplificando o ciclo de vida de uma *activity*:

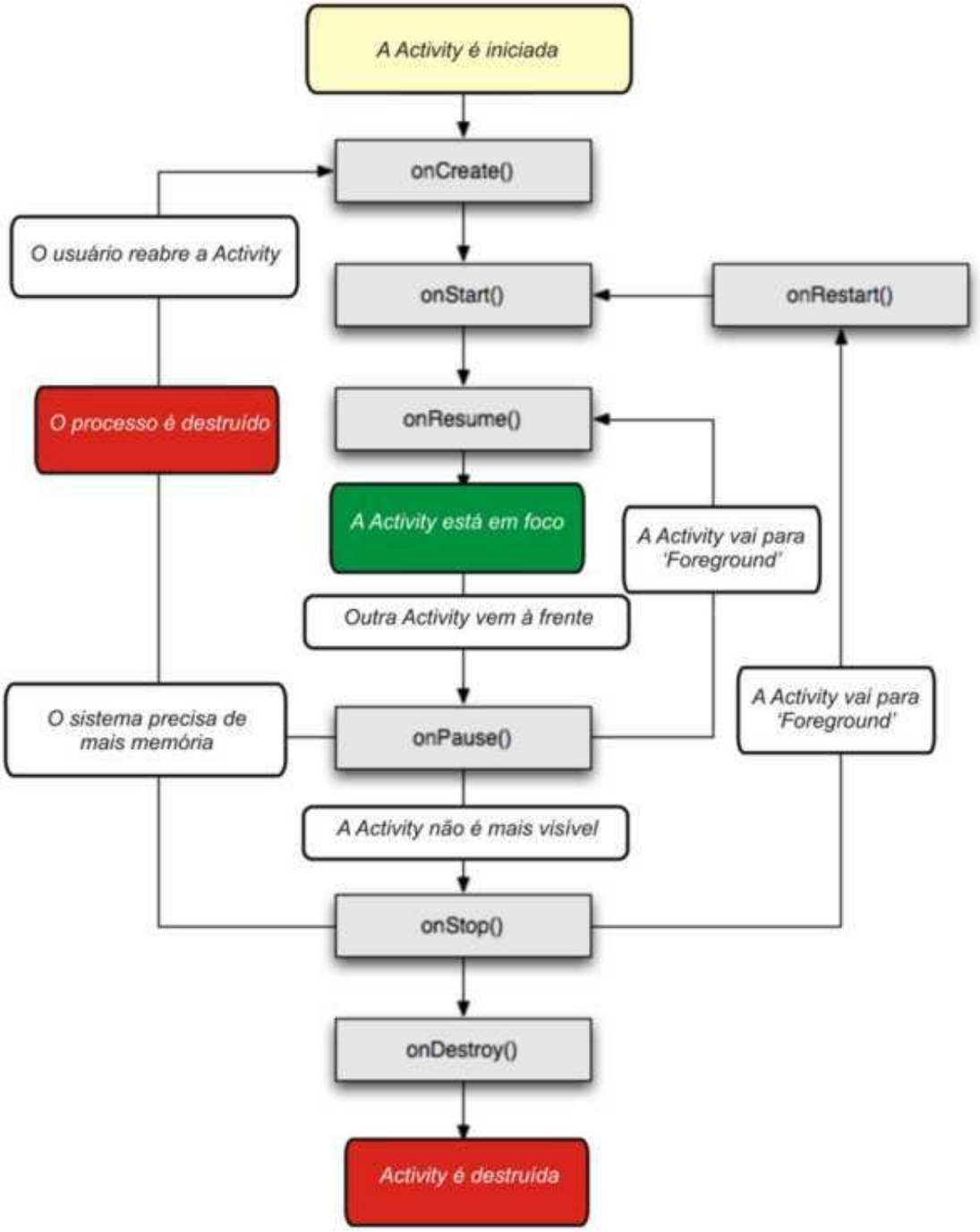

Figura 17: Ciclo de vida de uma Activity
Fonte: Silveira, 2010

2.4.5.2 INTENT

Monteiro (2013) afirma que *Intents* são basicamente intenções do usuário de realizar algo. Elas podem ser definidas como mensagens enviadas de um componente da aplicação (por exemplo uma *activity*) para o Android, informando a intenção de inicializar outro

componente ou encerrar a aplicação. Através deste recurso que é possível fazer com que as aplicações interajam entre si.

> Através de Intents é possível iniciar novas activities, como fazer uma busca e selecionar um contato do telefone, abrir a aplicação de mapas com as coordenadas de localização do GPS, abrir uma página da web, tirar fotos utilizando a câmera e etc, apenas reaproveitando funcionalidades já existentes, disponibilizadas pelos aplicativos instalados no aparelho. (MONTEIRO, pág. 01, 2013)

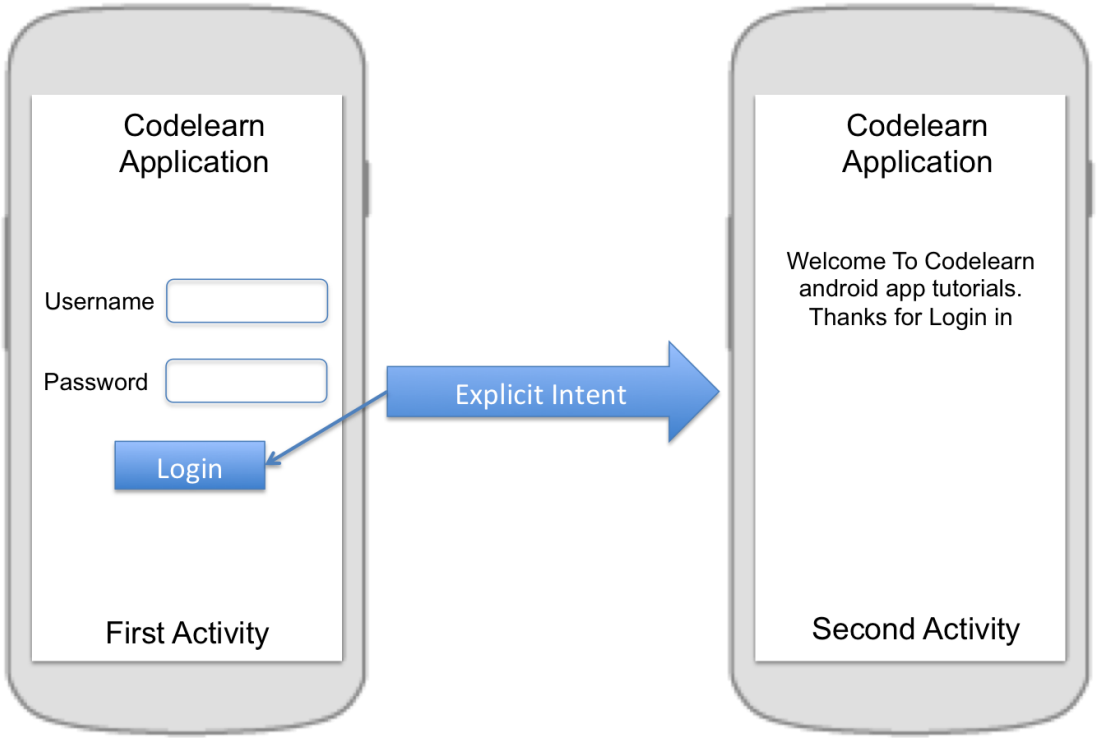

Figura 18: Intent
Fonte: Code Learn, 2014

De acordo com Monteiro (2013), a estrutura básica de uma *intent* contém: nome do componente, ação, dados, informações extras e uma categoria.

a) Nome do Componente – É definido pelo nome da classe e do pacote que estão no AndroidManifest.xml, que representa o componente que deve ser carregado;
b) Ação – é uma string que define qual ação deve ser realizada; Exemplos:
   1. ACTION_CALL – indica que uma chamada telefônica deve ser realizada
   2. ACTION_VIEW – indica que algum dado deve ser exibido para o usuário
   3. ACTION_EDIT – indica que se deseja editar alguma informação
   4. ACTION_SENDTO – indica que se deseja enviar alguma informação

c) Dados - uma *Intent* pode receber uma *Uri* que determina os dados sobre os quais a ação deve ser realizada;
d) Informações Extras - qualquer outro dado necessário para que o componente execute a ação pode ser informado através dos "extras" da *Intent*;
e) Categoria – é uma *string* e serve como informação adicional para que o Android qual componente é o mais adequado para receber a *Intent*.

2.4.5.3 INTENT FILTER

Intent filter é responsável por especificar os tipos de intenções que uma atividade, serviço ou receptor de transmissão pode responder. Um *intent filter* declara as capacidades do seu componente pai - o que uma atividade ou serviço pode fazer e quais os tipos de transmissões de um receptor pode manipular. Ele abre o componente para receber *intents* do tipo anunciado, enquanto filtra os que não são significativas para o componente.

A maior parte do conteúdo do filtro são descritos por sua <action>, <category> e <data> sub-elementos.

```
<Android intent-filter: icon = "recurso drawable"
              android: etiqueta = "recurso de cadeia"
              android: prioridade = "inteiro">
    . . .
</ Intent-filter>
```

Figura 19: Sintaxe Intent Filter
Fonte: Android Developer, 2015

2.4.5.4 BROADCAST RECEIVER

Sabe-se que é possível utilizar *Intents* para comunicar-se dentro do aplicativo ou para fazer uma requisição que outro componente irá executar. É possível também que um *Intent* seja enviado para qualquer receptor interessado, ao fazer isso não será requisitado a execução de uma tarefa específica, mas sim deixando que todos saibam que algo ocorreu. O Android envia esses *broadcasts* por diversas razões, como quando uma chamada telefônica ou mensagem de texto (SMS) é recebida. Para que a transmissão de um *Intent* seja registrada é necessário que um *BroadcastReceiver* seja implementado. (ABLESON, KING, SEM & ORTIZ, 2012)

A classe *BroadcastReceiver* fornece uma série de métodos que permitem obter e configurar um código de resultados, dados de resultado (no formato *String*) e um *Bundle* extra. Isso também define um método relativo ao ciclo de vida para ser executado quando o *Intent* apropriado é recebido. É possível associar um *BroadcastReceiver* com um *IntentFilter* no código ou no arquivo de manifesto XML. (ABLESON, KING, SEM & ORTIZ, 2012)

AÇÃO	**DESCRIÇÃO**
ACTION_BATTERY_ CHANGED	Enviado quando nível da bateria ou estado de carregamento muda
ACTION_BOOT_ COMPLETED	Enviado quando a plataforma acaba de ser inicializada
ACTION_PACKAGE_ ADDED	Enviado quando um pacote é adicionado à plataforma
ACTION_PACKAGE_ REMOVED	Enviado quando um pacote é removido da plataforma
ACTION_TIME_ CHANGED	Enviado o usuário acerta o relógio do dispositivo
ACTION_TIME_ TICK	Enviado a cada minuto para indicar que o relógio está funcionando
ACTION_TIMEZONE_ CHANGED	Enviado quando o usuário muda o fuso horário

Tabela 3: Ações de transmissão fornecidas pela plataforma Android
Fonte: Ableson, King, Sen & Ortiz, 2012

```
public class FileAlertServiceReceiver extends BroascastReceiver{
    @Override
    public void onReceive(Context context, Intent intent){
        if (intent.getAction().equals(Intent.ACTION_BOOT_COMPLETED)){
            context.startService(new Intent(context, FileAlertService.class));
        }
    }
}
```

Figura 20: Implementação de BroadcastReceiver
Fonte: O próprio autor

Quando seu próprio receptor de transmissão *Intent* é criado, como no exemplo da figura a acima, a classe *BroadcastReceiver* é extendida e implementado o método abstrato *onReceive(Context c, Intent i).* Em nosso exemplo, iniciamos o *FileAlertService*. Esta classe *service*, é iniciada utilizando o método *Context.startService(Intent i, Bundle b*).

Instâncias da classe *receiver* têm um ciclo de vida curto e focalizado. Após completar o método *onReceive(Context c, Intent i)* a instância e o processo que invocaram o receptor não são mais necessários e podem ser encerrados pelo sistema. Por esta razão, não é possível executar nenhuma operação assíncrona em um *BroadcastReceiver*, como iniciar uma *thread* ou mostrar um diálogo, em vez disso é possível iniciar um *Service*. (ABLESON, KING, SEM & ORTIZ, 2012)

2.4.5.5 SERVICE

Em aplicativo Android básico são criadas classes *Activity* e move-se de uma tela para outra usando chamadas de *Intent*. Essa abordagem funciona perfeitamente para o aplicativo Android de primeiro plano, que vai de tela em tela. Mas não funciona quando é necessário monitorar alterações de segundo plano, mesmo quando o usuário não esteja com o aplicativo aberto no momento da mudança, então é necessário implementar um *Service*. (ABLESON, KING, SEM & ORTIZ, 2012)

> As melhores práticas do Android desencorajam Services de longa duração. Services que rodam contínua e constantemente usam a rede ou realizam tarefas pesadas para a CPU vão consumir a bateria do dispositivo e podem deixar outras operações mais lentas. Pior ainda, já que rodam em segundo plano, o usuário não vai saber que aplicativos são responsáveis pelo desempenho pobre dos seus dispositivos. O sistema operacional irá, por fim, interromper Services em execução se precisar de mais memória, mas não vai interferir de outro modo com serviços mal projetados. (ABLESON, KING, SEM & ORTIZ, pág. 121, 2012)

2.4.5.6 CONTENT PROVIDER

Segundo Nakahara (2011) não existe um local comum para guardar dados que todos os pacotes Android possam acessar. Por esta razão foram criados os *Content Providers*, que armazenam e recuperam dados e os deixam disponíveis para todas as aplicações. Por padrão o Android contém alguns *Content Providers* para tipos de dados comuns, por exemplo dados de áudio, de imagens, de vídeo, informações de contatos, entre outros.

Para tornar dados da sua aplicação públicos, é possível implementar um *Content Provider* próprio extendendo a classe *ContentProvider*, ou adicionar dados em um *provider* existente, se este controlar o mesmo tipo de dados que deseja-se manipular. (NAKAHARA, 2011)

URI	OBJETIVO
content://food/ingredients/	Retorna uma lista (List) de todos os ingredientes a partir do provedor registrado para cuidar de content://food
content://food/meals/	Retorna uma lista (List) de todos os pratos a partir do provedor registrado para cuidar de content://food
content://food/meals/1	Retorna ou manipula uma única refeição com ID 1 a partir do provedor registrado para cuidar de content://food

Tabela 4: Variações de ContentProvider URI para diferentes objetivos
Fonte: Ableson, King, Sen & Ortiz, 2012

Cada classe *ContentProvider* expõe um *CONTENT_URI* único que identifica o tipo de conteúdo que vai gerenciar. Esse *URI* pode pesquisar dados de duas maneiras, singular ou plural. Para pesquisar no provedor é necessário conhecer seu *URI*.

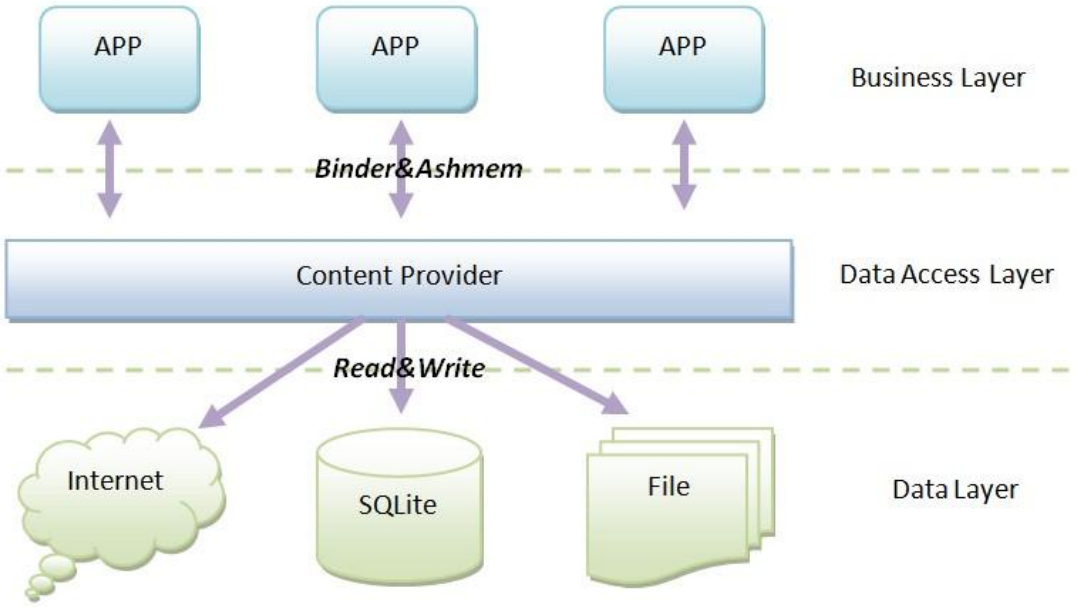

Figura 21: Funcionamento de Content Provider
Fonte: Shengyangluo, 2011

2.5 SQLITE

O SQLite é uma ferramenta que possibilita o armazenamento dos dados das aplicações em tabelas e que estes dados sejam manipulados com comandos SQL, com a diferença de que não é necessário acessar um SGBD. (GONÇALVES, 2014)

> O SQLite é um banco autocontido, compacto, com suporte nativo no Android e sem necessidade de configuração ou instalação. Isto torna-o a escolha natural para um ambiente em que devemos prezar por desempenho, disponibilidade de memória e praticidade de uso. (LUZZI, pág. 01, 2013)

Pode ser definido como uma biblioteca desenvolvida em C que pode ser integrada com aplicações desenvolvidas em diversas linguagens possibilitando a manipulação de dados através de instruções SQL. Seu funcionamento é capaz de criar um arquivo de extensão .db em disco e ler e escrever diretamente neste arquivo. (GONÇALVES, 2014)

Diferentemente de outras ferramentas que necessitam de um servidor de banco de dados, o SQLite não possui a arquitetura cliente/servidor. O motor de banco de dados inteiro está integrado a qualquer aplicativo que necessite acessar os dados. Ao contrário de um SGDB tradicional que exige funções multitarefas e avançados processos de comunicação de alto desempenho, o SQLite exige pouco mais do que a capacidade de ler e escrever dados em um arquivo. (KREIBICH, 2010)

As figuras a seguir demonstram essas diferenças:

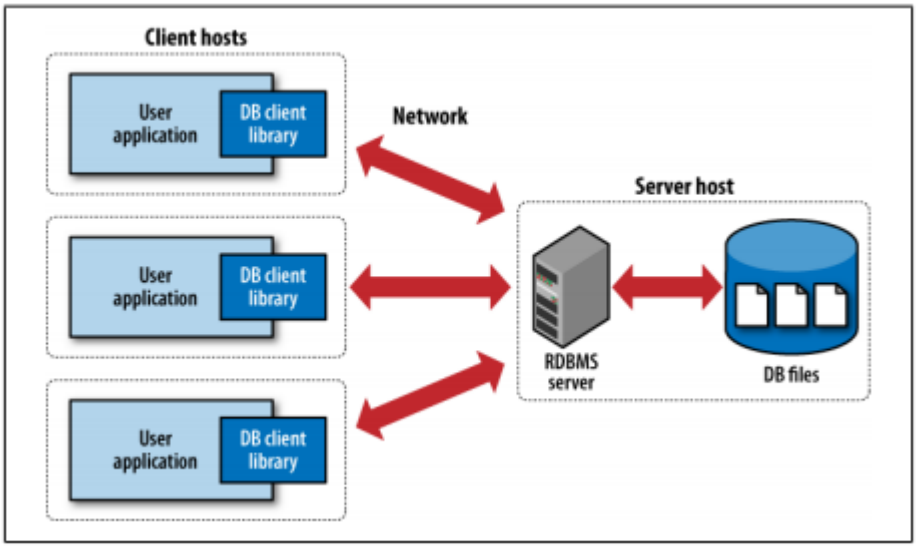

Figura 22: Funcionamento de um SGDB
Fonte: Kreibich, 2010

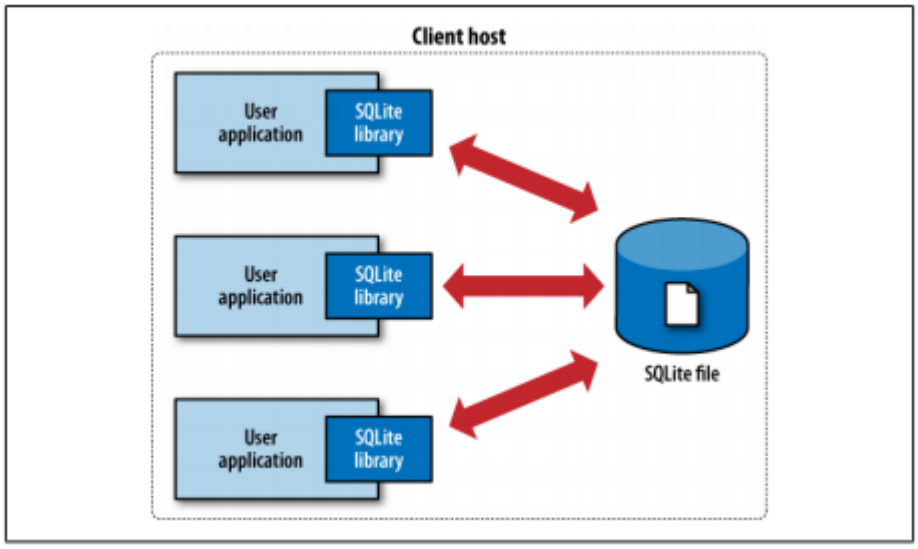

Figura 23: Funcionamento do SQLite
Fonte: Kreibich, 2010

O acesso ao banco de dados SQLite é realizado através a extensão da classe *SQLiteOpenHelper*. Nesta classe é implementado o código para criar e deletar um banco de dados, *scriptSQLCreate* e *scriptSQLDelete*, respectivamente. A figura a seguir exemplifica essas implementações. (PAULA; PEREIRA, 2014)

```
class SQLiteHelper extends SQLiteOpenHelper {

    private static final String CATEGORIA = "livro";

    private String[] scriptSQLCreate;
    private String scriptSQLDelete;

    SQLiteHelper(Context context, String nomeBanco, int versaoBanco, String[] scriptSQLCreate, String scriptSQLDelete) {
        super(context, nomeBanco, null, versaoBanco);
        this.scriptSQLCreate = scriptSQLCreate;
        this.scriptSQLDelete = scriptSQLDelete;
    }
```

Figura 24: Criando e deletando um banco de dados com SQLite
Fonte: Paula; Pereira, 2014

3. METODOLOGIA

Este trabalho foi executado baseando-se no estudo e pesquisa dos atuais problemas diários, ainda sem solução, da população cega e desenvolvimento de um aplicativo capaz de realizar uma interação virtual entre a necessidade e a solução. A metodologia de desenvolvimento deste trabalho deu-se nas seguintes etapas:

a) Etapa 1: Análise da literatura das seguintes áreas: tecnologia assistiva, desenvolvimento mobile com Android e deficiência visual.

b) Etapa 2: Levantamento de requisitos baseando-se em entrevistas e estudos com o público alvo.

c) Etapa 3: Análise dos requisitos e pré-modelagem de classes.

d) Etapa 4: Modelagem do banco de dados e da camada de persistência.

e) Etapa 5: Modelagem da camada de negócios e de apresentação.

f) Etapa 6: Desenvolvimento do sistema e pequenos testes.

g) Etapa 7: Correção de problemas encontrados e manutenção

4. APLICATIVO “SMARTEYES” – PROJETO

4.1 DESCRIÇÃO E CONCEITO

O aplicativo “*SmartEyes*” para a plataforma Android tem como principal objetivo amenizar a necessidade de deficientes visuais de terem sempre alguém sem a deficiência por perto para auxiliá-los nas atividades do dia a dia.

“*SmartEyes*” trabalha basicamente recebendo em um servidor os questionamentos e as mídias autoexplicativas, cada vez que nosso servidor receber um novo arquivo fará uma verificação dos usuários cadastrados que não possuem deficiência que estejam com seu aparelho de celular conectado à internet e enviará à estes uma notificação da existência de um novo questionamento não respondido. O usuário possuirá a opção de resolver o questionamento e enviar para o servidor a solução ou ignorar, caso a segunda opção seja escolhida, o servidor fará uma nova busca e enviará a notificação para outro usuário e assim sucessivamente até o momento em que o questionamento seja resolvido. Assim que obtivermos uma solução nosso servidor enviará a resposta para o usuário solicitante. A resposta será entregue ao solicitante em formato .mp3.

4.2 DIAGRAMA DE FLUXO DE DADOS

Segundo Gane(1988) o Diagrama de Fluxo de Dados (DFD) é uma das principais ferramentas para se planejar um sistema, pois é o único documento que mostra todas as relações entre os processos, os dados e as funções que transformam os dados.

> O diagrama de fluxo de dados é uma ferramenta de modelagem que permite que um sistema seja visto como uma rede de processos assíncronos e funcionais, interligados por fluxos de dados e repositórios de armazenamento de dados. É uma das mais utilizadas ferramentas de modelagem de sistemas, principalmente para sistemas nos quais as funções do sistema sejam mais importantes e mais complexas que os dados manipulados pelo sistema. (BALANCIERI, BOVO & FERRARI, pág. 4, 2014)

A seguir é apresentando o DFD do aplicativo “*SmartEyes*”

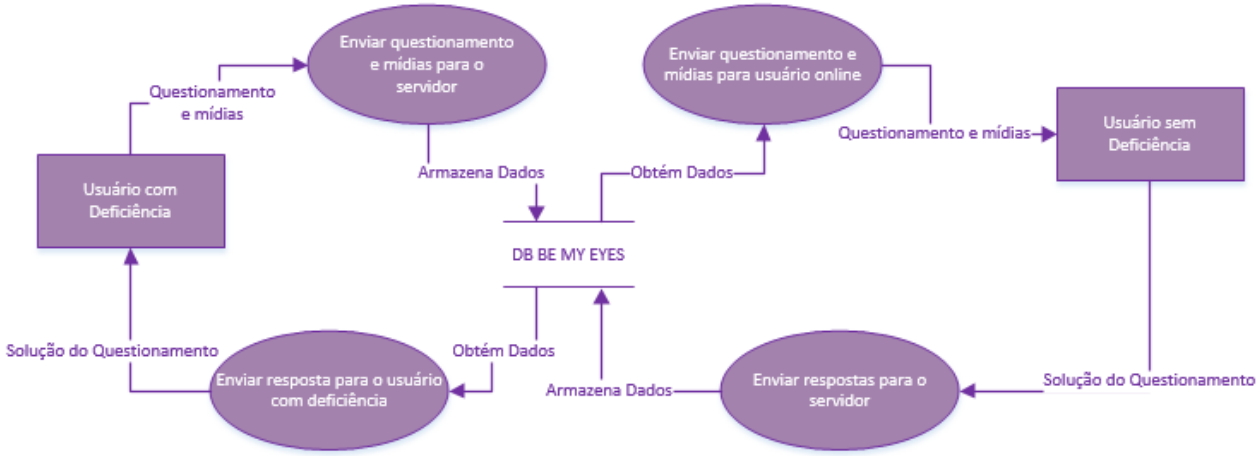

Diagrama 1: DFD do aplicativo SmartEyes.
Fonte: O próprio autor

4.3 DIAGRAMA DE CASOS DE USO

Conforme Booch (2000) um sistema interage com atores humanos ou qualquer outro agente externo que se utilizam desse sistema para algum propósito e, esses atores esperam que o sistema se comporte conforme o esperado. Um caso de uso especifica o comportamento de um sistema ou de parte de um sistema e descreve um conjunto de sequências de ações, incluindo variantes realizadas pelo sistema para produzir um resultado observável do valor de um ator.

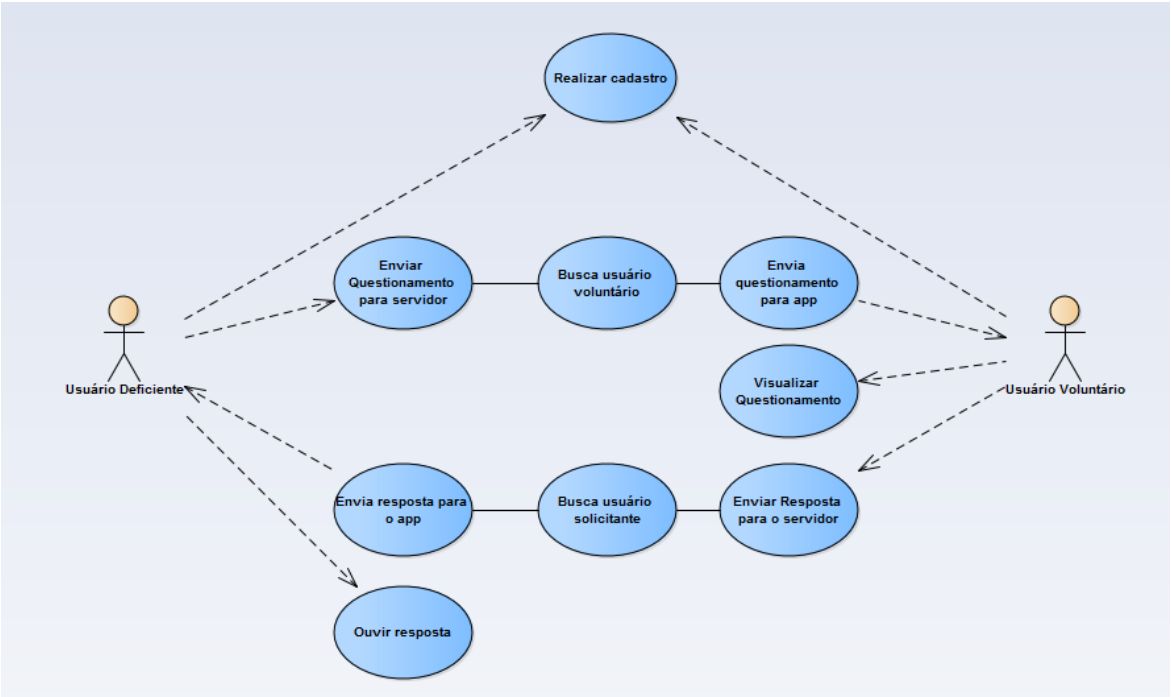

Diagrama 2: Diagrama de Caso de Uso do aplicativo SmartEyes
Fonte: O próprio autor

4.4 TABELA DE EVENTOS

A lista de eventos relaciona os fatos do ambiente externo que o sistema obrigatoriamente deve reconhecer. A inclusão ou não de uma entidade externa e os fluxos de dados correspondentes podem aumentar ou diminuir a abrangência de um sistema. Podem ser consideradas entidades externas um elemento do ambiente externo ao sistema que gera informações que dão entrada no sistema ou que recebe informações que são geradas pelo sistema (BEZERRA, 2003).

Os eventos podem ser classificados como externos, quando são fatos que ocorrem fora do sistema e fazem com que este gere uma resposta, ou temporais, quando estão relacionados com a passagem de tempo (BEZERRA, 2003).

Estímulos são os fluxos de dados que entram no sistema, são estes fluxos que informam ao sistema que um evento ocorreu (BEZERRA, 2003).

Eventos	Classificação	Estímulos	Resposta
Usuário deficiente/voluntário realiza cadastro	Evento externo	Usuário	Confirmação de cadastro
Usuário realiza login	Evento externo	Credenciais	Confirmação de login
Usuário deficiente grava um vídeo de questionamento	Evento externo	Arquivo de vídeo .mp4	Confirmação de envio
É hora de buscar um usuário voluntário	Evento temporal	-----	Envio de arquivo de vídeo
Usuário voluntário grava um áudio respostas	Evento externo	Arquivo de áudio .mp3	Confirmação de envio
É hora de enviar uma resposta	Evento temporal	------	Envio de arquivo de áudio

Tabela 5: Tabela de Eventos
Fonte: O próprio autor

4.5 DIAGRAMA DE SEQUÊNCIA

É considerado um dos diagramas de interação, pode ser usado para mostrar a evolução de uma dada situação em determinado momento do software, mostrar dada colaboração entre duas ou mais classes e pode, também, ser usado para mostrar a tradução de um Caso de Uso desde a interação com o usuário até a finalização daquele dado processo (RAMOS, 2013).

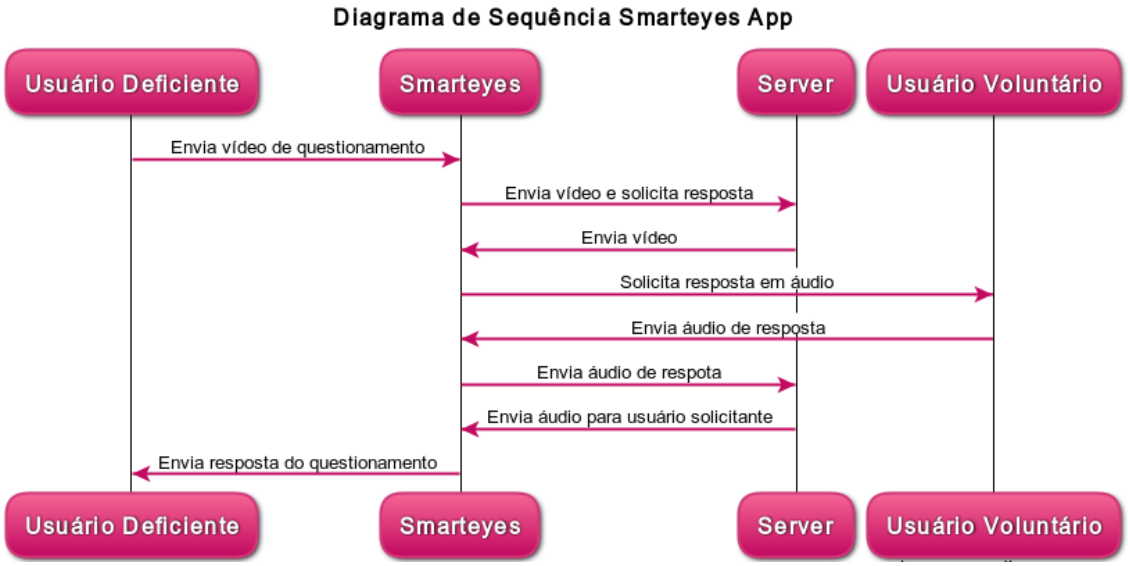

Diagrama 3: Diagrama de Sequência
Fonte: O próprio autor

4.6 DIAGRAMA DE ATIVIDADE

O diagrama de atividades é um diagrama UML utilizado para modelar o aspecto comportamental de processos. É um dos diagramas que mais sofreu mudanças em seu meta-modelo, desde seu surgimento no UML 1.0. Neste diagrama, uma atividade é modelada como uma sequência estruturada de ações, controladas potencialmente por nós de decisão e sincronismo. Em seu aspecto mais simples, um diagrama de atividades pode ser confundido com um fluxograma. Entretanto, ao contrário de fluxogramas, os diagramas de atividades UML suportam diversos outros recursos, tais como as partições e os nós do tipo fork e merge, além da definição de regiões de interrupção, que permitem uma modelagem bem mais rica do que simplesmente um fluxograma (GUDWIN, 2014).

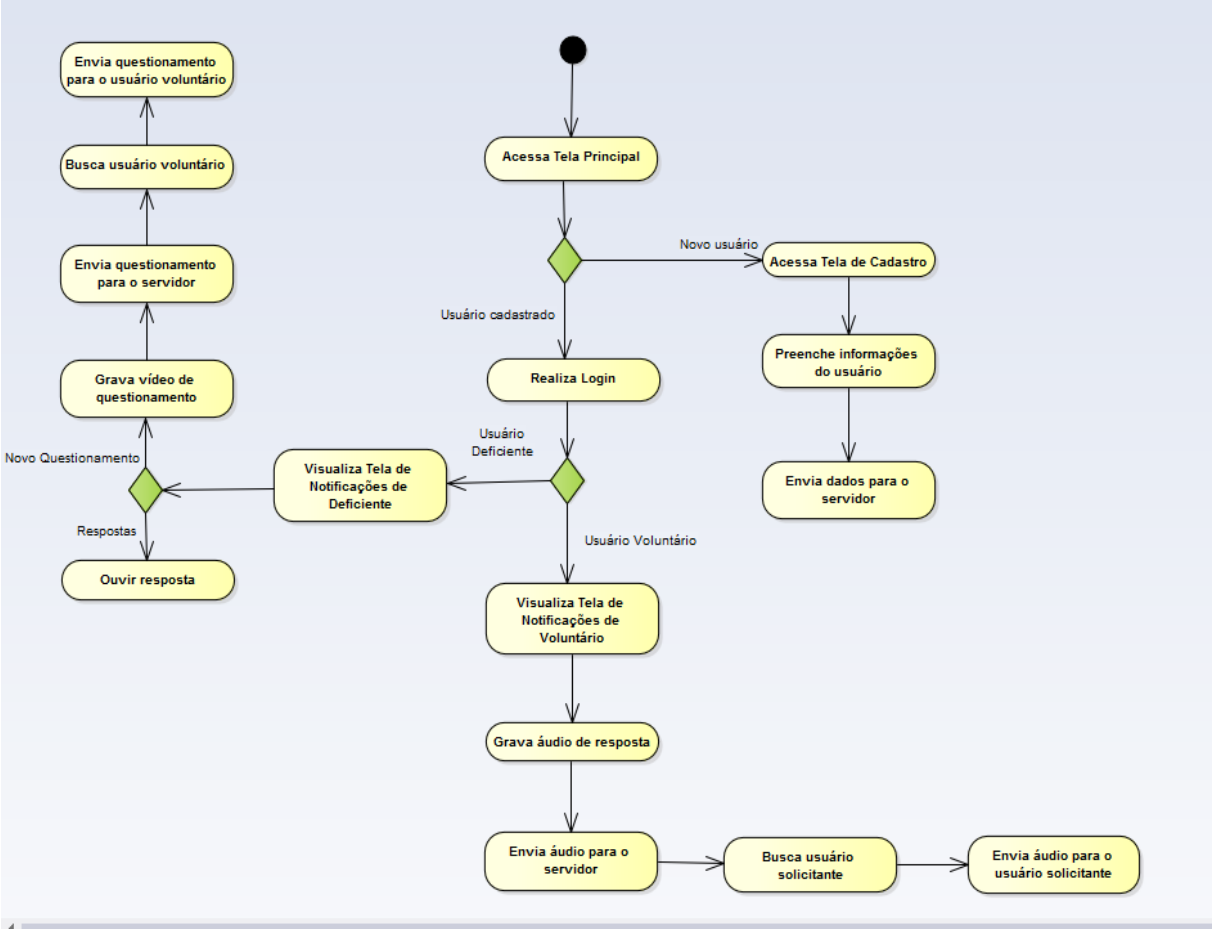

Diagrama 4: Diagrama de Atividade
Fonte: O próprio autor

4.7 TELAS

A seguir serão apresentadas as principais telas do aplicativo *SmartEyes*, que foram desenvolvidas até a data final de entrega desta monografia, podendo haver pequenas alterações no software final apresentado à banca.

Lembrando que sempre que houver um novo questionamento ou uma nova resposta, independentemente de o aplicativo estar em uso, desde que haja conexão com a internet, uma notificação será apresentada na barra de tarefas do Android.

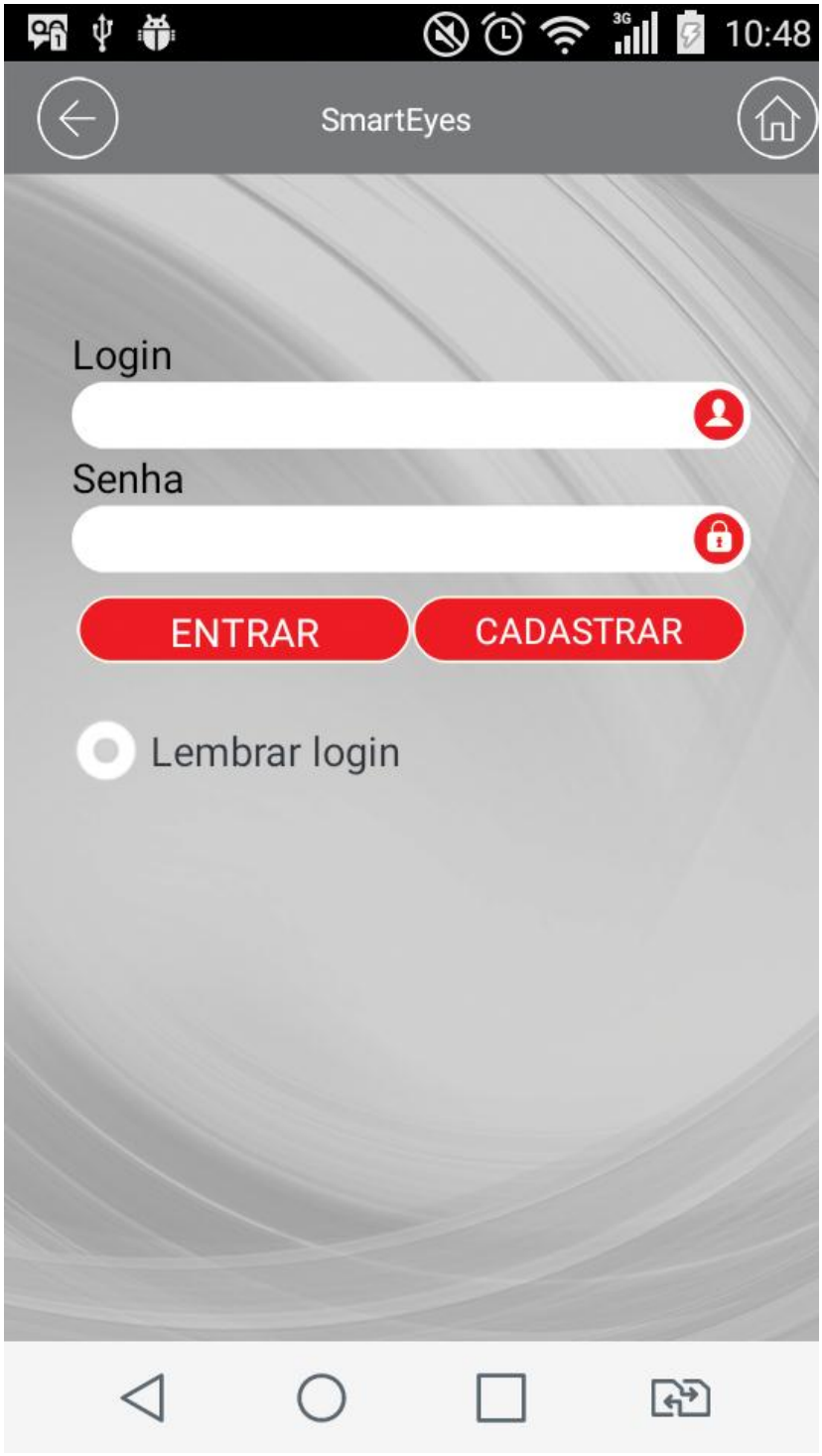

Figura 25: Tela Principal do Aplicativo
Fonte: O próprio autor

Esta tela será apresentada quando o usuário selecionar o ícone do aplicativo. Ele terá então a opção de acessar o aplicativo diretamente, caso já esteja cadastrado ou realizar o seu cadastro. As telas apresentadas serão as mesmas para ambos os usuários (deficientes ou não).

Existe a opção de “Lembrar login”, para que não seja necessário realiza-lo toda vez que acessar o aplicativo.

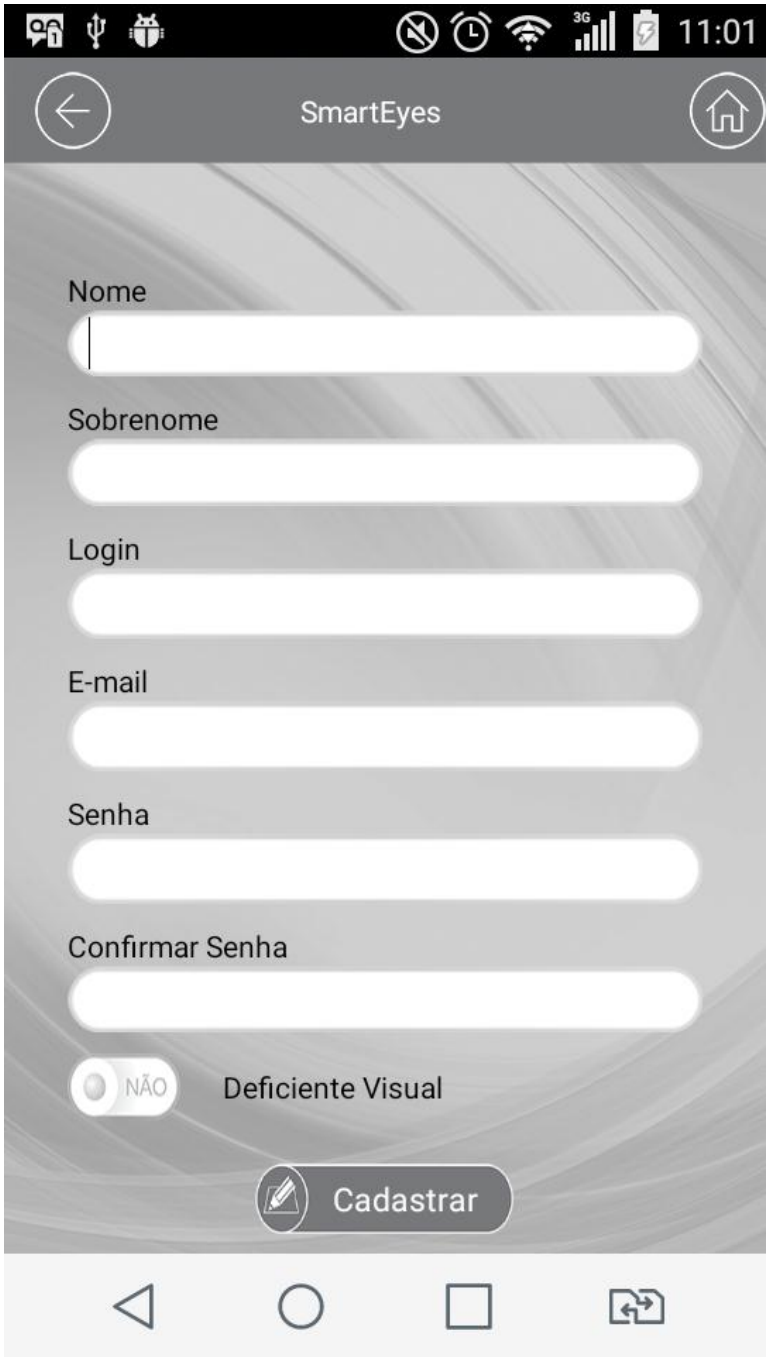

Figura 26: Tela de Cadastro
Fonte: O próprio autor

Na tela de cadastro, o usuário deverá informar seus dados básicos para ter acesso ao aplicativo. Deverá também ser informado se é um deficiente ou voluntário, para que ele seja

inserido corretamente no grupo de usuário correspondente. Estes grupos são necessários para que as trocas de arquivos sejam realizadas de maneira correta.

Figura 27: Tela de Notificações de Deficiente
Fonte: O próprio autor

A tela notificações saudará o usuário de acordo com o nome cadastrado e informará as notificações não visualizadas, e existirá a possibilidade de enviar um novo questionamento,

acionando o botão correspondente com o auxílio do Talk Back do Android. Ao selecionar o botão de questionamento será aberta a câmera do aparelho no modo de gravação de vídeo. Com o vídeo gravado, ao aceitar a gravação esta é automaticamente enviada ao web service via http, caso o aparelho possua conexão com a internet. Caso a conexão esteja indisponível, o arquivo não será enviado quando a conexão voltar, será necessário efetuar a gravação novamente.

Figura 28: Tela Notificação de Voluntário
Fonte: O próprio autor

Nesta tela serão apresentadas, ao voluntário, as notificações pendentes. Ao clicar em uma delas será exibido o player de vídeo padrão do aparelho, para reproduzir o vídeo. Ao clicar em responder, o usuário será encaminhado para o aplicativo de gravação de voz padrão do aparelho e ao aceitar o áudio este será encaminhado para o web service, via http, caso haja conexão com a internet, e o web servisse se encarrega de enviar o arquivo para o usuário que enviou o questionamento correspondente.

4.8 TRECHOS DE CÓDIGO

A seguir são apresentados alguns trechos de códigos para exemplificar o funcionamento do sistema.

- Botão de Cadastro – Localizado na ActivityMain, a primeira tela do sistema. O código a seguir representa a busca do componente gráfico da interface do usuário e é setado o evento de clique do botão, que indica a abertura de uma nova activity.

```
private void getButtonCadastre() {
    if(this.buttonCadastre == null){
        this.buttonCadastre = (Button) findViewById(R.id.buttonCadastrar);
        this.buttonCadastre.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent intent = new Intent(activity, ActivityCadastre.class);
                startActivity(intent);
            }
        });
    }
}
```

Figura 29: Código do botão de Cadastro
Fonte: O próprio autor

- Lembrar Login – Localizado na ActivityMain. Este código representa o método que irá guardar as credenciais de login do usuário, para que não seja necessário inseri-las a cada utilização do aplicativo.

```
public void saveLogin() {
    if (toggleButtonSaveLogin.isChecked()) {
        AtsSingleton.getInstance().setRememberLogin(
                editTextLogin.getText().toString().replace(".", "").
                replace("-", ""), activity);
        AtsSingleton.getInstance().setRememberPassword(
                editTextPassword.getText().toString(), activity);
    } else {
        clearEditTexts();
        AtsSingleton.getInstance().setRememberLogin(null, activity);
        AtsSingleton.getInstance().setRememberPassword(null, activity);
    }
}
```

Figura 30: Método para lembrar login
Fonte: O próprio autor

- Botão de Cadastro – Localizado na ActivityCadastre, este é o botão responsável por chamar o método de validação dos campos de informações pessoais preenchidos pelo usuário e persisti-los no banco de dados do aparelho, SQLite.

```
private void getButtonCadastre() {
    if (this.buttonCadastre == null){
        this.buttonCadastre = (Button) findViewById(R.id.buttonCadastre);
        this.buttonCadastre.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String validate = validateUser();
                if (!validate.equals(""))
                    Toast.makeText(ActivityCadastre.this,
                            validate, Toast.LENGTH_LONG).show();
                else {
                    try{
                        userDAO = new UserDao(ActivityCadastre.this);
                        userDAO.open();
                        userDAO.save(getUser());

                        Toast.makeText(ActivityCadastre.this,
                                "Usuário inserido com sucesso!", Toast.LENGTH_LONG).show();
                    } catch (Exception ex){
                        Toast.makeText(ActivityCadastre.this,
                                ex.getMessage().toString(), Toast.LENGTH_LONG).show();
                    }
                }
            }
        });
    }
}
```

Figura 31: Botão de Cadastro
Fonte: O próprio autor

- ClassToValues – Método responsável por converter o objeto Usuário em um formato para ser inserido no banco de dados SQLite.

```
public static ContentValues classToValues(User user) {

    ContentValues values = new ContentValues();

    try {
        values.put(ID, user.getId());
        values.put(NAME, user.getName());
        values.put(LASTNAME, user.getLastname());
        values.put(EMAIL, user.getEmail());
        values.put(PASSWORD, user.getPassword());
        values.put(IS_DEFICIENT, user.isDeficient() ? BooleanProps.TRUE : BooleanProps.FALSE);
        values.put(IS_HELPER, user.isHelper() ? BooleanProps.TRUE : BooleanProps.FALSE);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return values;
}
```

Figura 32: Método classToValues – Usuário
Fonte: O próprio autor

- CursorToClass – Método responsável por converter os valores retornados de uma consulta ao banco de dados SQLite em um objeto do sistema.

```
public static User cursorToClass(Cursor c){
    if(c == null || c.getCount() < 1)
        return null;

    User user = new User();
    user.setId(c.getLong(c.getColumnIndex(ID)));
    user.setName(c.getString(c.getColumnIndex(NAME)));
    user.setLastname(c.getString(c.getColumnIndex(LASTNAME)));
    user.setEmail(c.getString(c.getColumnIndex(EMAIL)));
    user.setPassword(c.getString(c.getColumnIndex(PASSWORD)));
    user.setDeficient(c.getInt(c.getColumnIndex(IS_DEFICIENT))
            == BooleanProps.TRUE ? true : false);
    user.setHelper(c.getInt(c.getColumnIndex(IS_HELPER))
            == BooleanProps.TRUE ? true : false);
    return user;
}
```

Figura 33: Método cursorToClass – Usuário
Fonte: o próprio autor

- Núcleo do método de login no Web Service - Localizado na classe UserWebService, este trecho de código realiza a autenticação do usuário no Web Service. Em um primeiro momento, é realizada as validações básicas do Web Service, tais como a validação do token e estrutura básica de dados. Em seguida, é realizado uma busca no banco de dados com os dados oriundos da requisição, afim de localizar o usuário solicitado. Por fim, é realizada a conversão do resultado da busca em JSON e retornado ao solicitante da informação (cliente-side)

```
request = convertJsonToObject(requestJson,
        new TypeToken<JsonRequest<JsonLoginRequest>>() {});

User user = getUserDao().loadByLogin(request.getRequest().getLogin(),
        request.getRequest().getPassword(), false);

JsonResponse response = null;
if (user != null)
    response = new JsonResponse(true, "Login realizado com sucesso!", user);
else
    response = new JsonResponse(true, "Usuário não encontrado!");

return getGson(true, false).toJson(response);
```

Figura 34: Núcleo do método de login no Web Service
Fonte: O próprio autor

5. CONSIDERAÇÕES FINAIS

O objetivo geral desse trabalho foi desenvolver uma arquitetura que permita, de maneira distribuída, realizar a troca de arquivos de áudio e vídeo a partir de aparelhos de celular, smartphones, com conexão ativa à internet. Nesse sentido, foi realizada uma revisão das áreas de descoberta de conhecimento e computação móvel, visando suportar a proposição do trabalho. Na base da arquitetura encontra-se o protocolo de transferência de hipertexto. Esse protocolo de comunicação é utilizado em sistemas de informação de hipermídia, distribuídos e colaborativos. Ele é a base para a comunicação de dados da Worl Wide Web. Tal abordagem utiliza o modelo cliente-servidor, baseando-se no paradigma de requisição e resposta, possibilitando assim o ganho de desempenho no sistema desenvolvido.

O método de transferência de arquivos utilizado, se mostrou consistente e viabilizou a aplicação do sistema em grande escala. Com o intuito de melhorar o desempenho e oferecer possibilidades adicionais, o modelo foi implementado de modo que este possa ser refatorado e permita a adição de novas funcionalidades, sem que sua estrutura necessite grandes alterações.

O protótipo desenvolvido atendeu as expectativas gerando resultados satisfatórios e permitindo a utilização do sistema em um grande número de usuários. A arquitetura do protótipo demonstrou flexibilidade e escalabilidade podendo ser expandida, quando necessário, por meio de smartphones com sistema operacional Android.

Outro ponto importante a considerar refere-se ao modelo de dados que suporta o protótipo desenvolvido. Para tal, foi utilizado o conceito de modelagem entidade-relacionamento, no qual a finalidade é descrever os dados a serem utilizados no sistema de informação. Esse modelo pode em princípio representar qualquer domínio de aplicação que se baseie em relacionamentos, entidades e atributos. Possui ainda, como característica importante, a possibilidade de representação gráfica, o diagrama entidade relacionamento.

Ao longo deste trabalho surgiram novas possibilidades que não foram desenvolvidas, pois tornariam este trabalho muito extenso. As duas principais possibilidades são a implementação do modelo de rancking de auxilio entre usuários e a conexão do aplicativo com as principais redes sociais, tais como Facebook, Twitter e Instagram. Apesar destes conceitos não terem sidos acoplados ao protótipo o modelo foi projetado pensando nestas futuras melhorias.

REFERÊNCIAS BIBLIOGRAFICAS

ABLESON, W. Frank; KING, Chris; SEN, Robi; ORTIZ, C. Enrique. **Android em ação** – 3ª Edição. – Rio de Janeiro – 2012.

ALECRIM, Emerson. **Máquina Virtual Java (Java Virtual Machine).** 2005. Disponível em <http://www.vivaolinux.com.br/artigo/Maquina-Virtual-Java-(Java-Virtual-Machine)>. Acesso em 25 de março de 2015.

AMARAL, Adriana do. **Principais causas da cegueira**. 2012. Disponível em <http://www.universovisual.com.br/publisher/preview.php?edicao=1112&id_mat=5300>. Acesso em 09 de abril de 2015.

BALANCIERI, Renato. BOVO, Alessandro Botelho. FERRARI, Sandra. **Do Diagrama de Fluxo de Dados ao Use Case**. 2014

BERSCH, Rita; SARTORETTO, Maria Lúcia. **Assistiva, Tecnologia e Educação**. 2014. Disponível < http://www.assistiva.com.br/tassistiva.html>. Acesso em 10 de abril de 2015.

BEZERRA, E. **Princípios de Análise e Projeto com a UML**, ed. Campus-Elsevier. 2003

BLEWITT, Alex. **Qual é o futuro do Apache Harmony?**. 2011. Disponível em < http://www.infoq.com/br/news/2011/03/apache-harmony>. Acesso em 26 de março de 2015.

BLEWITT, ALEX. **10 anos de Eclipse**. 2011. Traduzido por Giovanni Abner. Disponível em <http://www.infoq.com/br/news/2011/11/eclipse-10>. Acesso em 20 de março de 2015.

BOOCH, G. et al. (2000) **UML – Guia do Usuário**. Editora Campus, Rio de Janeiro.

BULL, Ian. Top 10 Eclipse Luna Features. 2014. Disponível em <http://eclipsesource.com/blogs/2014/06/25/top-10-eclipse-luna-features/>. Acesso em 20 de março de 2015.

BUI, Si. **Android SDK Instruction**. 2012. Disponível em < http://support.tapit.com/entries/22015031-Android-SDK-Instruction>. Acesso em 26 de março de 2015.

CACCO. **Software para desenvolvimento de diagramas**. Disponível em <https://cacoo.com/>. Acesso em 19 de abril de 2015.

CAELUM. **Apostila Java e Orientação a Objetos**. Disponível em: < http://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/#2-3-maquina-virtual>. Acesso em 25 de março de 2015.

CARVALHO, Suelen Goularte. **Android Studio: Vantagens e Desvantagens com relação ao Eclipse**. 2013. Disponível em <http://imasters.com.br/mobile/android/android-studio-vantagens-e-desvantagens-com-relacao-ao-eclipse/>. Acesso em 26 de março de 2015.

CARVALHO, Suelen Goularte – **A História do Surgimento da Plataforma Android** – 2014. Disponível em <http://pt.slideshare.net/suelengc/a-histria-do-surgimento-da-plataforma-mvel-android>. Acesso em 10 de março de 2015

CAT, Comitê de Ajudas Técnicas. **Coordenadoria Nacional para a Integração da Pessoa Portadora de Deficiência.** 2014. Disponível em <portal.mj.gov.br/corde/arquivos/doc/Ata_VII_Reunião_do_Comitê_de_Ajudas_Técnicas.doc>. Acesso em 10 de abril de 2015.

CHEPTSOY, Andrey. **IntelliJ IDEA 12 is Available for Download**. 2012. Disponível em < http://blog.jetbrains.com/idea/2012/12/intellij-idea-12-is-available-for-download/>. Acesso em 20 de março de 2015.

CODELEARN. **Android Intent**. 2014 Disponível em < http://www.codelearn.org/android-tutorial/android-intent>. Acesso em 26 de março de 2015.

CONDE, Antônio João Menescal. **Deficiência Visual: a cegueira e a baixa visão.** 2012. Disponível em <http://www.bengalalegal.com/cegueira-e-baixa-visao>. Acesso em 8 de abril de 2015.

CONSTANT, Instituto Benjamin. **Vocabulário Braille, Conceituação Básica.** Disponível em <http://www.ibc.gov.br/Nucleus/?catid=112&itemid=344>. Acesso em 05 de março de 2015.

COSTA, Renata. **Como funciona o sistema Braille?** 2009. Disponível em <http://revistaescola.abril.com.br/formacao/como-funciona-sistema-braille-496102.shtml>. Acesso em 10 de abril de 2015.

DÉBORA. **Alunos deficientes visuais em uma aula de matemática**. 2010. Disponível em < http://sopadenumerosecalculos.com.br/2010/11/alunos-deficientes-visuais-numa-aula-de.html>. Acesso em 10 de abril de 2015.

DEITEL, Harvey. Paul. **Java Como Programar**. 2005

DEMARTINI, Felipe. **Google revela novas ferramentas para desenvolvedores de aplicativos**. 2013. Disponível em < http://www.tecmundo.com.br/google-i-o-2013/39758-google-revela-novas-ferramentas-para-desenvolvedores-de-aplicativos.htm>. Acesso em 20 de março de 2015.

DEVELOPERS, Android. **Android Studio Overview**. 2015. Disponível em < http://developer.android.com/tools/studio/index.html>. Acesso em 20 de março de 2015.

DEVELOPERS, Android. **Dashboards**. 2015. Disponível em < https://developer.android.com/about/dashboards/index.html>. Acesso em 10 de março de 2015.

DEVELOPERS, Android. **<Intent-filter>.** 2015. Disponível em < http://developer.android.com/guide/topics/manifest/intent-filter-element.html>. Acesso em 31 de março de 2015.

DIAS, Klessis Lopes; FONTES, Wescley Pimentel. **Desenvolvimento de Aplicações para Dispositivos Móveis utilizando a Plataforma J2ME**. 2003.

DIMARZIO; JEROME F.; **Android: A programmer's Guide**, New York. McGrawHill E-Book. 2008.

DONAHUE, Bill. **A Tale of migration from Eclipse to Android Studio**. 2014. Disponível em <http://engineering.meetme.com/2014/02/a-tale-of-migrating-from-eclipse-to-android-studio/>. Acesso em 26 de março de 2015.

ECLIPSE. **Simultaneous Releases.** Disponível em <http://projects.eclipse.org/releases/>. Acesso em 20 de março de 2015.

FARAH, Sean. **Why It's Important to Update Java**. 2014. Disponível em < http://onsitepcsolution.com/important-update-java/>. Acesso em 24 de março de 2015.

FINSLAB. **IntelliJ IDEA**, 2014. Disponível em < http://finslab.com/enciclopedia/letra-i/intellij-idea.php>. Acesso em 20 de março de 2015.

FLAVIO, Andre. **Java Virtual Machine e seus conceitos (para iniciantes).** 2010. Disponível em <http://www.vivaolinux.com.br/dica/Java-Virtual-Machine-e-seus-conceitos-(para-iniciantes)>. Acesso em 25 de março de 2015.

FONSECA, Natanael. **Introdução à plataforma Android**. 2013. Disponível em < http://pt.slideshare.net/natanaelfonseca/introduo-plataforma-android>. Acesso em 26 de março de 2015.

GANE, C. (1988**) Desenvolvimento Rápido de Sistemas.** Livros Técnicos e Científicos Editora LTDA, Rio de Janeiro.

GONÇALVES, Eduardo Corrêa. **SQLite, Muito Prazer**! 2014. Disponível em < http://www.devmedia.com.br/sqlite-muito-prazer/7100>. Acesso em 03 de abril de 2015.

GUDWIN, Ricardo R. **Diagramas de Atividade e Diagrama de Estado**. 2014. Disponível em <http://www.dca.fee.unicamp.br/~gudwin/ftp/ea976/AtEst.pdf>. Acesso em 26 de julho de 2015.

JACOME, Fábio. **NetBeans - História**. 2013. Disponível em <http://blogfabricabrasil.blogspot.com.br/2013/12/netbeans.html>. Acesso em 20 de março de 2015.

JETBRAINS. **JetBrains Company History and Timeline**. Disponível em < https://www.jetbrains.com/company/history.jsp>. Acesso em 20 de março de 2015.

JORDÃO, Fábio. **Google lança oficialmente o Android 4.4 KitKat**. 2013. Disponível em <http://www.tecmundo.com.br/android/46463-google-lanca-oficialmente-o-android-4-4-kitkat.htm> Acesso em 10 de março de 2015.

KREIBICH, Jay A. **Using SQLite**. 2010.

LARAMARA, **Associação Brasileira de Assistência à Pessoa com Deficiência Visual. Estatísticas e Causas.** Disponível em <http://laramara.org.br/deficiencia-visual/estatisticas-e-causas>. Acesso em 04 de março de 2015.

LEAL, Dra. Daena Nascimento Barros. **SOCIEDADE BRASILEIRA DE VISÃO SUBNORMAL – Acuidade Visual**. 2014. Disponível em <http://www.cbo.com.br/subnorma/conceito.htm>. Acesso em 08 de abril de 2015.

LISBOA, Adriano. **AFINAL DE CONTAS, O QUE É O GRADLE E COMO USÁ-LO?.** 2015. Disponível em <http://adrianolisboa.com/afinal-de-contas-o-que-e-o-gradle-e-como-usa-lo/>. Acesso em 26 de março de 2015.

LOUREIRO, Melissa. **Sistema da Visão Humana Anatomia do Olho.** 2014. Disponível em < http://slideplayer.com.br/slide/1842811/>. Acesso em 08 de abril de 2015.

LUZZI, Luciano. Android – **Persistência de Dados usando SQLite**. 2013. Disponível em < http://www.mobiltec.com.br/blog/index.php/android-persistencia-de-dados-usando-sqlite/>. Acesso em 03 de abril de 2015.

MONTEIRO, João B. **Entendendo as Intents do Android**. 2013. Disponível em < http://blog.tecsinapse.com.br/post/20652538899/android-intents>. Acesso em 26 de março de 2015.

MONTEIRO, João B. **Entendendo as Intents do Android - Parte 2**. 2013. Disponível em <http://blog.tecsinapse.com.br/post/21322036778/entendendo-as-intents-do-android-parte-2>. Acesso em 27 de março de 2015

MORANDI, Paulo Sérgio; RODOLPHO, Luiz Roberto Lethang**. Desenvolvimento Mobile nas Plataformas Android e iOS.** 2011. Disponível em < http://www.mobiltec.com.br/blog/index.php/desenvolvimento-mobile-nas-plataformas-android-e-ios/>. Acesso em 26 de março de 2015.

NAKAHARA, Leonardo. **Content Providers – Parte 1**. 2011. Disponível em <http://celeiroandroid.blogspot.com.br/2011/04/content-providers-parte-01.html>. Acesso em 03 de abril de 2015.

NETBEANS. **Um breve histórico do NetBeans**. Disponível em <https://netbeans.org/about/history_pt_BR.html>. Acesso em 20 de março de 2015.

NOWILL, Fundação Dorina. **Deficiência Visual – Cuidados com a visão na infância**. 2014. Disponível em <http://www.fundacaodorina.org.br/deficiencia-visual/>. Acesso em 10 de abril de 2015.

PACIEVITCH, Yuri. **História do Java**. Disponível em < http://www.infoescola.com/informatica/historia-do-java/>. Acesso em 24 de março de 2015.

PAULA, Nilton Cézar de; PEREIRA, Guilherme Henrique Vieira**. Gerenciamento de dados em dispositivos móveis.** 2014. Disponível em <http://periodicos.uems.br/novo/index.php/enic/article/viewFile/4570/2002>. Acesso em 05 de abril de 2015.

R4R, Right Place for Right Person. **Architecture of J2ME.** 2015. Disponível em < http://r4r.co.in/java/j2me/basic/tutorial/j2me/Architecture_of_J2ME.shtml>. Acesso em 06 de abril de 2015.

RAMOS, Ricardo Argenton. **UML – Aula I Diagramas de Caso de Uso, Sequência e Colaboração**, 2013. Disponível em <http://www.univasf.edu.br/~ricardo.aramos/disciplinas/ES_II_2013_1/UML_AulaI.pdf> Acesso em 25 de julho de 2015.

REWORDIT. **Looking Back on World Braille Day: How Louis Braille Changed the World for the Bling.** 2015. Disponível em <http://www.rewordit.org/looking-back-on-world-braille-day-how-louis-braille-changed-the-world-for-the-bling/>. Acesso em 10 de abril de 2015.

RINALDI, Camila. **Review do Android 5.0 Lollipop: 10 coisas que você precisa saber sobre o novo OS da Google.** 2013. Disponível em <http://www.androidpit.com.br/android-5-0-lollipop-review>. Acesso em 10 de março de 2015.

ROCHA, Leonardo. **Android Studio: ferramenta de criação de apps da Google ganha versão 1.0.** 2014. Disponível em < http://www.tecmundo.com.br/android/69111-android-studio-ferramenta-criacao-apps-google-ganha-versao-1-0.htm>. Acesso em 20 de março de 2015.

ROMANATO, Allan**. Entenda como funciona a Java Virtual Machine (JVM).** Disponível em < http://www.devmedia.com.br/entenda-como-funciona-a-java-virtual-machine-jvm/27624>. Acesso em 24 de março de 2015.

ROUSE, Margaret. **Integrated Development Environment (IDE).** 2007. Disponível em <http://searchsoftwarequality.techtarget.com/definition/integrated-development-environment>. Acesso em 18 de março de 2015.

SANTANA, Otávio Gonçalves de. **Eclipse no Brasil: 10 anos de história**. 2011 Disponível em <http://www.devmedia.com.br/eclipse-no-brasil-10-anos-de-historia/22989>. Acesso em 20 de março de 2015.

SANTINO, Renato. **Android já teve 10 versões diferentes; Relembre a evolução do sistema.** 2013. Disponível em <http://olhardigital.uol.com.br/noticia/android-ja-teve-10-versoes-diferentes-relembre-a-evolucao-do-sistema/35801> Acesso em 10 de março de 2015.

SHENGYANG, Luo. **Componente de aplicação Android – Content Provider, Breve introdução.** 2011. Disponível em < http://blog.csdn.net/luoshengyang/article/details/6946067>. Acesso em 03 de abril de 2015

SILVEIRA, Felipe. **Activity – o que é isso?.** 2010. Disponível em < http://www.felipesilveira.com.br/2010/05/activity-o-que-e-isso/>. Acesso em 26 de março de 2015.

SOARES, Diego**. Glossário Android – O Que É Dalvik VM?.** 2014. Disponível em < http://www.escolaandroid.com/android-dalvik/>. Acesso em 26 de março de 2015.

SOUSA, Rainer. **Código Braille**. 2008. Disponível em <http://www.brasilescola.com/portugues/braile.htm>. Acesso em 10 de abril de 2015.

SOUZA, Felipe A. Gavazza de. **Tecnologia ART: o que realmente é e quais as suas vantagens?.** 2014. Disponível em < http://www.androidpit.com.br/tecnologia-art-dalvik>. Acesso em 26 de março de 2015.

STRICKLAND, Jonathan. **A arquitetura do Android**. 2014. Disponível em < http://tecnologia.hsw.uol.com.br/google-phone2.htm>. Acesso em 26 de março de 2015.

VEJAM. **Campo Visual.** 2014. Disponível em < http://www.vejam.com.br/baixavisao-campo-visual/>. Acesso em 08 de abril de 2015.

WINCHESTER, Joe. **IBM WebSphere Developer Technical Journal: Comparing WebSphere Studio Application Developer with Visual Age for Java -- Part 3.** 2002. Disponível em < http://www.ibm.com/developerworks/websphere/techjournal/0210_winchester/winchester.html>. Acesso em 20 de março de 2015.