UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Gustavo Carvalho Souza

APLICATIVO ANDROID PARA A FERRAMENTA DE BUSCA MATEMÁTICA *SEARCHONMATH*

Alfenas, 30 de junho de 2015.

UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

APLICATIVO ANDROID PARA A FERRAMENTA DE BUSCA MATEMÁTICA *SEARCHONMATH*

Gustavo Carvalho Souza

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Alfenas como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador: Dr. Flavio Barbieri Gonzaga

Alfenas, 30 de junho de 2015.

GUSTAVO CARVALHO SOUZA

DESENVOLVIMENTO DE APLICATIVO ANDROID PARA A FERRAMENTA DE BUSCA MATEMÁTICA *SEARCHONMATH*

A Banca examinadora abaixo-assinada aprova a monografia apresentada como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

MSc. Everton Josué Silva

Prof. Dr. Ricardo Menezes Salgado

Universidade Federal de Alfenas

Prof. Dr. Flavio Barbieri Gonzaga (Orientador)

Universidade Federal de Alfenas

Alfenas, 30 de junho de 2015.

AGRADECIMENTO

Agradeço primeiramente a Deus por me conceder a vida e por me permitir evoluir em todos os aspectos dia após dia.

Aos meus pais Ana Beatriz Carvalho Souza e Vicente Roberto de Souza por ter me ensinado a valorizar o estudo e a aquisição de conhecimento e por constantemente me apoiarem.

Ao meu irmão e amigo Ricardo Carvalho Souza que muito me ensina.

Por cada um dos meus familiares que direta ou indiretamente me apoiam.

A minha esposa Gislene Regina Fernandes pelo carinho, pelos conselhos e por me apoiar na tomada de decisão.

A cada um dos professores do Curso de Bacharelado em Ciência da Computação pelos ensinamentos de teorias e práticas transmitidos a mim.

Ao professor Dr. Flavio Barbieri Gonzaga pela oportunidade da realização deste projeto, pela orientação e pelo conhecimento proporcionado.

Gustavo Carvalho Souza

RESUMO

Este projeto teve por objetivo desenvolver um aplicativo móvel para a ferramenta de busca por fórmulas matemáticas *SearchOnMath*. A ferramenta *SearchOnMath* permite que estudantes e pesquisadores encontrem páginas na web que contêm graus de similaridade com fórmulas matemáticas pesquisadas (GONZAGA, 2013). Desenvolver um aplicativo para a ferramenta é de extrema importância, visto que os maiores acessos à ferramenta tem sido realizados por dispositivos móveis. A *SearhOnMath* versão móvel foi desenvolvida para a plataforma *Android* tendo como requisito mínimo a versão 3.0 com API nível 11 ou versões superiores. O presente aplicativo oferece ao usuário a facilidade e eficiência na realização de consultas por fórmulas matemáticas pela Web, mantendo a qualidade e eficiência do sistema *SearchOnMath* versão Web já existente. Neste projeto, foram explorados componentes da tecnologia JAVA úteis para o desenvolvimento de aplicativos *Android*, assim como detalhes da arquitetura e componentes do sistema operacional *Android*.

Palavras-Chave: *SearchOnMath*, motor de busca, fórmulas matemáticas, *Android*, aplicativo móvel.

ABSTRACT

This project aimed to develop a mobile application for the search engine for mathematical formulas SearchOnMath. The SearchOnMath tool allows students and researchers to find web pages that contain degrees of similarity researched mathematical formulas (Gonzaga, 2013). Develop an application for the tool is of utmost importance, since the greatest access to the tool has been conducted by mobile devices. The SearhOnMath mobile version was developed for the Android platform with a minimum requirement version 3.0 with API level 11 or higher versions. This application offers the user the ease and efficiency of consultations by mathematical formulas for the Web while maintaining quality and system efficiency SearchOnMath existing Web version. In this project we were explored components of Java technology useful for developing Android applications as well as architectural details and components of the Android operating system.

Keywords: *SearchOnMath,* search engine, mathematical formulas, Android, mobile application.

LISTA DE FIGURAS

FIGURA 1: QUANTIDADE DE ACESSOS À *SEARCHONMATH* ENTRE 01 DE JANEIRO E 20 DE JUNHO DE 2015..... 13
FIGURA 2: QUANTIDADE DE ACESSOS À *SEARCHONMATH* ENTRE 21 DE MAIO E 20 DE JUNHO DE 2015.13
FIGURA 3: COMPONENTES DA ESTRUTURA GERAL DO PROJETO E SUAS RELAÇÕES. 20
FIGURA 4: CÓDIGO UTILIZADO PARA DEFINIR UM *SERVICE*. 22
FIGURA 5: CÓDIGO PARA A CRIAÇÃO DO METADADO UTILIZADO PELO *SERVICE*. 22
FIGURA 6: CÓDIGO PARA A CRIAÇÃO DO LAYOUT DO TECLADO VIRTUAL..... 23
FIGURA 7: TECLADO *QWERTY* DESTACANDO-SE OS SÍMBOLOS ADICIONADOS E O TECLADO DE SÍMBOLOS MATEMÁTICOS EXIBIDO. 24
FIGURA 8: 1ª LINHA DO TECLADO DE SUBCONJUNTOS MATEMÁTICOS E SEU CÓDIGO CORRESPONDENTE. 25
FIGURA 9: MECANISMO DE CONVERSÃO AO SER PRESSIONADA UMA TECLA EQUIVALENTE A UM SÍMBOLO MATEMÁTICO..... 27
FIGURA 10: CICLO DE VIDA DE UMA *ACTIVITY*, MÉTODOS E AÇÕES DOS USUÁRIOS A ELES RELACIONADOS (DEVELOPERS, 2015G)..... 29
FIGURA 11: TELA INICIAL DO APLICATIVO CONTENDO O LOGO DA *SEARCHONMATH*..... 32
FIGURA 12: TELA DE PESQUISA DESTACANDO A EQUAÇÃO TEX PROCESSADA COM *MATHJAX* E EXIBIDA COM *WEBVIEW*..... 36
FIGURA 13: RESULTADO FINAL DA TELA QUE PERMITE O USUÁRIO INSERIR A DIMENSÃO DA MATRIZ... 39
FIGURA 14: RESULTADO FINAL DA TELA PARA A CRIAÇÃO DE UMA MATRIZ 2X2. 41
FIGURA 15: TELA DE VISUALIZAÇÃO DA MATRIZ 2X2. 43
FIGURA 16: TELA DE PESQUISA PARA INSERIR EQUAÇÃO EM TEX, VISUALIZÁ-LA E CONSULTÁ-LA AO WEBSERVICE. 46
FIGURA 17: A - FORMATO DE DADOS JSON RETORNADOS. B - *VIEW* DA LISTA DE RESULTADOS CONTENDO OS VALORES RETORNADOS. 50
FIGURA 18: ESTRUTURA UTILIZADA PARA A CRIAÇÃO DA LISTA DE RESULTADOS..... 55
FIGURA 19: TELA DE RESULTADOS DISPONIBILIZADO EM LISTAS. 56
FIGURA 20: PRIMEIRA LINHA DO TECLADO DE SUBCONJUNTO MATEMÁTICO E SEUS NOVOS TECLADOS. 58
FIGURA 21: SEGUNDA LINHA DO TECLADO DE SUBCONJUNTO MATEMÁTICO E SEUS NOVOS TECLADOS RELACIONADOS..... 58
FIGURA 22: TERCEIRA LINHA DO TECLADO DE SUBCONJUNTOS MATEMÁTICOS E ESEUS NOVOS TECLADOS RELACIONADOS..... 59
FIGURA 23: SEQUÊNCIA DE TECLAS PRESSIONADAS PARA CRIAR O 1º MEMBRO DA EQUAÇÃO. 60
FIGURA 24: SEQUÊNCIA DE TECLAS PRESSIONADAS PARA CONSTRUIR O 2º MEMBRO DA EQUAÇÃO, DE MANEIRA GERAL. 61
FIGURA 25: EQUAÇÃO EM FORMATO TEX CRIADA E PROCESSADA NA TELA DE PESQUISA. 62
FIGURA 26: TELA DE RESULTADOS PARA A PESQUISA DA EQUAÇÃO REALIZADA. 63
FIGURA 27: 4 PRIMEIROS RESULTADOS APRESENTADOS APÓS UMA CONSULTA DA FÓRMULA (1) NA *SEARCHONMATH* VERSÃO *WEB*, COMPROVANDO A IGUALDADE DOS RESULTADOS COM O APLICATIVO DESENVOLVIDO. 64
FIGURA 28: BARRA DE PROGRESSO EM DESTAQUE NA TELA DE PESQUISA E TELA DE RESULTADOS..... 65
FIGURA 29: ALTERAÇÃO DA COR DA *VIEW* AO SER PRESSIONADA. 66

SUMÁRIO

1 INTRODUÇÃO 11

1.1 JUSTIFICATIVA E MOTIVAÇÃO 12
1.2 PROBLEMATIZAÇÃO 14
1.3 OBJETIVOS 14
1.3.1 Gerais 14
1.3.2 Específicos 15

2 REVISÃO BIBLIOGRÁFICA 16

2.1 SISTEMA OPERACIONAL *ANDROID* 16
2.2 *WEBSERVICE* 17
2.3 TRABALHOS RELACIONADOS 17
2.3.1 Aplicativos *Android* 17
2.3.2 Aplicativos *Android* no contexto de ferramentas de busca 18
2.3.3 Aplicativos *Android* no contexto de utilitários matemáticos 19

3 DESENVOLVENDO O APLICATIVO *SEARCHONMATH* 20

3.1 ESTRUTURA GERAL DO PROJETO 20
3.2 DESENVOLVENDO UM TECLADO VIRTUAL 21
3.2.1 Desenvolvendo um Novo *Input Method Editor* 21
3.2.2 Desenvolvendo Layout do Teclado Virtual 22
3.2.3 Desenvolvendo Classe para Exibir o Teclado e Processar eventos 25
3.2.4 *HashMap* para Conversão de Símbolos *Unicodes* em TeX Correspondentes 26
3.3 DESENVOLVENDO TELAS EM APLICATIVOS *ANDROID* 27
3.3.1 Desenvolvimento de Elementos de UI - *Layouts* e *Views* 27
3.3.2 Desenvolvendo Classe para Interface Gráfica – *Activity* 28
3.3.3 Definindo *Activity* no arquivo *AndroidManifest.xml* 30
3.4 EVENTOS EM APLICATIVOS *ANDROID* – *LISTENERS* 31
3.5 DESENVOLVENDO TELA INICIAL – *SPLASH SCREEN* 31
3.5.1 Criando *Layout* da Tela Inicial 32
3.5.2 Criando classe da tela inicial 32
3.6 CONFIGURANDO *WEBVIEW*, *MATHJAX* E CHAMADA DE SCRIPTS 33
3.6.1 *WebView* 33
3.6.2 *MathJax* 33
3.6.3 Configuração da *WebView* com *MathJax* 34
3.7 DESENVOLVENDO TELAS PARA INSERÇÃO DE MATRIZ 36
3.7.1 Desenvolvendo Tela para Escolha do Tamanho da Matriz 37
3.7.2 Desenvolvendo Tela de Matriz 39
3.7.3 Desenvolvendo tela de visualização da matriz 42
3.8 DESENVOLVENDO TELA PESQUISA 43
3.8.1 Desenvolvendo *Layout* em *xml* para a Tela de Pesquisa 44
3.8.2 Desenvolvendo Classe de *User Interface* para Tela de Pesquisa 45
3.8.3 Adicionando eventos ao Botão de Busca 45
3.9 CONECTANDO O APLICATIVO *SEARCHONMATH* AO *WEBSERVICE* 47
3.9.1 Premissa para Eficiência e Performance em Conexões com a Internet 47
3.9.2 Clientes *Android* HTTP para Acesso a Internet 47

3.9.3 Biblioteca *Volley* para Transmissão de Dados em Rede 48
3.9.4 WebService 48
3.9.5 Conectando ao *WebService* e Realizando Análise de JSON 50
3.10 DESENVOLVENDO TELA DE RESULTADOS 51
3.10.1 Desenvolvendo Listas de Exibição de Resultados 52
3.10.2 Criando *Layouts* da Lista de Resultados 52
3.10.3 Criando Padrão *Adapter* 53
3.10.4 Adicionando o *Adapter* à *ListView* 54
3.10.5 Criando Paginação para Controle de Dados 55
4 RESULTADOS 57
4.1 TECLADO PERSONALIZADO 57
4.2 PESQUISA REALIZADA 59
4.3 EXPERIÊNCIA DO USUÁRIO 64
4.3.1 Barra de Progresso 64
4.3.2 Resposta Visual a Eventos 65
4.3.3 Rápido Acesso a Símbolos Matemáticos 66
5 CONCLUSÕES E TRABALHOS FUTUROS 67
5.1 CONCLUSÕES 67
5.2 TRABALHOS FUTUROS 69
6 REFERÊNCIAS BIBLIOGRÁFICAS 70

1 Introdução

Este capítulo tem por finalidade apresentar as principais motivações e objetivos do projeto desenvolvido.

Segundo um estudo realizado pela *Idc - Analize The Future* (2013), empresa especializada no mercado de tecnologia e telecomunicações, a venda de *smartphones* (celulares que apresentam sistemas operacionais e disponibilizam o uso de aplicativos) irá superar o uso de celulares simples, os chamados *feature phones.* Além disso, já era esperado que em 2014 o número de vendas de *Tablets* ultrapassasse o número de *Notebooks* vendidos, segundo este mesmo órgão.

Acompanhado deste crescimento das vendas e da utilização de *smartphones* e *Tablets,* está o aumento do uso de aplicativos, das mais diferentes categorias, por parte dos usuários. Conforme uma pesquisa realizada pela *Flurry Analytics,* em 2013, a utilização global de aplicativos publicou um crescimento de 115% ano-a-ano, sendo que utilitários e aplicativos de produtividade apresentaram um aumento de 150% no uso (KHALAF, 2014).

Saindo do cenário de aplicativos, mas associado ao principal responsável por grande número de acessos na Web, estão os Mecanismos de busca na Internet. Eles contribuem para que os usuários possam encontrar todos os tipos de informações sobre pessoas, lugares, estados, esportes, dentre muitas outras coisas. Os motores de busca estão evoluindo a ponto de se tornarem guias confiáveis que realçam toda experiência on-line, em vez de serem instrumentos que simplesmente apontam o caminho para um viajante perdido na Internet.

O sistema *SearchOnMath*, um mecanismo de busca criado recentemente, permite que os usuários realizem buscas por fórmulas matemáticas na Internet. O sistema é de extrema importância para alunos, professores, pesquisadores e demais usuários que necessitem de comprovações, teoremas, axiomas e do saber matemático em geral (GONZAGA, 2013).

Sendo assim, teve por objetivo neste trabalho desenvolver um aplicativo *Android* para a ferramenta de busca por fórmulas matemáticas *SearchOnMath,* para a partir daí, unir o crescimento desta vasta utilização de aplicativos de produtividade com uma alternativa de qualidade e eficiência para a busca de comprovações por fórmulas matemática na Web.

1.1 Justificativa e Motivação

A preferência da utilização de *smartphones* e *Tablets* por parte dos usuários em relação aos convencionais celulares simples e *notebooks* é comprovada pelo aumento do número de vendas de *smartphones* que acabou por ultrapassar o número de vendas de *notebooks*.

O aumento da utilização de aplicativos de produtividade e utilitários pode ser comprovado pela pesquisa realizada pela empresa *Flurry Analytics* no ano de 2013 que registrou um aumento do mesmo de 150% a cada ano.

Sendo assim, a maior justificativa para se desenvolver o aplicativo para a ferramenta de busca por fórmulas matemáticas *SearchOnMath* é atender cada um dos usuários que utilizam aplicativos de produtividade em *smartphones* para a resolução de problemas matemáticos cotidianos.

Esta crescente utilização de *smartphones* por parte dos usuários, pode ser comprovada pelo aumento do número acessos à ferramenta *SearchOnMath* por usuários utilizando dispositivos móveis. Conforme apresentado nas Figuras 1 e 2 abaixo, percebe-se que: na Figura 1, referente aos meses entre janeiro e junho do ano de 2015, houve uma quantidade maior de acessos de usuários com desktops em relação a acessos de usuários com *Tablets* e dispositivos móveis; na Figura 2, referente aos meses entre maio e junho do ano de 2015, percebe-se que houve uma quantidade maior de acessos realizados por usuários com dispositivos móveis.

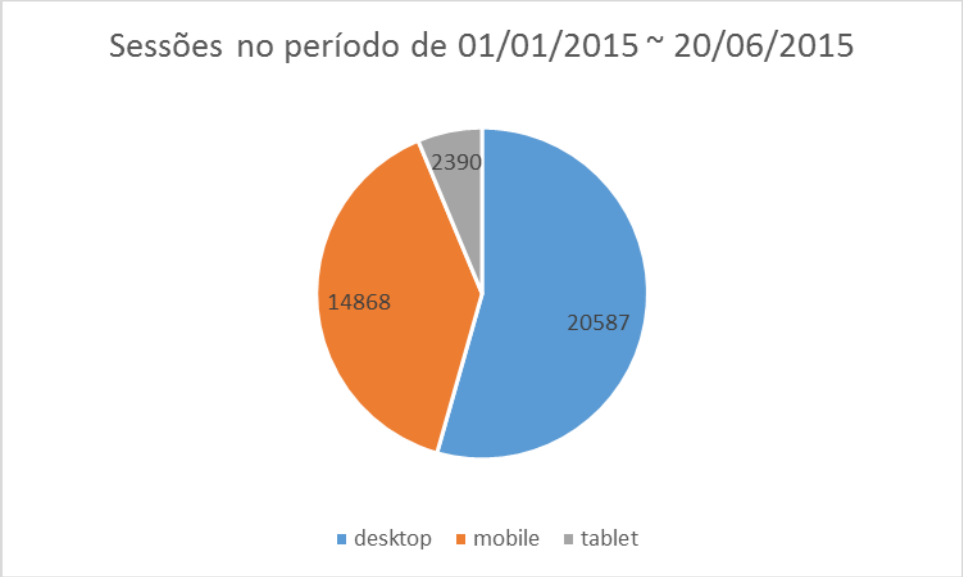


Figura 1: Quantidade de acessos à *SearchOnMath* entre 01 de janeiro e 20 de junho de 2015.

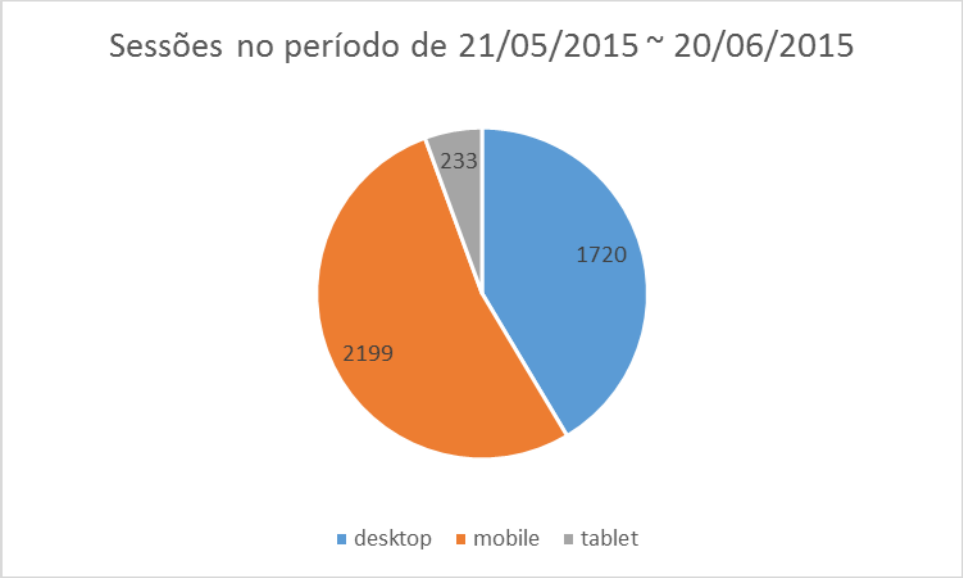


Figura 2: Quantidade de acessos à *SearchOnMath* entre 21 de maio e 20 de junho de 2015.

Desta forma, a motivação maior é atender de forma específica cada um destes usuários de aplicativos, abranger o campo de atuação da ferramenta e aumentar ainda mais o número de acessos.

Do ponto de vista do sistema operacional utilizado, optou-se pelo *Android* pelo fato de que ele constitui hoje, a plataforma móvel mais popular do mundo, estando presente em 190 países. Além disso, conta com um poderoso *framework* de desenvolvimento e abre mercado para a distribuição de seus aplicativos.

1.2 Problematização

No cenário atual, não são encontrados aplicativos que realizem buscas matemáticas pela web para a plataforma *Android*, consistindo no desenvolvimento de um aplicativo cuja ideia é única. Além do que oferecerá mobilidade para a realização das consultas por fórmulas matemáticas expandindo a proposta da ferramenta *SearchOnMath* já existente na versão web. Desta forma, será possível desenvolver um aplicativo para a ferramenta de busca por fórmulas matemáticas *SearchOnMath* disponível para a plataforma *Android?*

1.3 Objetivos

1.3.1 Gerais

O objetivo principal deste projeto é desenvolver um aplicativo *mobile* para a ferramenta de busca por fórmulas matemáticas *SearchOnMath* - já existente em versão web, e que seja voltado para a plataforma *Android*.

1.3.2 Específicos

Os objetivos específicos são:

- Desenvolver uma tela de busca do sistema, que permitirá o usuário criar suas equações em formato TeX – linguagens para expressões matemáticas (KNUTH, 1984), exibir esta equação processada e realizar a busca desta equação ao *WebService*.
- Desenvolver um novo teclado para a plataforma *Android*, sendo que este deve conter os diferentes símbolos matemáticos utilizados para formação de diferentes equações no formato TeX.
- Desenvolver ou utilizar bibliotecas/APIs que permitam o processamento de equações no formato TeX para a possível montagem da consulta para a realização da busca da equação matemática.
- Desenvolver mecanismos para acessar um *WebService* a partir do aplicativo e carregar as informações do resultado da busca matemática oferecidos pelo *WebService*.
- Desenvolver tela de resultado da consulta realizada pelo usuário a partir da equação fornecida, contendo informações do site que contêm a equação e que permita o usuário ser redirecionada até ela.

2 Revisão Bibliográfica

Este capítulo apresenta uma revisão bibliográfica sobre o projeto desenvolvido.

2.1 Sistema Operacional *Android*

O A*ndroid* é um sistema operacional móvel baseado em uma versão Linux. Foi desenvolvido pela *OpenHandset Alliance*, uma aliança entre várias empresas, dentre elas a *Google*. Como o *Android* é de código aberto e está disponível gratuitamente para que os fabricantes o personalizem, não há configurações fixas de *hardware* e *software* (LEE, 2011). Dentre as principais vantagens do *Android*, pode-se citar que ele oferece uma abordagem unificada para o desenvolvimento de aplicativos. Internamente o *Android* suporta as seguintes funcionalidades:

- Armazenamento – utiliza o *SQlite,* para armazenar dados estruturados em um banco de dados particular; *Shared Preferences,* para armazenar dados privados em formatos de pares chave/valor; *Network Connection,* para armazenar dados na web dentro de seu próprio servidor, dentre outras*;*
- Conectividade – suporta *WIFI, Bluetooth, GSM* e outras;
- Troca de mensagens – suporta tanto SMS quanto MMS;
- Navegador Web – baseado no WebKit de código aberto, juntamente com *JavaScript* V8 do *Chrome*;
- Suporte a diferentes formatos de mídias – inclui MP3, MP4, MIDI, JPEG dentre outras;
- Suporte a hardware – sensor de acelerômetro, câmera, bússola digital, sensor de proximidade;
- Multitoques – suporte a telas multitoques;
- Multitarefa – suporte aplicativo multitarefa;

- Suporte *flash* – o *Android* 2.3 suporta o flash 10.1;
- Tethering – suporte ao compartilhamento de conexões de Internet como *hotspot* com e sem fios.

2.2 *WebService*

WebServices são serviços que podem ser tanto clientes quanto servidores de aplicações que se comunicam pela *web* por meio de protocolos HTTP (Oracle, 2013). Quando se comportam como servidores de aplicações podem disponibilizar os resultados em formatos de dados como XML e JSON para melhor eficiência na análise dos dados por parte da aplicação cliente.

2.3 Trabalhos Relacionados

2.3.1 Aplicativos *Android*

Muitos são os trabalhos sendo realizados cujos intuitos são o desenvolvimento de aplicativos para a plataforma *Android* destinado a resolver problemas dos usuários num contexto de aplicações específicas.

Na área da medicina, pode-se citar o desenvolvimento de um aplicativo que permite obter informações completas e atualizadas sobre uma lista enorme de doenças, procedimentos e drogas chamado *Medscap (*MEDSCAPE MOBILE*, 2015)*. Este aplicativo oferece a vantagem do usuário baixar sua base de dados no dispositivo móvel, podendo utilizar o aplicativo sem estar conectado à Internet.

Na área da agricultura, pode-se citar o desenvolvimento de um aplicativo utilizado para prática da agricultura de precisão chamado *Agri Precision* – Agricultura (GOOGLE PLAY, 2015_a). Neste aplicativo é possível criar grade amostral e pontos de contorno da grade para que sejam criados mapas nos *softwares* de Agricultura de Precisão para computadores pessoais.

Como pode ser observado, o desenvolvimento de aplicativos *Android* está presente em diferentes áreas objetivando resolver problemas distintos.

2.3.2 Aplicativos *Android* no contexto de ferramentas de busca

Dada a grande importância da utilização de ferramentas de busca pela Web, ultimamente tem-se desenvolvido um grande número de aplicativos visando expandir a mobilidade destes serviços.

O aplicativo *DuckDuckGo*, também disponível para outras duas plataformas *(iOS* e *BlackBerry*), vem com melhorias da proposta utilizada no seu motor de busca disponível na web. *DuckDuckGo* apresenta-se como um motor de busca capaz de oferecer privacidade ao usuário, ou seja, não recolhe ou compartilha suas informações pessoais (*GOOGLE PLAY*, 2015$_b$).

O aplicativo *Search Engine,* possui a particularidade de realizar buscas a partir de todos os diferentes mecanismos de buscas inteligentes existentes como Google, Ask, Bing, Yahoo, Amazon dentre outros (GOOGLE PLAY, 2015$_c$).

Percebe-se também um crescente desenvolvimento de aplicativos de motores de busca utilizados para objetivos em específicos.

Neste contexto, o aplicativo *Job Search,* um abrangente motor de busca utilizado para encontrar empregos, permite que o usuário, a partir de uma única pesquisa, tenha acesso a milhões de postos de trabalho a através de milhares de sites de empresas e sites de empregos (GOOGLE PLAY, 2015$_d$).

No setor turístico, podemos citar o aplicativo *Travel Search Engine* que realiza buscas através de 100 sites diferentes de viagem, permitindo que os usuários encontrem imediatamente um histórico contendo sugestões de sites de viagem que possuam as menores taxas e preços (GOOGLE PLAY, 2013).

No contexto de buscas por imagens, o aplicativo *Image Search* destaca-se por oferecer aos usuários a capacidade de buscar imagens por diferentes cores, formatos, tamanhos, níveis de segurança e domínios específicos (GOOGLE PLAY, 2014).

Na gastronomia, um aplicativo de busca com grande destaque é o *Recipe Search Engine.* Com este aplicativo é possível encontrar mais de 100 mil receitas, criar lista de favoritos das receitas preferidas e compartilhar com os amigos suas receitas favoritas *(EASYCOOKING.IN, 2012).*

2.3.3 Aplicativos *Android* no contexto de utilitários matemáticos

Muitos são os aplicativos existentes visando auxiliar aos usuários na realização de operações matemáticas em geral.

Em Chavan (2013) é desenvolvido um aplicativo que apresenta a solução de uma equação linear a partir de uma imagem da equação presente em uma câmera. A aplicação pode tanto resolver equações escritas a mão quanto impressas em computador.

O aplicativo *Pre-Algebra* tem por objetivo ser um professor de matemática pessoal. Neste aplicativo é simulado uma sala de aula de matemática de escola particular, onde o usuário é único aluno (GOOGLE PLAY, 2012).

Procurando disponibilizar aos usuários fórmulas matemáticas para possíveis utilizações, o aplicativo *Math Formulas* destaca-se por conter o maior número de fórmulas variadas e por permitir que os próprios usuários possam sugerir outras não inseridas (GOOGLE PLAY, 2015_e).

No aplicativo *Symbolab* disponível para plataforna *Iphone* e *Android*, é permitido ao usuário inserir equações matemáticas e resolvê-las (GOOGLE PLAY, 2015_f).

3 Desenvolvendo o Aplicativo *SearchOnMath*

As secções seguintes descrevem as etapas necessárias para o desenvolvimento do aplicativo SearchOnMath para dispositivos Android. Para o desenvolvimento deste aplicativo foi utilizado a linguagem de programação orientada a objetos Java.

3.1 Estrutura Geral do Projeto

Para a criação e desenvolvimento do aplicativo *SearchOnMath* disponível para a plataforma *Android,* dois componentes externos foram necessários para a estrutura do projeto. O primeiro componente trata-se de um banco de dados, contendo as informações referentes aos sites e cada uma das equações, expressões e símbolos matemáticos que eles contêm. Para isto, foi utilizado o banco de dados do próprio sistema *SearchOnMath* disponível para *web.* O segundo componente utilizado foi um *WebService.* A utilização de um *WebService* foi necessária para disponibilizar as informações presentes no banco de dados em um formato de dados mais leve e de fácil análise e processamento. A estrutura geral do projeto de desenvolvimento do aplicativo *SearchOnMath* pode ser visualizado na Figura 3 abaixo:

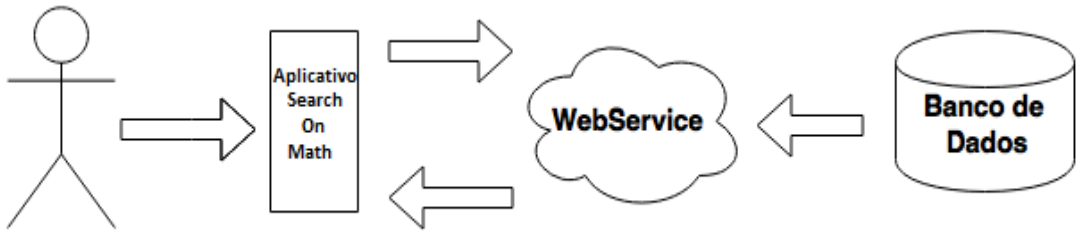


Figura 3: Componentes da estrutura geral do projeto e suas relações.

Pode-se observar na figura acima um banco de dados disponibilizando os dados e informações para o *WebService.* Ao mesmo tempo, o *WebService*

disponibilizando estes dados em um formato de dados mais leve e facilmente processado ao aplicativo. Estes dados retornados equivalem as respostas contendo os dados relacionados ao site que contem a equação consultada pelo usuário.

3.2 Desenvolvendo um Teclado Virtual

Todas novos teclados criados seguiram os preceitos das secções seguintes.

3.2.1 Desenvolvendo um Novo *Input Method Editor*

Os tipos de teclados dos dispositivos móveis variam muito conforme o fabricante. Nos dispositivos *Android* pode-se utilizar tanto teclados em *hardware* quanto utilizar *software* como um teclado virtual (DEVELOPERS, 2015$_a$). Para que isto seja possível, existe o que é chamado de IME (*input method editor*), ou seja, um editor de método de entrada de texto nos dispositivos. Para o desenvolvimento de um novo teclado virtual para o aplicativo do *SearchOnMath* foi necessário a implementação de um novo mecanismo de entrada de texto, seguindo as especificações definidas em Hathibelagal (2014$_a$) .

O desenvolvimento de um teclado virtual personalizado foi de extrema importância para permitir que os usuários pudessem ter acesso a símbolos matemáticos no teclado. O teclado modificado permitiu que os usuários pudessem digitar muitos dos símbolos matemáticos existentes em formato TeX. Em *Android*, todo IME se apresenta como um *Service,* ou seja, como um componente da aplicação que executa em segundo plano (YAMASANI, 2009). Sendo assim, foi definido um *Service* no arquivo de configuração *AndroidManifest.xml* correspondente ao novo IME a ser criado. A definição de um *Service* correspondente ao novo IME criado pode ser visto na Figura 4 apresentada a seguir:

```
<service android:name=".ServiceSOM"
   android:label="SearchOnMath IME"
   android:permission="android.permission.BIND_INPUT_METHOD">
     <meta-data android:name="android.view.im"   android:resource="@xml/method"/>
     <intent-filter>
          <action android:name="android.view.InputMethod" />
     </intent-filter>
</service>
```

Figura 4: Código utilizado para definir um *Service*.

Importantes atributos foram definidos. O atributo *android:name* refere-se ao nome da classe que este *Service* irá responder. Uma permissão é atribuída ao *Service* criado no atributo *android:permission*. Esta permissão indica que este *Service* deverá ser acessado pela chamada de um método de entrada. Na *tag* <*meta-data*> é definido um metadado utilizado para este *Service*. O próximo passo foi criar este metadado definido (HATHIBELAGAL, 2009_b). Este metadado é de extrema importância para que o sistema *Android* reconheça este *Service* como um IME. Este arquivo que possui a extensão *xml* (*eXtensive Markup Language*) é criado contendo informações como o idioma a qual pertence e o nome a ser apresentado no dispositivo. O código utilizado para criar o metadado pode ser visto na Figura 5 abaixo:

```
<input-method xmlns:android="http://schemas.android.com/apk/res/android">
     <subtype
     android:label="English (US) "
     android:imeSubtypeLocale="en_US"
     android:imeSubtypeMode="keyboard" />
</input-method>
```

Figura 5: Código para a criação do metadado utilizado pelo *Service*.

3.2.2 Desenvolvendo Layout do Teclado Virtual

Após definido um *Service* e seu correspondente metadado, foi criado o *Layout* do teclado. O *Layout* do teclado é composto apenas por uma *KeyboardView*. *KeyboardView* é uma *View* que processa o teclado virtual do *Android*

(DEVELOPERS, 2015b). O trecho de código da Figura 6 abaixo mostra como foi criado o Layout do teclado virtual:

```
<android.inputmethodservice.KeyboardView
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/keyboard"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:keyPreviewLayout ="@layout/preview"
/>
```

Figura 6: Código para a criação do Layout do teclado virtual.

O atributo *layout_alignParentBottom* foi marcado como verdadeiro para que o teclado seja posicionado na parte inferior da tela. Logo em seguida, foram criadas as teclas do teclado. Os detalhes das teclas do teclado e suas posições são definidas em um arquivo *xml*. Cada tecla do teclado contêm os seguintes atributos:

- *keyLabe*l: atributo que contém o texto a ser exibido na tecla;
- *keyIcon*: atributo em substituição ao *keylabel*, que permite adicionar ícone a tecla;
- *codes*: contém os valores *unicodes* que as teclas representam ao serem pressionadas;

Existem também outros atributos opcionais como:

- *keyEdgeFlags*: este atributo é utilizado para deixar a tecla mais à direita ou mais à esquerda da tela;
- *keyWidth*: este atributo define a largura da tecla, normalmente em porcentagem;
- *isRepeatable*: se este atributo é definido como verdadeiro, quando a tecla for pressionada longamente terá sua ação repetida. Normalmente usado para barras de espaço e teclas de exclusão.

O teclado inicial do aplicativo *SearchOnMath*, chamado *qwerty*.xml, foi redefinido em relação aos teclados presentes nos dispositivos atuais. A grande mudança foi um ícone novo do teclado contendo uma integral referente ao ícone do aplicativo, que ao ser pressionado trazia para tela o teclado personalizado do aplicativo contendo teclas de subconjuntos matemáticos. Para este teclado foi adicionado também os símbolos de contra barra e abre e fecha chaves.

O teclado personalizado de subconjuntos matemáticos foi criado contendo 12 teclas distribuídas em 3 linhas de 4 colunas. As 12 teclas são relacionadas aos 10 subconjuntos de equações matemáticas existentes, 1 subconjunto de elementos mais utilizados e 1 tecla de retorno para o teclado inicial *qwerty*. Pressionadas estas teclas de subconjuntos de elementos matemáticos, uma nova aba do teclado é apresentada possuindo teclas representando os elementos destes subconjuntos. Os 10 subconjuntos matemáticos representados no teclado personalizado do aplicativo são: subconjuntos de elementos relacionais, subconjunto de setas, subconjunto de letras gregas, subconjunto de elementos lógicos, subconjunto de elementos geométricos, subconjuntos de matrizes, subconjuntos de conjuntos, subconjunto de operadores condicionais, subconjunto de funções inferiores e superiores, subconjunto de operadores de funções específicas. Na Figura 7 abaixo é apresentado o teclado *qwerty* ressaltando os símbolos adicionados e o novo teclado de símbolos matemáticos:

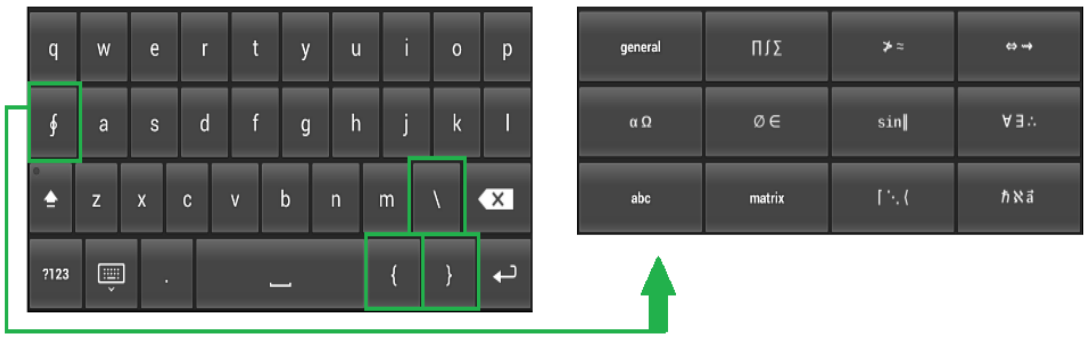


Figura 7: Teclado *qwerty* destacando-se os símbolos adicionados e o teclado de símbolos matemáticos exibido.

Na Figura 8 apresentada a seguir, é representada a 1 primeira linha do teclado de subconjuntos matemáticos, e o código a ele correspondente, sendo a *tag*

Row relacionada a criação de linha do teclado e a *tag Key* relacionada a cada tecla do teclado:

```
<Row>
    <Key android:codes="-11" android:keyLabel="general" />
    <Key android:codes="-105" android:keyIcon="@drawable/sym_keyboard_plustimes"/>
    <Key android:codes="-175" android:keyIcon="@drawable/sym_keyboard_relations"/>
    <Key android:codes="-173" android:keyIcon="@drawable/sym_keyboard_arrow"/>
</Row>
```

Figura 8: 1ª linha do teclado de subconjuntos matemáticos e seu código correspondente.

3.2.3 Desenvolvendo Classe para Exibir o Teclado e Processar eventos

Logo em seguida, foi criada uma classe utilizada para criar o teclado a partir do layout construído e processar o clique de cada botão pressionado. Para isto foi criada uma classe que estendesse da superclasse *InputMethodService* e que implementasse a interface *OnKeyboardActionListener* (HATHIBELAGAL, 2009c). *InputMehodService* fornece métodos para personalizar as diferentes formas de entrada de texto e *OnKeyboardActionListener* contém os métodos que são chamados quando os botões do teclado são tocados ou pressionados (DEVELOPERS, 2015c). Alguns métodos da interface *OnKeyboardActionListener* são:

- *onKey:* utilizado para enviar os códigos pressionados para o ouvinte e a partir daí processá-los. Recebe como parâmetro o código pressionado;
- *onPress:* chamado quando o usuário pressiona uma tecla. Recebe como parâmetro o código pressionado;
- *onRelease:* chamado quando o usuário solta uma tecla. Recebe como parâmetro o código pressionado.

A classe criada para este objetivo recebeu o nome de *CustomKeyboardSOM.java* e possui 3 campos básicos:

- uma *KeyboardView*: usada para referenciar a exibição definida no Layout *keyboardView.xml*;
- um *Keyboard*: uma instância da classe *Keyboard* que é atribuída a *KeyboardView*;
- um *boolean*: utilizado para identificar se a tecla *caps lock* está ou não habilitada.

No momento de criar o teclado, o método o*nCreateInputView* é chamado. Neste método foram inicializadas todas variáveis básicas. A *keyboardView* foi exibida ao usuário com o *Layout* definido em *KeyboardView.xml* e o *Keyoard* foi inflado com o *Layout* do teclado definido em *qwert.xml.* Logo em seguida, o método *onKey* foi implementado. Toda vez que o usuário pressiona uma tecla no teclado virtual, este método é chamado contendo um valor de *unicode* como parâmetro. Por exemplo, ao pressionar a tecla 'A', é passado como parâmetro deste método o *unicode* 97. A partir deste *unicode* é processada a ação a ser realizada. No caso de *unicodes* equivalentes a letras, apenas era mostrado no campo de texto a letra equivalente ao *unicode*, em caso *unicodes* equivalentes a símbolos matemáticos, era consultado em uma tabela *hash* o valor a ser retornado de acordo com o *unicode* passado. Conforme descrito a seguir.

3.2.4 *HashMap* para Conversão de Símbolos *Unicodes* em TeX Correspondentes

Cada tecla pressionada do teclado corresponde ao *unicode* de um símbolo que deve ser mostrado no campo de texto, com exceção de teclas que quando pressionadas redirecionam para outros novos teclados. Porém existem *unicodes* que correspondem a símbolos matemáticos. Neste caso, contudo não é o símbolo que deve ser retornado e mostrado no campo de texto, mas sim o seu correspondente em formato TeX. Para realizar este mapeamento criou-se uma classe *Symbols.java.* Nesta classe é registrado em uma estrutura de dados do tipo *HashMap* chaves equivalentes a cada um dos *unicodes* existentes e seus respectivos valores TeX correspondentes. Esta classe permitia também recuperar o TeX correspondente de

um *unicode* passado como parâmetro. Na Figura 9 é apresentada o formato de conversão de um *unicode* para o seu TeX correspondente:

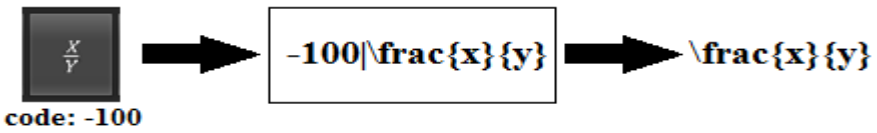


Figura 9: Mecanismo de conversão ao ser pressionada uma tecla equivalente a um símbolo matemático.

3.3 Desenvolvendo Telas em Aplicativos *Android*

A criação de telas em aplicativos *Android* depende basicamente de dois passos muito importantes. O primeiro passo consiste em construir elementos de interface gráfica que irá compor a tela. O segundo passo consiste em recuperar estes elementos já criados e encapsulá-los em objetos Java. A partir deste ponto, é possível adicionar ações aos objetos e torná-los mais dinâmicos. As secções seguintes explicam de maneira clara a forma como isto pode ser feito.

3.3.1 Desenvolvimento de Elementos de UI - *Layouts* e *Views*

A base para a construção de elementos de interface gráfica são os *Layouts*. Os *Layouts* definem uma estrutura visual para a interface do usuário. Eles podem ser definidos declarando o elemento de interface do usuário em um arquivo *xml* ou instanciado em tempos de execução (DEVELOPERS, 2015_d). Um *Layout* pode ser de diferentes tipos. Estes diferentes tipos existentes diferenciam pela forma como seus componentes se apresentam na tela (DEVELOPERS, 2015_d). Dois importantes *Layouts* que podem ser destacados são *o LinearLayout* e o *RelativeLayout.*

O *RelativeLayout* permite exibir os componentes de tela de maneira a posicionar cada um deles em relação aos outros componentes irmãos, posicionando à direita, à esquerda, à cima ou abaixo (DEVELOPERS, 2015$_e$).

O *LinearLayout* permite posicionar seus elementos em linhas ou colunas (DEVELOPERS, 2015$_f$). Um *Layout* pode conter como elemento filho outros *Layouts* ou outros elementos de tela (*Views*).

Outro importante componente para a construção de interface gráfica (UI) são as *Views* . Basicamente todos os elementos de tela são subclasses de *View*. Como principais elementos de tela que são subclasses de *View* pode-se citar: *Buttons*, utilizado para criar botões; *TextFields*, usado para criar campos de textos, *RadioButtons*, usado para permitir ao usuário selecionar uma opção dentro de um conjunto entre outros. Cada elemento de tela é definido com atributos que permitem realizar suas configurações e que permitem reconhecê-los como elementos únicos (id). Destaca-se como atributos de elementos as cláusulas *android:layout_height* e *android:layout_weight* que respectivamente definem a altura e largura dos elementos. A abordagem escolhida para a criação de telas foi a de criar *Layouts* em *xml*, por conter a vantagem de separar elementos de UI dos códigos de controle da aplicação (DEVELOPERS, 2015$_d$).

3.3.2 Desenvolvendo Classe para Interface Gráfica – *Activity*

Em *Android,* o tipo de classe que oferece a capacidade de criar componentes de tela, recuperá-los e adicionar ações a elas é chamada de *Activity.* Toda tela em *Android* é uma *Activity,* ou está sobre uma *Activity.* Sendo assim, toda classe que se comporta como tela, deve estender a classe *Activity* e implementar os métodos que melhor descreve o comportamento do aplicativo. A *Activity* possui ciclos de vida que estão diretamente relacionado a métodos. Os principais métodos relacionados aos ciclos de vida da *Activity* são (DEVELOPERS, 2015$_g$):

- *onStart():*este método é chamado para iniciar a *Activity.*
- *onCreate()*: este método é chamado para criar a *Activity*. É neste método que são adicionados o *Layout* em *xml*, recuperados os elementos da interface

gráfica em objetos e adicionados ações a estes elementos para os que possuem.

- *onResume()*: este método é chamado quando a tela é visível ao usuário e é permitido ao usuário interagir com a tela.
- *onPause():* este método é chamado quando a tela está em segundo plano - existe uma outra tela visível ao usuário e permitindo a interação com ela.
- *onStop():*método utilizado para encerrar uma *Activity*.
- *onDestroy():*método utilizado para destruir uma *Activity*.

O ciclo de vida de uma *Activity* pode ser visualizado na Figura 10.

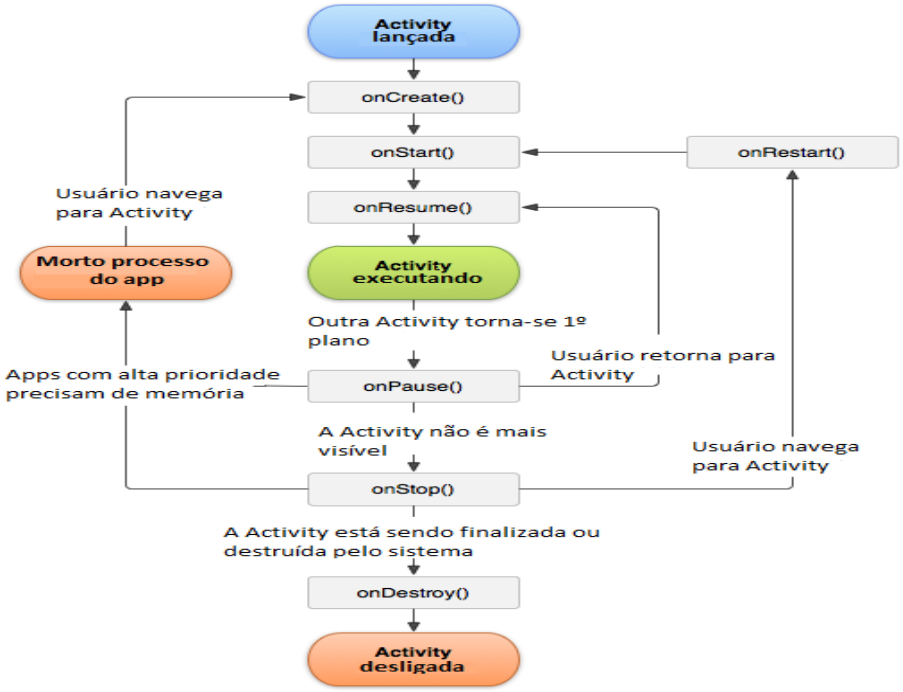


Figura 10: Ciclo de vida de uma *Activity*, métodos e ações dos usuários a eles relacionados (DEVELOPERS, 2015g).

Os métodos *setContentView e findViewById,* são outros métodos de extrema importância para a perfeita criação de telas em *Android* a partir de *Activities.*

O método *setContentView* é utilizado para adicionar à tela o *Layout* em *xml* definido. Sendo assim, para adicionar um *Layout* chamado *activity_main.xml* na interface gráfica, basta chamar o método abaixo:

setContentView(R.layout.activity_main.xml).

A classe R é uma classe gerada automaticamente pelo ambiente de desenvolvimento. Esta classe permite acessar todos os recursos da aplicação. No caso acima, *R.layout.activity_main.xml* significa que está sendo recuperado um recurso de *Layout* chamado *activity_main.xml*.

O método *findViewById* é o método utilizado para recuperar um componente de *View* do layout em *xml* e a partir daí criar objetos para manipulação e criação de ações. Como parâmetro ele recebe o id do recurso a ser recuperado. Desta forma, para criar um objeto *Button* a partir de um *Button* em *xml* de id *myButton*, basta utilizar o método abaixo:

Button myButton = (Button)findViewById(R.id.myButton)

A partir daí, já se pode adicionar ações (*listeners*) ao objeto *Button* definido no layout em *xml*.

3.3.3 Definindo *Activity* no arquivo *AndroidManifest.xml*

Para que uma classe do aplicativo seja reconhecida como uma tela (*Activity*) esta classe deve ser definida em um arquivo de configuração chamado *AndroidManifest,xml*. Neste arquivo de configuração é que são definidas permissões, filtros, serviços, intenções entre tantas outras configurações do aplicativo. Para definir uma *Activity* deve-se usar a tag <activity> dentro da tag <application> e definir alguns atributos (DEVELOPERS, 2015$_g$). A maneira de definir uma *Activity* de nome *ActivityMain* é mostrada pelo pseudocódigo abaixo:

```
<application>
        <activity android:name=".ActivityMain"/>
</application>
```

3.4 Eventos em Aplicativos *Android – Listeners*

Com o intuito de oferecer interação aos usuários, os aplicativos *Android* permitem que elementos de interface gráfica possam responder a gestos, cliques, foco e movimentações realizados pelo usuário. Isto é permitido devido a uma propriedade de elementos que compõem uma tela - *Views*. Os elemetos de *View* possuem as propriedades de permitir que os clientes que as implementam, sejam notificadas das mudanças que futuramente ocorram. Os principais métodos utilizados para implementarem estas interações das *Views* com os usuários são:

- *setOnClickListener()*: este método é chamado para adicionar ações quando o usuário pressiona rapidamente a *View (*realiza um click*).* Este método possui como parâmetro um objeto do tipo *OnClickListener*, que de fato implementa a ação realizada pelo usuário;
- *setOnFocusChangeListener()*: este método é chamado para adicionar ações quando o foco de uma determinada *View* é modificada. Este método possui como parâmetro um objeto do tipo *OnFocusChangeListener*, que de fato implementa a ação quando é modificado o foco da *View*.

Os *Listeners* permitem que os aplicativos sejam mais dinâmicos e flexíveis.

3.5 Desenvolvendo tela inicial – *Splash Screen*

A tela inicial do aplicativo, também chamada de *Splash Screen* é utilizada apenas para introduzir o aplicativo ao usuário. Para o aplicativo *SearchOnMath*, foi utilizada uma abordagem simples contendo apenas o logo do aplicativo. Esta tela é visível para o usuário por 3 segundos e logo em seguida é iniciada a tela de busca.

3.5.1 Criando *Layout* da Tela Inicial

Foi utilizado nesta tela o *LinearLayout* com orientação vertical. Dentro deste *LinearLayout* foi definido um elemento *ImageView,* utilizado para inserir imagens em *Layouts xml.* Na Figura 11 a seguir é exibida a tela inicial do aplicativo *SearchOnMath:*

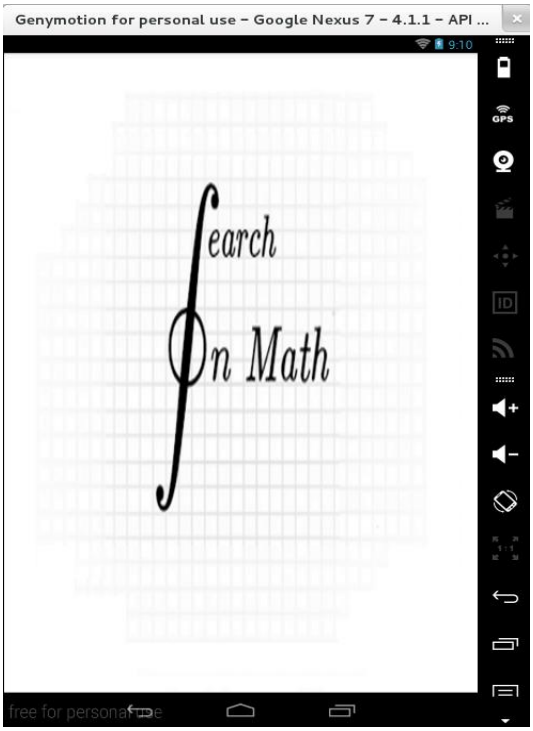


Figura 11: Tela inicial do aplicativo contendo o logo da *SearchOnMath.*

3.5.2 Criando classe da tela inicial

Para criar esta classe, recuperar os dados e exibir o *Layout xml* na tela, utilizou-se as definições conforme descrito na secção 3.2.2.

Dentro do método *onCreate()* para deixar a tela disponível para o usuário em apenas alguns segundos, foi criado um outro processo em execução. Dentro deste processo foi realizado a lógica para que o usuário visualizasse por apenas 5 segundos a tela inicial. Após isto, foi chamada a tela de pesquisa.

Para realizar a comunicação entre duas telas foi utilizado o objeto *Intent*. Neste *Intent* foi definido como classe atual e de destino, *InitialScreen* e *MainActivity* respectivamente. Após isto, foi chamado o método *startActivity*, passando como parâmetro o objeto *Intent* criado, iniciando a tela de pesquisa, *MainActivity*.

3.6 Configurando *WebView*, *MathJax* e chamada de scripts

3.6.1 *WebView*

A *WebView* é utilizada para exibir aplicações web ou apenas parte dela em aplicativos *Android*. A classe *WebView* estende da classe *View* e utiliza um motor de renderização WebKit para exibir páginas da web. Esta classe possui métodos para navegar em diferentes orientações e escalas, permitindo realizar pesquisas de texto. (DEVELOPER, 2015$_h$).

A *WebView* foi utilizada para exibir as equações em formato *TeX* inseridas pelo usuário e processadas com a *engine MathJax*.

3.6.2 *MathJax*

MathJax é um motor de *Javascript* para a exibição de fórmulas, equações e símbolos matemáticos através dos navegadores (MATHJAX, 2015$_a$). Utilizando este motor de exibição, é possível transformar um elemento de formato TeX em um elemento de formato *html/css*.

Este processamento gráfico foi necessário no aplicativo, pois a equação de entrada do usuário e consultada no *WebService* é em formato TeX.

3.6.3 Configuração da *WebView* com *MathJax*

O elemento *Webview* foi utilizado para exibir as equações em formato TeX criadas pelo usuário a partir do teclado. Para que isto fosse possível foi utilizado junto com a *WebView* um mecanismo de renderização de TeX chamada *MathJax*. Para a utilização do *MathJax* no aplicativo foi necessário realizar alguns processos importantes, conforme definidos por Krautzberger (2014):

- manter a cópia do *MathJax* no próprio aplicativo para uso *offline*, para reduzir a velocidade de carregamento;
- manter apenas os arquivos de imagens SVG - gráficos vetoriais escaláveis; para melhor perfomance;
- configurar o *MathJax* para utilizar a quebra de linha automática para pequenas telas de exibição;
- renderizar o conteúdo de forma dinâmica para melhorar a experiência do usuário.

Uma vez alocado a *engine MathJax* ao projeto e mantido os arquivos de imagens SVG inicia-se a etapa de configuração da *WebView* e da *engine MathJax.* Para a configuração da *WebView,* primeiramente foi utilizado o método *getSettings,* para obter uma instância que permita realizar configurações na *WebView*. Logo em seguida, a partir desta instância obtida, dois importantes métodos são chamados, sendo eles:

- *setJavaScriptEnabled(boolean)*: utilizado para permitir ou não a execução de Javascript na *WebView*, conforme o parâmetro *boolean*. Utilizou-se o parâmetro *true* no projeto, pois a *engine MathJax* renderiza TeX a partir de Javascript;
- *setBuiltInZoomControls(boolean):* utilizado para permitir ou não o zoom sobre a WebView, conforme o parâmetro *boolean*. Foi utilizado o parâmetro *true* para o projeto.

Para a adição de um *browser* para *WebView*, que permitisse de fato visualizar páginas web, é chamado o método *setWebChromeClient()* passando como parâmetro um objeto do tipo *WebChromeClient* que representa de fato o *browser.* Em seguida,

inicia-se a configuração do *MathJax* pela *WebView*. Para carregar os scripts do *MathJax* e os componentes *htmls* a serem exibidos na *WebView* foi chamado o método *loadDataWithBaseURL.*

Nas configurações realizadas dentro deste método, podem-se destacar os seguintes atributos:

- *showMathMenu*: especifica se deve ou não ser exibido o menu do *MathJax*. Para o aplicativo usou-se *false*;
- *jax*: indica qual será o tipo do elemento de entrada e qual será o tipo do elemento de saída. Definiu-se para o aplicativo os tipos Tex e HTML-CSS para os elementos de entrada e saída respectivamente;
- SVG (gráficos vetoriais escaláveis): usada para controlar as operações do processador de saída SVG. Dentre deste atributo definiu uma escala de 120, para otimizações de processamentos em dispositivos móveis com telas menores;
- *Script*: utilizado para carregar os scripts do *MathJax* e os demais utilizados no *browser* para exibir a saída em *html/css* de cada equação inserida como elemento de entrada TeX.

Um dos scripts carregados possuía uma *tag* <*span*> com o valor do id igual a "*math*". Para cada clique do usuário no teclado, era realizado a conversão do símbolo matemático em seu TeX correspondente. O *MathJax* processava esta equação TeX e inseria o *html/css* correspondente ao elemento de id com o valor de "*math*". A Figura 12 apresentada a seguir, exibe a tela de pesquisa que utiliza *WebView* e *MathJax* para exibir o TeX processado:

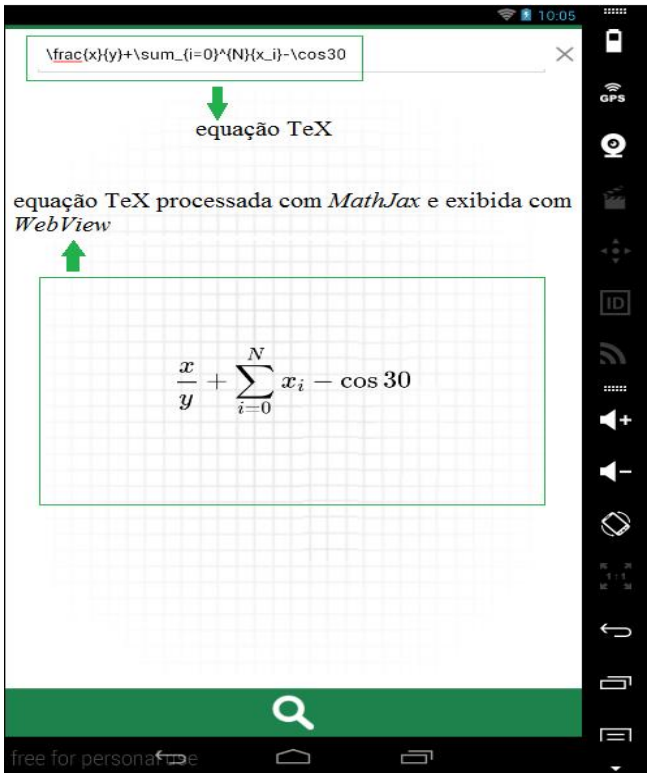


Figura 12: Tela de pesquisa destacando a equação TeX processada com *MathJax* e exibida com *WebView*.

3.7 Desenvolvendo Telas para inserção de Matriz

O teclado virtual personalizado permite que o usuário crie matrizes e busque por resultados de páginas na *web* que contenham referências a ela. O máximo de tamanho suportado são matrizes de 10 linhas por 10 colunas. Para que isto fosse possível, três telas são disponibilizadas para o usuário, sendo elas: telas para definição do tamanho da matriz, tela para a construção da matriz a partir do tamanho definido e tela de visualização da matriz inserida. As secções a seguir demonstram como foi desenvolvida cada uma das telas.

3.7.1 Desenvolvendo Tela para Escolha do Tamanho da Matriz

O objetivo principal desta tela é permitir que o usuário escolha o número de linhas e colunas da matriz a ser construída. Para isto, esta tela oferece quatro elementos importantes, sendo eles: um campo para inserir o número de linhas, um campo para inserir o número de colunas, um botão para que o usuário cancelasse a criação da matriz e um botão que permitisse o usuário criar a matriz a partir do tamanho já definido.

3.7.1.1 Criando *Layout* para a Tela de Escolha do Tamanho da Matriz

O *Layout* externo utilizado para a construção desta tela foi um *LinearLayout* com orientação vertical. Como primeiro elemento deste *Layout* criou-se um *TextView* que recebeu como texto o nome da tela '*size*'. Após este elemento, criou-se outro *LinearLayout*, com orientação horizontal. Dentro deste *LinearLayout*, dois elementos *EditText* foram criados para receber em tempo de execução o número de linhas e colunas por parte do usuário. Logo em seguida, um outro *LinearLayout* foi definido. Dentro deste último dois *Buttons* foram definidos, um contendo o texto '*cancel*', que possibilitava o usuário interromper a operação de criação da matriz e o outro contendo o texto '*load*' que permitiria o usuário ir para outra tela criar a matriz de acordo com o tamanho definido. O último elemento desta tela é uma *KeyboardView* utilizada para exibir o teclado desenvolvido quando solicitado pelo campo de texto.

3.7.1.2 Criando Classe para Exibir o *Layout* e Processar Eventos

Após o *Layout xml* criado, construiu-se uma classe que permitisse exibir este *Layout* e processar eventos relacionados aos cliques nos botões quando pressionados, conforme descrito na secção 3.2.2. Como grande diferencial desta tela, destacam-se os eventos relacionados a cada um dos botões.

Para criar a ação do evento relacionado ao botão '*cancel*', criou-se um objeto do tipo *onClickListener* e dentro do método *onclick* deste objeto fez-se uma chamada do método *finish()*, utilizado para sair da tela em questão e retornar a tela anterior, neste caso a tela de pesquisa. Logo em seguida, este objeto *onClickListener* foi adicionado ao botão *cancel* com o método *setOnClickListener*. Para criar a ação do

evento relacionado ao botão com texto '*load*' o mesmo processo foi realizado porém com ações dentro do método *onClick* diferentes. Dentro do método *onClick,* foi inicialmente criado um objeto do tipo *Bundle,* para encapsular as informações referentes aos números de linhas e colunas escolhidos. Estas informações eram inseridas no objeto *Bundle* pelo método *putInt("chave",valor),* passando como chave uma *String* que permitisse diferenciar os elementos ('*ROWS*','*COLUMNS*') e como valor os escolhidos pelos usuários.

Logo em seguida, um objeto do tipo I*ntent* foi criado. A classe *Intent* é responsável por realizar comunicações entre duas *Activities* (DEVELOPERS, 2015$_i$). Seu construtor possui como parâmetro a classe atual e a classe à qual deverá ser chamada, neste caso, *DialogMatrixSize* e *Matrix* respectivamente.

Por fim, o objeto do tipo *Bundle* foi inserido no objeto *Intent* criado, utilizando o método *putExtras()*.O método *startActivity()* foi então chamado passando o objeto *Intent* como parâmetro e iniciando a execução da nova tela pelo método *onCreate*() e a parada da tela atual pelo método *onPause*().

A Figura 13 a seguir apresenta o resultado final da tela que permite o usuário inserir o número de linhas e colunas para construir a matriz:

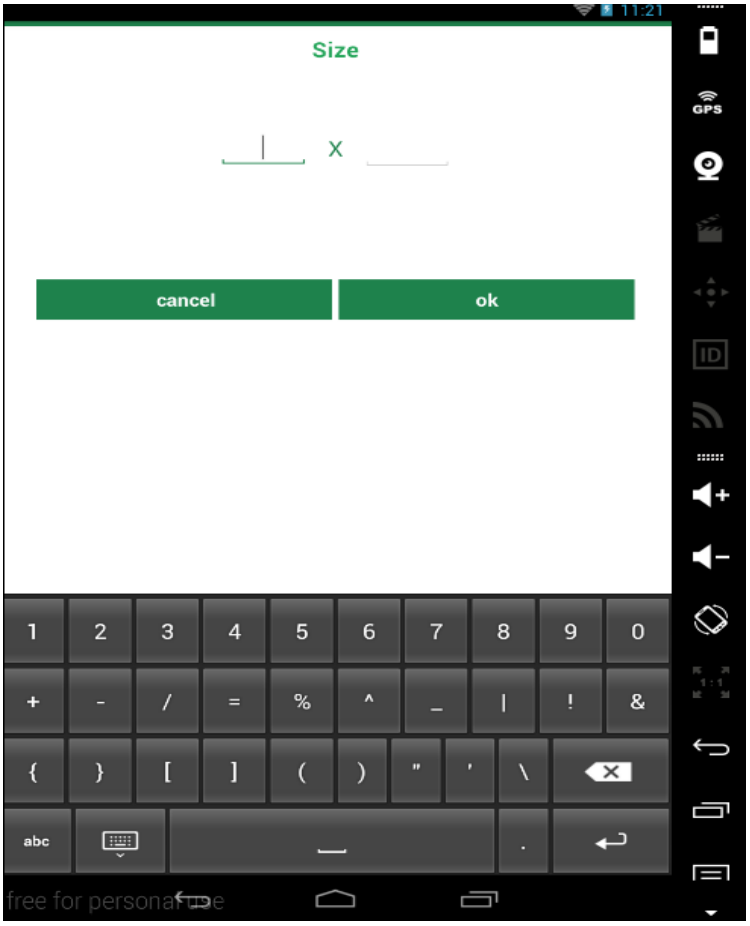


Figura 13: Resultado final da tela que permite o usuário inserir a dimensão da matriz.

3.7.2 Desenvolvendo Tela de Matriz

A tela para a criação da matriz permite que o usuário preencha os campos da matriz com quaisquer elementos disponíveis no teclado. Esta tela é criada dinamicamente a partir do número de linhas e colunas escolhidas na tela anterior. Esta tela apresenta ainda um botão para cancelar a operação e voltar à tela anterior e um botão para prosseguir com a operação.

3.7.2.1 Criando *Layout* para a Tela de matriz

Para o desenvolvimento desta tela foi utilizado o *LinearLayout* com orientação vertical. O primeiro elemento deste *LinearLayout,* é um *TextView* contendo um texto 'Matrix'. Logo em seguida, foi definido outro elemento para permitir a criação de

matrizes, chamado *TableLayout*. O *TableLayout* é um *Layout* que permite disponibilizar os dados em formato de tabelas a partir de um número de linhas e colunas já conhecidas. Após adicionado este elemento, um novo *LinearLayout* foi definido. Dentro deste *LinearLayout* foram colocados outros dois elementos do tipo *Button*. O primeiro *Button* apresenta o texto '*cancel*', utilizado para cancelar a operação de preencher a matriz, o outro *Button* apresentava o texto de '*preview*' permitindo o usuário passar para outra tela e visualizar a matriz inserida. Abaixo deste *LinearLayout* externo, foi definido um *KeyboardView* representando o teclado personalizado a ser exibido com um toque no campo de texto da matriz criada.

3.7.2.2 Criando Classe para a Tela de Matriz

O desenvolvimento desta classe seguiu os fundamentos apresentados na secção 3.2.2, para recuperação dos elementos e exibição do *Layout* definido na tela. Esta tela destaca-se pela construção dinâmica da matriz a partir do número de linhas e colunas preenchidas na tela anterior e recuperadas na tela atual. Para isto, inicialmente recuperou os valores de linhas e colunas. Isto foi possível recuperando o objeto *Bundle* a partir do método *getIntent().getExtras()* e a partir daí, recuperando os valores com o método *getInt("chave")*, passando como parâmetro a chave utilizada pela tela anterior ('*ROWS,COLUMNS*').

Em seguida, dois laços de repetição foram aninhados de maneira a preencher os elementos da matriz. Para o laço de repetição mais externo que percorre número de linhas vezes, é criado um objeto do tipo *TableRow* responsável por criar cada uma das linhas do *TableLayout.* Para o laço de repetição mais interno que percorre número de colunas vezes é criado um objeto do tipo *EditTex*, relacionado ao campo de texto, e adicionado à *TableRow* com o método *addView()*. Ao final da repetição, cada uma destas *TableRows* são adicionadas ao *TableLayout*, apresentando ao usuário uma matriz de campos de texto a serem preenchidos.

No método *onclick* da ação do botão *cancel* foi chamado o método *finish()*, que permite fechar a tela atual e voltar para a tela anterior.

No método *onclick* da ação do botão *preview* criou-se um objeto *Bundle* para encapsular as informações de cada elemento de texto preenchido. Este objeto *Bundle* foi adicionado a um objeto do tipo *Intent* que contêm como parâmetros de classe atual e classe de destino, *Matrix e Preview* respectivamente. A partir daí,

realizou a chamada do método *startActivity()* passando a *Intent* criada como parâmetro e iniciando a execução da tela *Preview*.

A Figura 14 a seguir exibe o resultado final da tela para a criação de uma matriz de 2 linhas por 2 colunas:

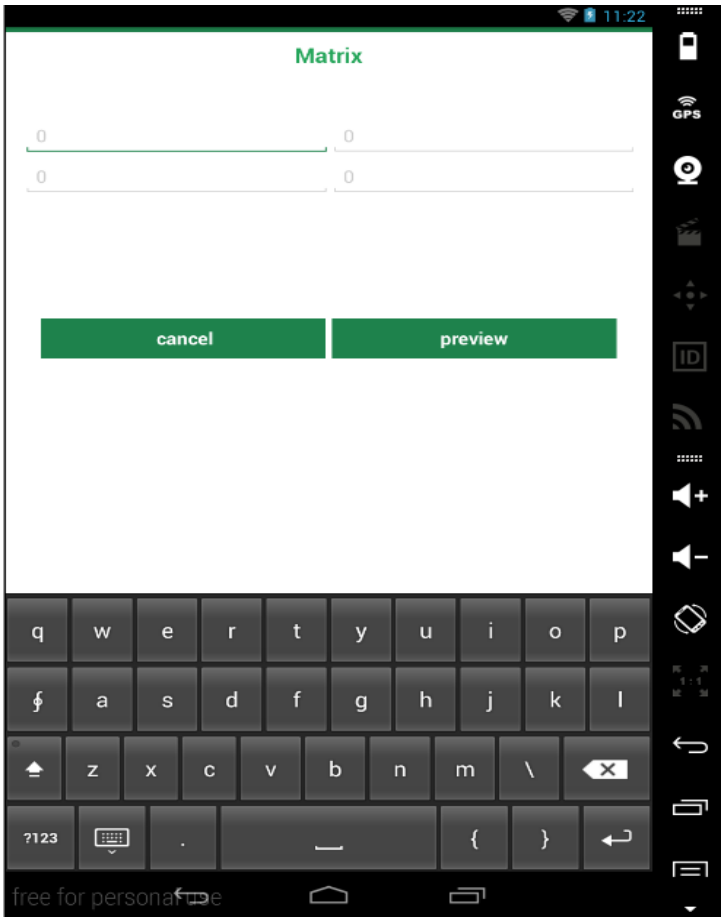


Figura 14: Resultado final da tela para a criação de uma matriz 2x2.

3.7.3 Desenvolvendo tela de visualização da matriz

Esta tela permite ao usuário visualizar a matriz inserida e realizar ações que permite editar a matriz, a partir do clique no botão *edit* ou enviar esta matriz para a tela principal, a partir do clique no botão *done.*

3.7.3.1 Criando Layout para Tela de Visualização da Matriz

Foi utilizado um *LinearLayout* com orientação vertical para esta tela. Dentro deste *LinearLayout* foi inserido um elemento de *WebView*, que permite carregar páginas *web* ou exibir conteúdos *htmls/scripts/css*. Logo em seguida, outro *LinearLayout* foi definido. Dentro deste *LinearLayout* interno foi definido dois elementos *Button*, um contendo o texto de *edit* e o outro contendo o texto de *done*.

3.7.3.2 Criando Classe para a Tela de Visualização da Matriz

Para criar a classe da tela de visualização da matriz seguiu os preceitos indicados na secção 3.2.2. Tem-se como particularidade desta tela a exibição na *WebView* da matriz construída na tela anterior. Para isto, inicialmente configurou a *WebView* com o motor de exibição *MathJax*, conforme descrito na secção 3.4.3. Em seguida é recuperado o TeX da matriz preenchidas na tela anterior com o objeto *Bundle.*

Logo em seguida, foi chamado o método *evaluateJavascript* da *WebView,* passando como parâmetro o script a ser executado. Este *script* é responsável por inserir na *tag* com o id *math* o processamento do TeX pelo *MathJax*, que é um elemento *html/css*.

No método *onclick* do botão *edit,* foi chamado o método *finish(),* para retornar a tela anterior de preenchimento da matriz.

No método *onclick* do botão done, foi criado um objeto *Bundle* para encapsular as informações do TeX correspondente a matriz. Este objeto *Bundle* foi adicionado a um objeto do tipo *Intent* que contém como parâmetros de classe atual e classe de destino, *Preview e MainActivity* respectivamente. A partir daí é realizado a chamada do método *startActivity()* passando a *Intent* criada como parâmetro e iniciando a execução da tela *MainActivity.*

A Figura 15 a seguir exibe o resultado final da tela da tela de visualização da matriz:

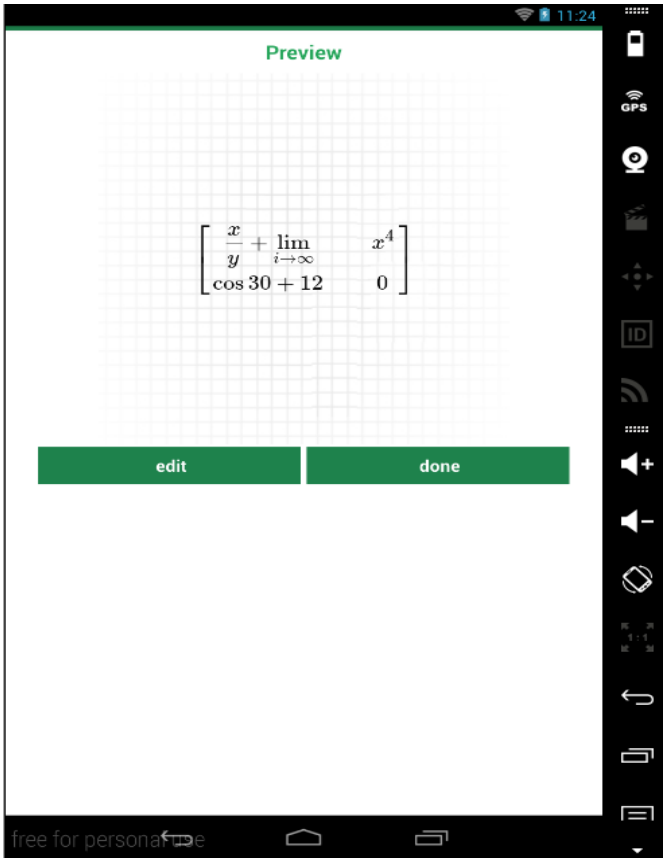


Figura 15: Tela de visualização da matriz 2x2.

3.8 Desenvolvendo Tela Pesquisa

A tela de pesquisa, ou tela de busca, é a tela responsável por permitir a entrada de equações no formato TeX e realizar a busca desta equação. Basicamente a tela de busca é formada por quatro componentes de *Views*: um *EditText*, uma *WebView*, um *Button* e um *KeyboardView*.

O *EditText* é o componente de texto editável que permite aos usuários inserirem texto no aplicativo. Além de inserir texto ele também permite colar, copiar e recortar o texto. Quando o usuário toca em um *EditText,* o teclado do app é suspenso permitindo ao usuário digitar o texto (DEVELOPERS, 2015$_j$).

A *WebView* é uma *View* que permite exibir páginas *web* em aplicativos, conforme descrito na secção 3.6.1.

Um *Button* é um componente utilizado para executar uma ação ao ser pressionado ou clicado pelo usuário, neste caso, para o momento em que o usuário for realizar a consulta da equação ao *WebService*.

Um *KeyboardView* é o elemento que é exibido na tela do aplicativo no momento que o usuário toca o elemento *EditText,* permitindo que o usuário digite um texto. Para o aplicativo, um novo *Keyboard* foi desenvolvido conforme descrito na secção 3.2. Esta *keyboardView* foi utilizada para todas as telas do aplicativo que tinham entrada de texto.

A construção de telas do aplicativo *SearchOnMath* foi realizada primeiramente criando os *Layouts* em *xml*, sendo que estes continham as referências dos componentes de *View*. Logo em seguida foi criada uma classe Java, que recuperavam estes elementos do *Layout* em *xml* pela criação de objetos e adiciona a eles as ações realizadas pelo usuário.

3.8.1 Desenvolvendo *Layout* em *xml* para a Tela de Pesquisa

Para a criação da tela de busca utilizou um *RelativeLayout* em combinação com *LinearLayout.*

A tela de busca foi criada com um *Layout* externo de *RelativeLayout.* Dentro deste *RelativeLayout* tem um *LinearLayout* com o campo *orientation* igual a vertical. Isto indica que os elementos filhos dentro deste *Layout* serão disponibilizados um embaixo do outro, verticalmente. Dentro deste *Layout*, estão outros dois *LinearLayouts* filhos.

O primeiro *LinearLayout* filho possui orientação horizontal. Neste *Layout* é colocado o elemento de tela *EditText,* para permitir o usuário editar sua equação.

O segundo *LinearLayout* filho possui um elemento de *WebView,* utilizado para processar graficamente as equações no formato TeX inserida no *EditText,* e um *Button,* utilizado para iniciar a consulta da equação ao *WebService*.. Abaixo do *LinearLayout* que está contido estes dois *LinearLayouts* filhos, foi criado um

elemento *KeyboardView* utilizado para que o usuário pudesse digitar as equações, conforme definido na secção 3.2.

Os elementos de tela são definidos com vários atributos de configuração, sendo um dos principais o id, que confere uma identidade única a cada elemento de interface gráfica.

3.8.2 Desenvolvendo Classe de *User Interface* para Tela de Pesquisa

Após criado o *Layout* contendo os componentes da tela em um arquivo *xml* foi necessário criar uma classe que permita exibir este *Layout*, recuperar cada um desses componentes em objetos e adicionar ações aos componentes, caso haja. Para isso foi realizada as operações definidas na secção 3.3.2.

O próximo passo foi definir configurações da *WebView* e realizar ações para cada um destes elementos conforme a interação com o usuário. Ao pressionar uma tecla do teclado, a letra ou símbolo correspondente à tecla pressionada é inserida no campo de texto. No aplicativo *SearchOnMath*, a cada tecla pressionada, além de ser mostrada no campo de texto, ela deve também ser processada graficamente no elemento *WebView* definido. Sendo assim, foi realizado a configuração da *WebView* com o motor de exibição *MathJax* conforme descrito na secção 3.4.3.

Logo em seguida, foi chamado o método *evaluateJavascript()* da *WebView,* passando como parâmetro o *script* a ser executado. Este *script* foi responsável por inserir na *tag* com o id *math* o processamento do TeX pelo *MathJax*, que é um elemento *html/css*.

3.8.3 Adicionando eventos ao Botão de Busca

Para adicionar eventos ao botão de busca da tela principal utilizou-se o método *setOnClickListener* do objeto Button. Este método recebe como parâmetro um objeto do tipo *OnClickListener* responsável por realizar as ações definidas e implementadas no método *onClick*. Sendo assim criou-se um objeto *onClickListner* e dentro do método *onClick* implementou as seguintes ações:

- recuperação da equação em formato Tex presente no campo EditText, a partir do método getText().toString deste objeto;
- envio desta informação para a página de resultados.

Para o envio de informações de uma tela para outra, foi criado inicialmente um objeto responsável por realizar a comunicação entre duas *Activities*, que é chamado de *Intent.* Ao criar este objeto se define a classe atual e para qual classe se deverá ir, neste caso, classes *MainActivity* e *Results* respectivamente. Para encapsular a informação, foi criado um objeto do tipo *Bundle* - responsável por armazenar informações da *Activity* em execução, e inserida a informação com o seu método *putString().* Logo em seguida, este *bundle* foi adicionado ao objeto *intent* pelo método *putExtras* e chamado o método *startActivity* passando como parâmetro o objeto *intent* criado.

A Figura 16 abaixo exibe o resultado final da tela de pesquisa:

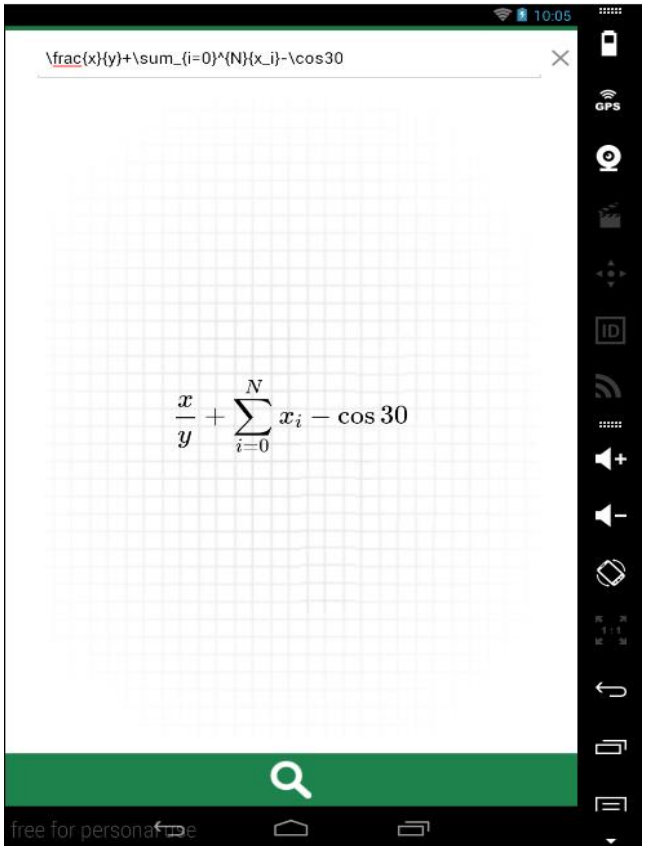


Figura 16: Tela de pesquisa para inserir equação em TeX, visualizá-la e consultá-la ao WebService.

3.9 Conectando o Aplicativo *SearchOnMath* ao *WebService*

3.9.1 Premissa para Eficiência e Performance em Conexões com a Internet

A conexão de um aplicativo à internet deve ser realizado de maneira a obedecer uma premissa necessária para a eficiência no consumo de dados, pois este consumo realizado de maneira errada pode levar a um gasto excessivo de memória e à consequente remoção do aplicativo por parte do usuário (MARIM, 2014). A premissa de extrema importância a ser utilizada é a de que trabalhos pesados e acesso a rede não devem ser executados dentro UI Thread - thread principal executada para exibir tela. Sendo as operações de rede bloqueantes, estas poderiam ocasionar travamentos no aplicativo e até mesmo levar a uma tela ANR (*Application Not Responding*).

Uma maneira de realizar este acesso de forma simples e eficiente é utilizar a classe *AsynkTask*. *Esta* classe que permite realizar operações em segundo plano e publicar os resultados na interface do usuário escondendo as complexidades de manipulação de *Threads*.

3.9.2 Clientes *Android* HTTP para Acesso a Internet

Grande parte dos aplicativos Android conectados à rede utilizam HTTP para envio e recebimento dos dados. *Android Http Client* e *HttpUrlConnection* são dois clientes HTTPs incluídos no Android, sendo que ambos dão suporte a *streaming* de *uploads* e *downloads*, configuração de limites de tempo, IPv6 e *pool* de conexão.

AndroidHttpClient é um cliente HTTP adequado para navegadores web. Pelo fato de possuir *APIs* grandes e flexíveis, este cliente tornou-se difícil para a equipe de desenvolvedores do Android melhorá-lo sem quebrar sua compatibilidade. Atualmente este cliente apresenta *bugs* e foi descontinuado pela equipe.

HttpUrlConnection é um cliente HTTP de propósito geral adequados para a maioria das aplicações (WILSON, 2011).

3.9.3 Biblioteca *Volley* para Transmissão de Dados em Rede

A bilioteca *Volley* foi desenvolvida para permitir a transmissão de dados em rede de forma simples e rápida. Dentre os benefícios de se utilizar esta biblioteca pode-se citar o fato de: realizar a programação automática de pedidos de rede, permitir várias conexões de rede simultâneas, oferecer a capacidade de poder dar prioridade a determinados pedidos, permitir cancelar pedidos de requisição já inicializados, utilizar memória cache para as respostas obtidas dentre outras.

Esta biblioteca utiliza como cliente HTTP o *HttpUrlConnection* para transmissão de dado. *Volley* não é utilizado para grandes operações de *download* ou *streaming*, pois ela armazena todos os dados em memória durante a análise (DEVELOPERS, 2015_K).

3.9.4 WebService

Para disponibilizar os resultados das buscas realizadas pelo aplicativo, um *WebService* foi desenvolvido. Para acesso a este *WebService* utiliza-se a url http://searchonmath.com/webservice/? que possui os parâmetros *equation*, relacionada a equação TeX a ser enviada para o *WebService* e *page*, relacionada ao número da página de requisição. Desta forma, para requisitar a página 1 que contém a equação TeX *'/sum'* para o *WebService* utiliza-se a url seguinte: http://searchonmath.com/webservice/?equation=%5Csum&page=1, sendo o parâmetro *equation* *'%5Csum'* equivalente ao TeX *'\sum'* convertido para um formato ASCII válido para envio em url.

3.9.4.1 Tipo de Dados Retornados pelo WebService

Definiu-se como retorno de dados do *WebService* e como consequente tipo de dados a ser consumido pelo aplicativo, o tipo de dados *JavaScript Object Notation*, ou popularmente conhecido *JSON*. *JSON* é um tipo de dados de intercâmbio leve, de

fácil leitura e escrita para os humanos e de fácil análise e criação por parte das máquinas. *JSON* é construído em duas estruturas, sendo elas: uma coleção de par nome/ valor normalmente criados como objetos por várias linguagens ou como uma lista ordenada de valores, criados como *arrays* ou matrizes pelas linguagens (ECMA INTERNATIONAL, 2013).

O JSON retornado pelo WebService da aplicação é formado por um objeto contendo os seguintes valores:

- *currentPage*: valor contendo uma indicação a página atual;
- *errorCode*: valor inteiro indicando se houve erro (1) ou não houve erro (0) durante a busca;
- *totalPages*: valor inteiro contendo o número total de páginas disponíveis;
- *totalResults*: valor inteiro contendo o número total de resultados disponíveis;
- *result*: um *array* de objetos contendo as informações da página de resultados.

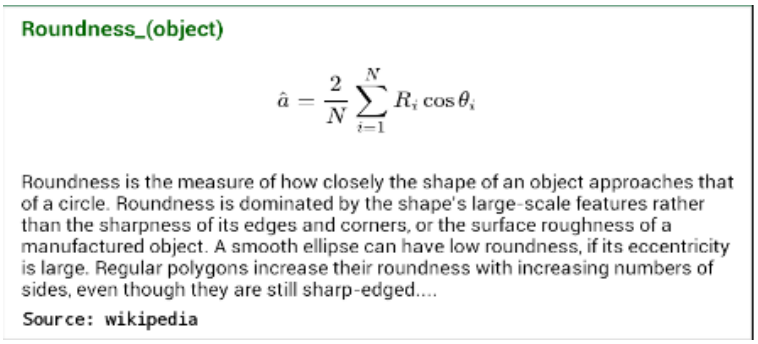
O *array* de objetos *result* possuía as seguintes informações relacionadas às páginas que continham a equação pesquisada pelo usuário:

- *abst*: valor contendo a descrição de referência da página que possui a equação encontrada;
- *equation*: a equação no formato TeX;
- *title*: valor contendo o título de referência da página que possui a equação encontrada;
- *url*: url do site que possui a equação encontrada como resultado da busca.

A Figura 17 exibe o formato de dados JSON retornados e uma ocorrência da lista de resultados criada com estes dados.

{"currentPage":1,"errorCode":0,
"result":[{"abst":"...","equation":"...","title":"...","url":"..."}],
"totalPages":10,"totalResults":100}

a



b

Figura 17: a - Formato de dados JSON retornados. b -*View* da lista de resultados contendo os valores retornados.

3.9.5 Conectando ao *WebService* e Realizando Análise de JSON

Para conectar o aplicativo *SearchOnMath* ao *WebService* foi utilizado a biblioteca *Volley*. Para acesso ao *WebService*, a biblioteca *Volley* utiliza inicialmente um objeto de *VolleyRequestQueue* para a inserir a requisição na fila de tarefas.

Com intuito de tornar mais rápida cada uma das requisições ao *WebService* criou-se uma classe *Singleton* (*VolleyRequestQueue.java*) que retornava uma instância de um objeto *RequestQueue*. Esta única instância do objeto de conexão era usada em todos os acessos ao *WebService* para cada usuário.

Logo em seguida, é criado um objeto do tipo *Response* utilizando uma interface *Listener* passando como classe de aceitação (*Generic*) um *JSONObject*. O método *OnResponse* da interface foi implementado. Este método é responsável por tratar a resposta da requisição realizada ao *WebService*. Como parâmetro ele recebe um objeto do tipo de dado esperado pela resposta, neste caso um objeto de *JSON* -

JSONObject. Os valores *currentPage*, *errorCode*, *totalPages*, e *totalResults* do objeto são recuperados através do método *getInt*("nome_do_valor") possuindo como parâmetro o nome do valor a ser recuperado. Estes valores recuperados são inseridos em uma estrutura de dados. O valor *results* é formado por um *array* de objetos contendo informações das páginas que contêm as equações. Este *array* foi recuperado com o método *getJSONArray*("nome_do_valor") possuindo como parâmetro o nome do *array* a ser recuperado, neste caso *results*.

Logo em seguida, este *array* é percorrido. Cada elemento do *array* é formado por um objeto de valores, sendo assim, foi feito um *cast* para (JSONObject) e a partir daí, os valores dos objetos são retornados e inseridos também em uma estrutura de dados para serem usados futuramente para criar a tela de resultados.

Após isto, um objeto de *Response.ErrorListener* é criado. Esta interface é responsável por tratar os erros que ocorrerem durante o acesso ao *WebService*. No método *onErrorResponse* desta interface, é simplesmente exibido o caminho *(trace)* do erro ocasionado.

Em seguida, é criado um objeto da classe *JsonObjectRequest*, responsável diretamente pela requisição ao *WebService*. Para criar este objeto, é passado como parâmetro: o tipo de método de requisição, *Get*, a url de acesso ao *WebService*, o objeto de *Response.Listener* e *Response.ErrorListener* já criados. Este objeto é inserido na fila de requisição de tarefas da biblioteca *VolleyRequestQueue*, para a realização da transmissão de dados entre WebService e aplicativo.

3.10 Desenvolvendo Tela de Resultados

Após recuperada as informações do *WebService* contendo os resultados da equação pesquisada pelo usuário, criou-se uma tela para apresentação destes resultados. A tela de resultados é composta por uma lista, sendo que cada elemento *(View)* da lista, contém basicamente cada um dos valores dos objetos *JSON* recuperados, tais como: um título da página que contém a equação encontrada, a exibição da equação TeX processada, uma descrição da página que contém a equação e o site

que a contém. Os resultados foram exibidos de dez em dez e controlados por paginação.

Existem várias maneiras de se criar uma lista de resultados para aplicativos *Android.* Dentre os mais usados destaca-se a utilização de *ScrollView* e *ListView. ScrollView* permite que o desenvolvedor crie *Layout* que possa se apresentar maior do que a tela física do dispositivo, uma vez que ele pode ser rolado verticalmente para a sua exibição(DEVELOPERS,2015l). *ListView* comporta-se de maneira parecida ao *ScrollView* com a diferença de que *Views* que compõem da lista advêm de uma outra estrutura chamada *Adapter* (DEVELOPERS, 2015m). O *Adapter* é responsável por receber os dados que vão compor cada um dos elementos da lista, criar um *Layout* para estes dados e inserir os elementos da lista.

3.10.1 Desenvolvendo Listas de Exibição de Resultados

Para a criação das listas de resultados do aplicativo *SearchOnMath* utilizou-se a *ListView*. A utilização da *ListView* possui uma grande vantagem em relação a *ScrollView* pois ela é reciclável. Ser reciclável significa que inserindo 50 elementos na lista, apenas os elementos visíveis na tela do dispositivo é que são criados, os outros elementos são criados a partir de outro que já deixou de existir, por exemplo, quando o usuário desliza a barra de rolagem (UDACITY, 2015). Este tipo de comportamento é o mais recomendável visando performance para grande quantidade de dados a ser apresentados.

3.10.2 Criando *Layouts* da Lista de Resultados

Cada elemento da lista de resultados é formado por um *Layout* previamente criado. Para a criação deste *Layout* utilizou-se um *LinearLayout.* Dentro desta estrutura de Layout foram colocados elementos de *View* que permite mostrar os dados recuperados do *WebService*.

O primeiro elemento de *View* do *Layout* foi um *TextView* utilizado para apresentar o título da página que contém o resultado. Logo abaixo foi colocado

uma *WebView* utilizada para exibir a equação no formato TeX processada em *html/css*.

Depois foi utilizado mais dois *TextViews,* sendo respectivamente utilizados para apresentar a descrição do site e apresentar o nome do site que continha a equação encontrada. Todos estes elementos foram colocados verticalmente um abaixo do outro. Este arquivo criado possui a extensão *xml*. O pseudocódigo abaixo mostra como foi criado o elemento descrito (*list_results_content.xml*).

```
<LinearLayout ... >
        <TextView .../>
        <WebView .../>
        <TextView .../>
        <TextView .../>
</LinearLayout>
```

3.10.3 Criando Padrão *Adapter*

O *Adapter* é o mecanismo utilizado para criar o *Layout* definido anteriormente contendo os dados recuperados do *WebService* e inserir este *Layout* como um elemento na lista de exibição dos resultados *(ListView)*. Primeiro é criado uma classe *ResultsAdapter.java*. Para que esta classe se comporte como um *Adapter*, ela precisa estender da classe *BaseAdapter*. *BaseAdapter* é uma superclasse comum de implementação para *Adapter* em *Android* (DEVELOPER, 2015n). No construtor desta classe foi utilizado como parâmetro uma estrutura de dados que contém os resultados recuperados do *WebService*. A estrutura de dados utilizada foi uma *ArrayList* de *HashMap*. Cada elemento do *ArrayList* equivale a um resultado recuperado e o *HashMap* armazena cada um dos valores do objeto JSON de retorno (*abst, title, equation,url*). O principal método implementado nesta classe foi o *getView.* Este método é responsável por criar o *Layout* que irá conter os dados recuperados pelo *WebService*, utilizando o *Layout* em *xml* descrito anteriormente; e recuperar cada elemento de *View* do *Layout xml*, como por exemplo *TextView* e *WebView*. Para cada um desses elementos são atribuídos o seu valor correspondente vindo da estrutura de dados. Após construído este *Layout* ele é exibido como um elemento da *ListView.*

Para cada *View* da lista de exibição é adicionado uma ação ao pressionar este elemento. A ação a ser executada é redirecionar o usuário para a tela do site que contém a equação desejada. Para isto foi criado um objeto do tipo *Intent,* passando como parâmetro: o atributo *Intent.ACTION_VIEW,* indicando que se trata de uma intenção de visualização; e a *url* do site.

Para criar cada elemento da lista, o método *getView* é chamado.

3.10.4 Adicionando o *Adapter* à *ListView*

Depois que cada um dos elementos que compoem a lista estão prontos, é necessário preencher cada um destes elementos.

Inicialmente faz-se com que a classe que irá apresentar os resultados estenda da classe *Activity* - sendo assim, esta classe será reconhecida com uma tela. Logo em seguida, obtem-se uma referêcia do *Layout* em *xml* relacionado a *ListView* - o objeto *ListView* é criado. Em seguida, é criado os elementos que irão compor a lista. Para isto, um acesso ao *WebService* é realizado e cada resultado é inserido em uma posição da estrutura de dados. Cria-se um objeto do tipo *Adapter* passando a estrutura de dados dos resultados como parâmetro. Após isto, os elementos da lista estão criados. Para que a lista receba estes conteúdos usa-se um método da *ListView*, *setAdapter* para inserir na lista o *Adapter* criado. Logo depois, a lista já é capaz de estar populada com todos os elementos construídos a partir dos resultados obtidos do *WebService.*

A Figura 18 a seguir apresenta a estrutura utilizada para a criação da lista de resultados. À esquerda possui uma estrutura de dados contendo os resultados retornados pelo W*ebService.* No centro, o padrão *Adapter,* criando *Layouts* com os dados da estrutura de dados passados para seu construtor e exibindo este *Layout* na *ListView.* À direita, possui a *ListView* contendo os *Layouts* inflados pelo *Adapter.*:

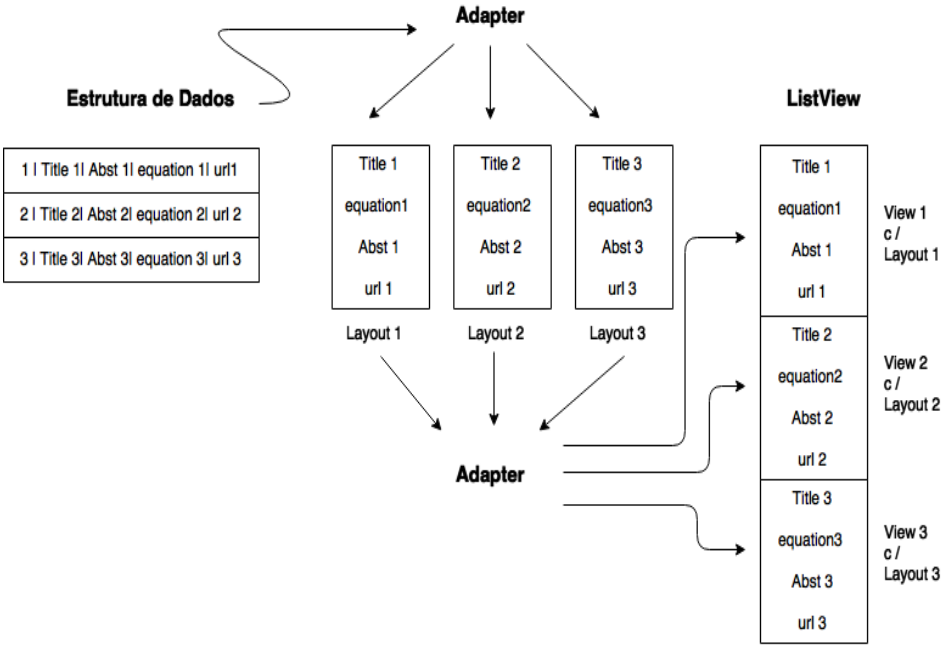


Figura 18: Estrutura utilizada para a criação da lista de resultados.

3.10.5 Criando Paginação para Controle de Dados

Para controlar a quantidade de dados retornados na lista de resultados, utilizou-se um mecanismo de paginação. Com este mecanismo foi definido que a cada acesso ao *WebService*, 10 resultados seriam retornados. Para implementar esta solução utilizou-se inicialmente um método da *ListView*, necessário para adicionar elementos de *View* no rodapé da lista, *addFooterView*. Como parâmetro para este método foi passado a instância de um objeto *Button*, que ao ser pressionado, fazia uma nova conexão ao *WebService*, e reconstruía a lista com os resultados adicionados.

Na Figura 19 abaixo apresenta-se o resultado final da tela de resultados a partir de uma consulta realizada, contendo os campos título, equação, descrição e o site que contém a página com a equação:

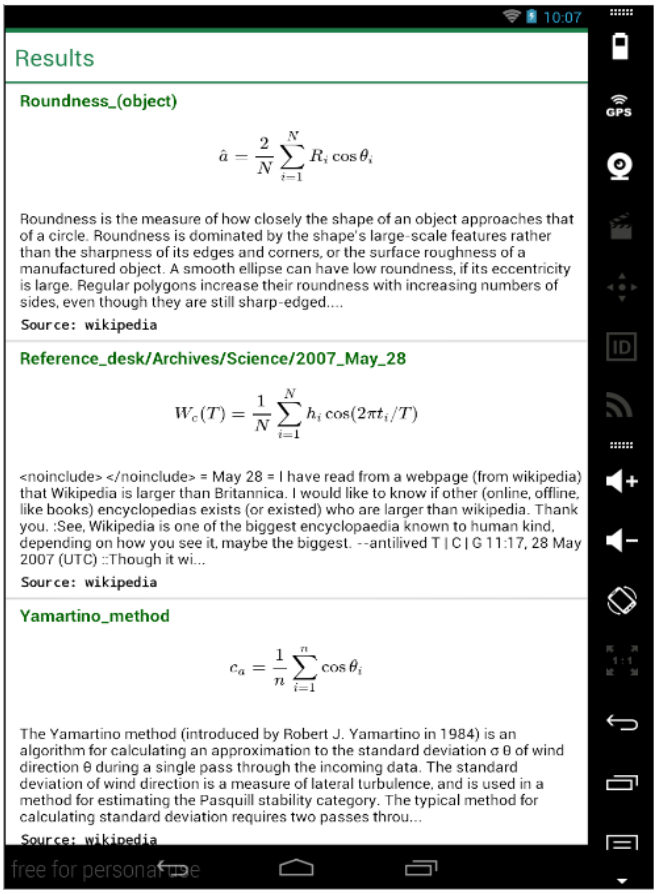


Figura 19: Tela de resultados disponibilizado em listas.

4 Resultados

Este capítulo apresenta os resultados obtidos do projeto realizado. Na secção 4.1 é apresentado o teclado desenvolvido, na secção 4.2 é apresentado o comportamento do aplicativo em uma consulta real e na secção 4.3 discute-se aspectos de experiência de usuário presentes no aplicativo.

4.1 Teclado Personalizado

Para oferecer ao usuário facilidade na construção de fórmulas matemáticas de formato TeX foi criado um teclado contendo os símbolos matemáticos (Figura 7).

A primeira linha deste teclado permite acessar a novos teclados (Figura 20). Nesta Figura é exibido: A - o teclado correspondente a 1ª linha, B – teclado correspondente a operadores mais utilizados, C – teclado contendo operadores de funções utilizados em cálculo, E – teclado contendo operadores relacionais e F – teclado contendo operadores de setas.

A segunda linha do teclado de subconjuntos matemáticos (Figura 21), permite ao usuário navegar por teclados: B – contendo letras gregas, C – contendo operadores de conjuntos, D - contendo funções trigonométricas e E – contendo operadores lógicos:

A terceira linha do teclado de subconjuntos matemáticos permite ao usuário navegar por novos teclados contendo: B – matriz, C - operadores de funções superiores e D operadores de funções específicas (Figura 22).

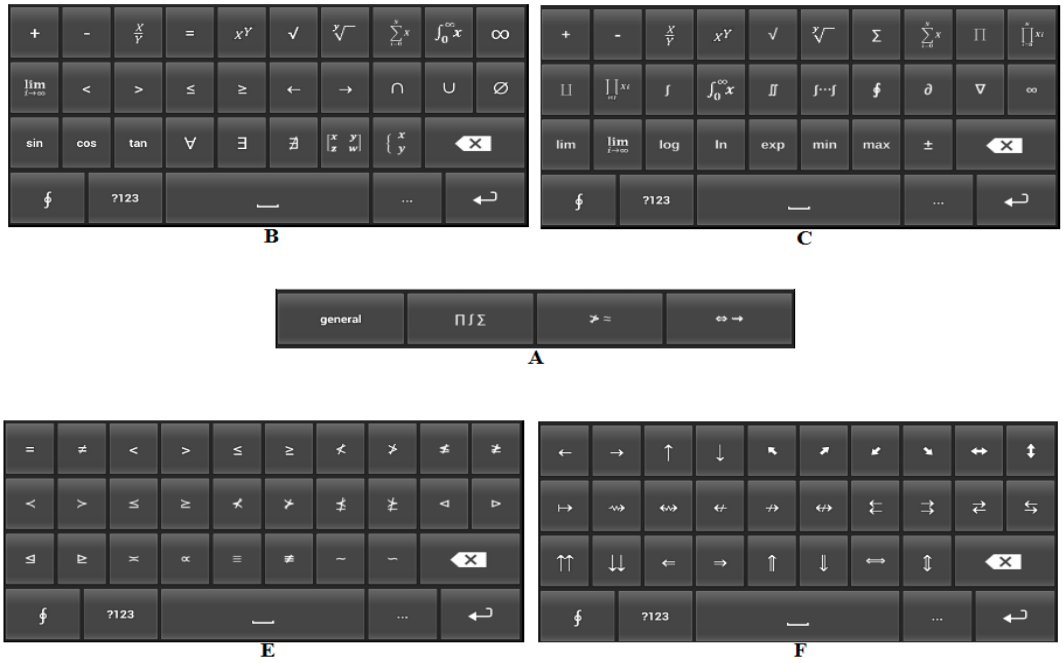


Figura 20: Primeira linha do teclado de subconjunto matemático e seus novos teclados.

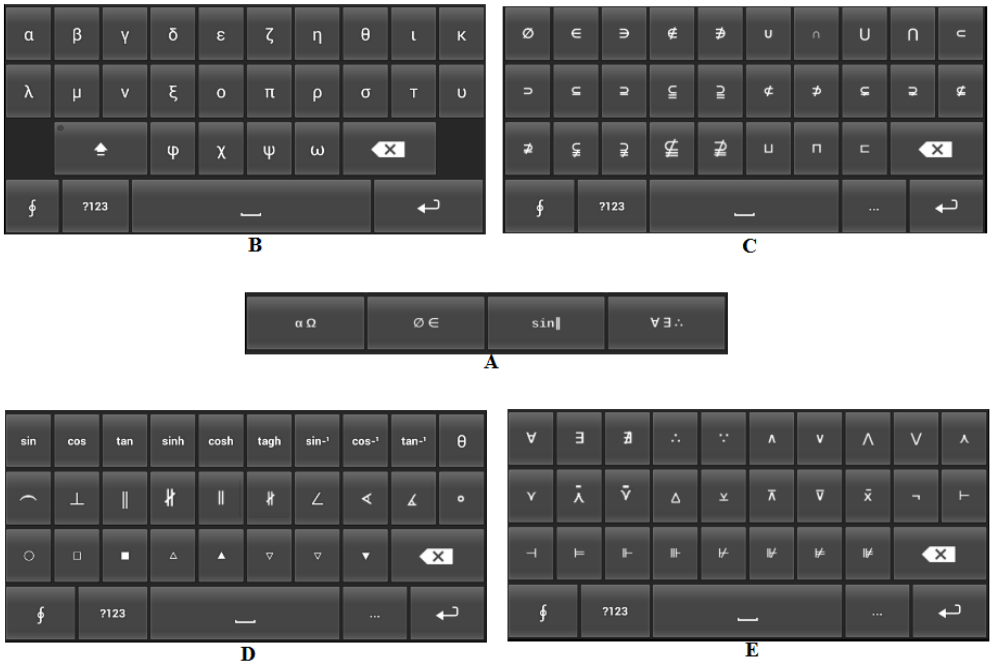


Figura 21: Segunda linha do teclado de subconjunto matemático e seus novos teclados relacionados.

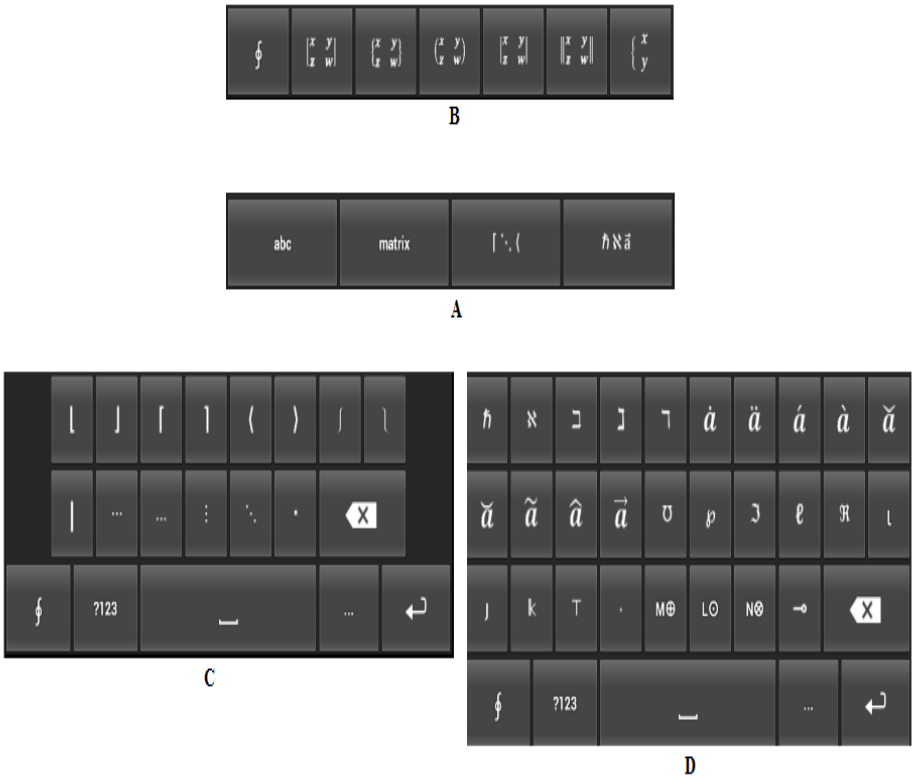


Figura 22: Terceira linha do teclado de subconjuntos matemáticos e eseus novos teclados relacionados.

Percebe-se que para cada novo teclado criado existe teclas que permitem o retorno para o teclado inicial facilitando a navegabilidade e a construção de fórmulas.

4.2 Pesquisa Realizada

A demonstração dos resultados é feita com base em uma busca da equação 1 gerada pelo assistente de fórmulas da ferramenta na versão *web(Try Me!)*. Este assistente gera automaticamente fórmulas para que o usuário realize testes.

$$\ln(1+x) = \sum_{k=0}^{\infty} \frac{(-1)^{(k+1)} x^k}{k} \qquad (1)$$

Inicialmente é criada o 1º membro da equação (membro antes do sinal de igualdade) partindo do novo teclado a - *'plustimes'*. A equação é criada pressionando cada uma das teclas conforme a sequência de números. Utiliza-se também os teclados b –numérico e c – *qwerty*. apresentadas na Figura 23 abaixo:

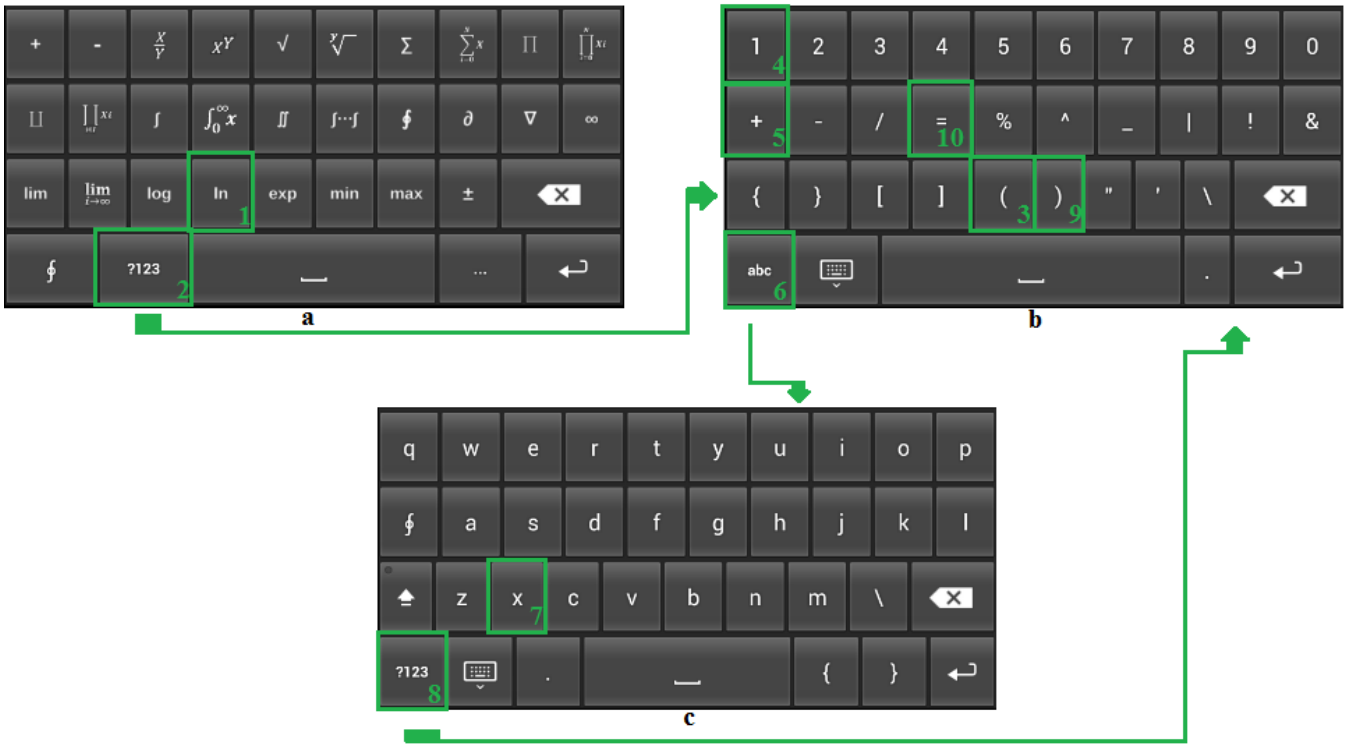


Figura 23: **Sequência de teclas pressionadas para criar o 1º membro da equação.**

Percebe-se que ao pressionar a tecla *'?123'* no teclado *plustimes,* o teclado numérico é exibido para o usuário, permitindo inserir o termo correspondente à função *ln*. Ao pressionar a tecla *'abc'* o teclado *qwerty* é exibido.

Logo em seguida é criado o 2º membro da equação. De maneira geral, é utilizada a sequência de teclas descritas na Figura 24. A pressionar a tecla 1 da sequência, é criado o símbolo de somatório apresentado em 2:

$$\sum_{x=0}^{N} x \qquad (2)$$

A variável x é modificada para k e o limite superior é modificado para infinito. Depois é modificado o termo do somatório, para isto é pressionado a tecla 2 da sequência, gerando uma fração, conforme apresentado em (3):

$$\frac{x}{y} \tag{3}$$

A partir daí, cria-se o 1º termo do numerador, pressionando a tecla 3 da sequência, para criar elementos da potenciação, conforme apresentado em 4:

$$x^{y} \tag{4}$$

Para a base da potenciação é utiliza-se -1 e como expoente utiliza-se a equação apresentada em 5:

$$k+1 \tag{5}$$

A continuação da construção do numerador e denomidor são utilizados elementos básicos do teclado numérico e *qwerty.*

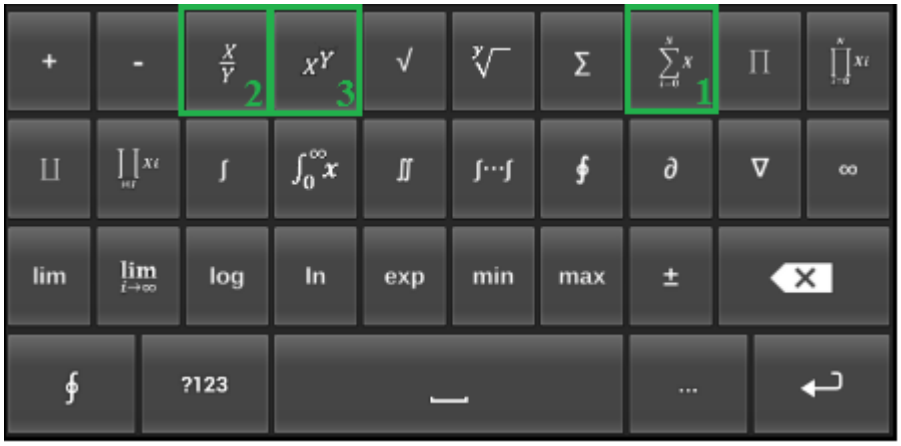


Figura 24: Sequência de teclas pressionadas para construir o 2º membro da equação, de maneira geral.

Simultaneamente com cada tecla pressionada, a tela de pesquisa está atualizando o campo de texto com a equação em formato TeX e a exibindo em um formato processado, conforme apresentado na Figura 25.

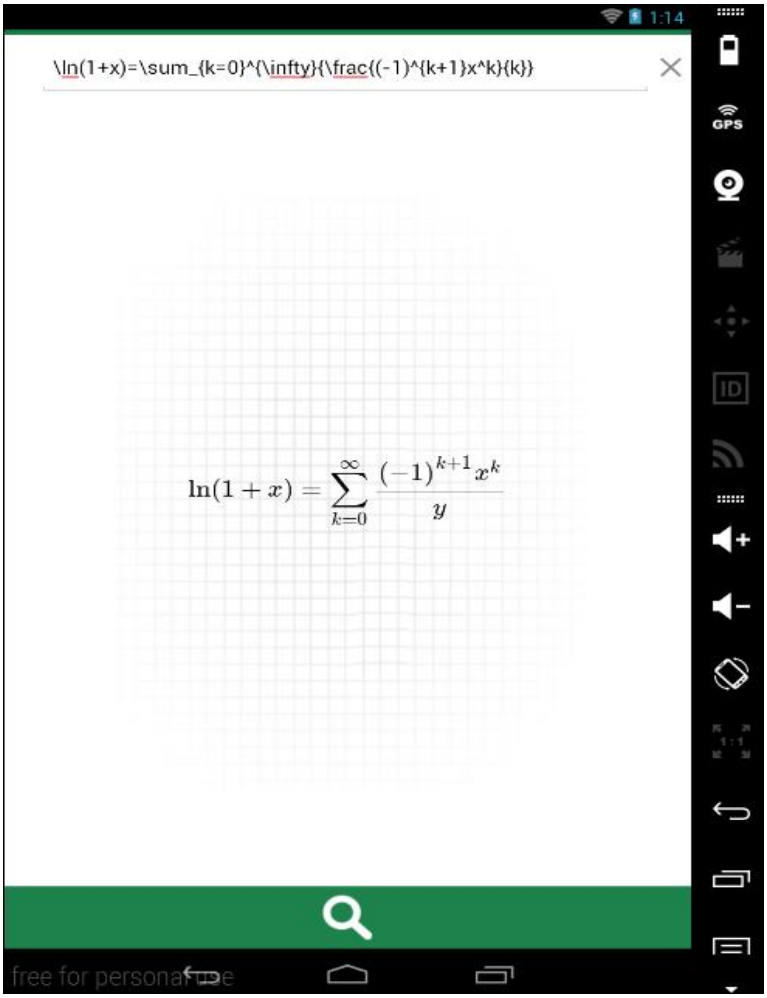


Figura 25: Equação em formato TeX criada e processada na tela de pesquisa.

Após criada a equação em formato TeX e pressionado o botão de pesquisa é realizado uma busca ao *WebService.* Depois de aproximadamente 4 segundos é exibida a tela de resultados da página 1 contendo 10 resultados.

Na Figura 26 é apresentado os três primeiros resultados desta 1ª página.

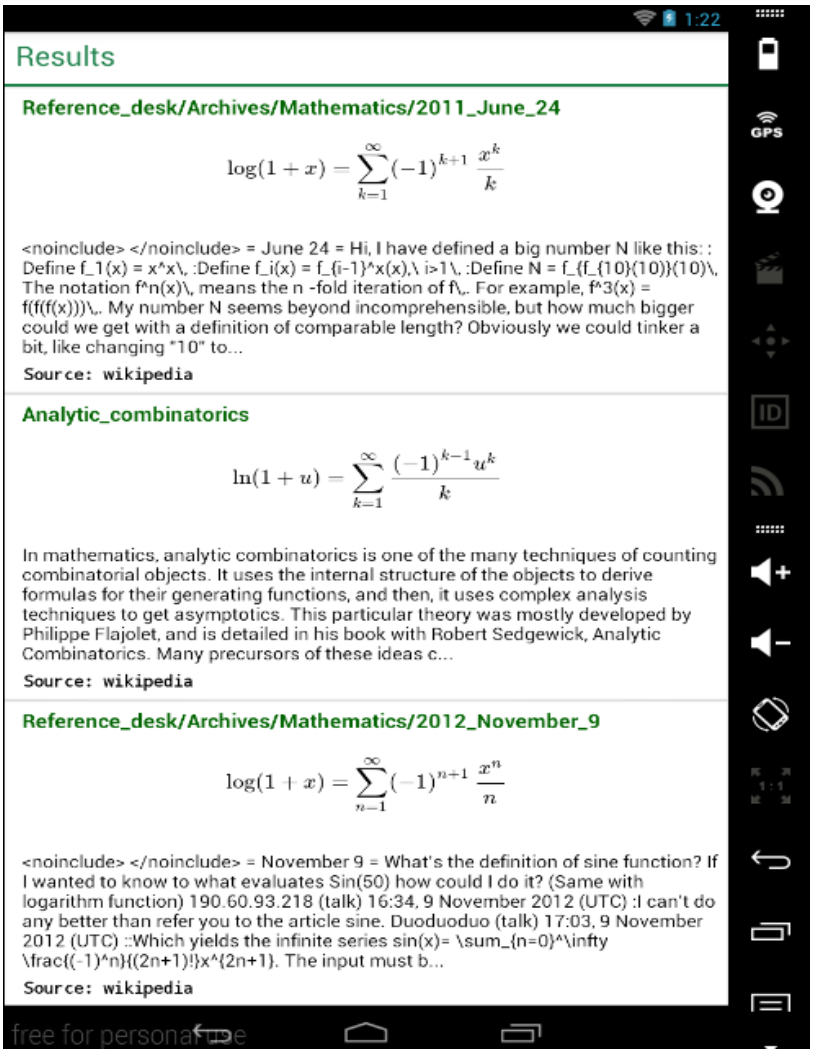


Figura 26: Tela de resultados para a pesquisa da equação realizada.

Ao realizar uma busca na *SearchOnMath* versão *web* contendo a mesma fórmula matemática apresentada em (1), comprova-se que ambas as versões, *web* e mobile para *Android* desenvolvido, apresentam os mesmos resultados. Na Figura 27 é exibido os três primeiros resultados da 1ª página de uma consulta da ferramenta na *web*.

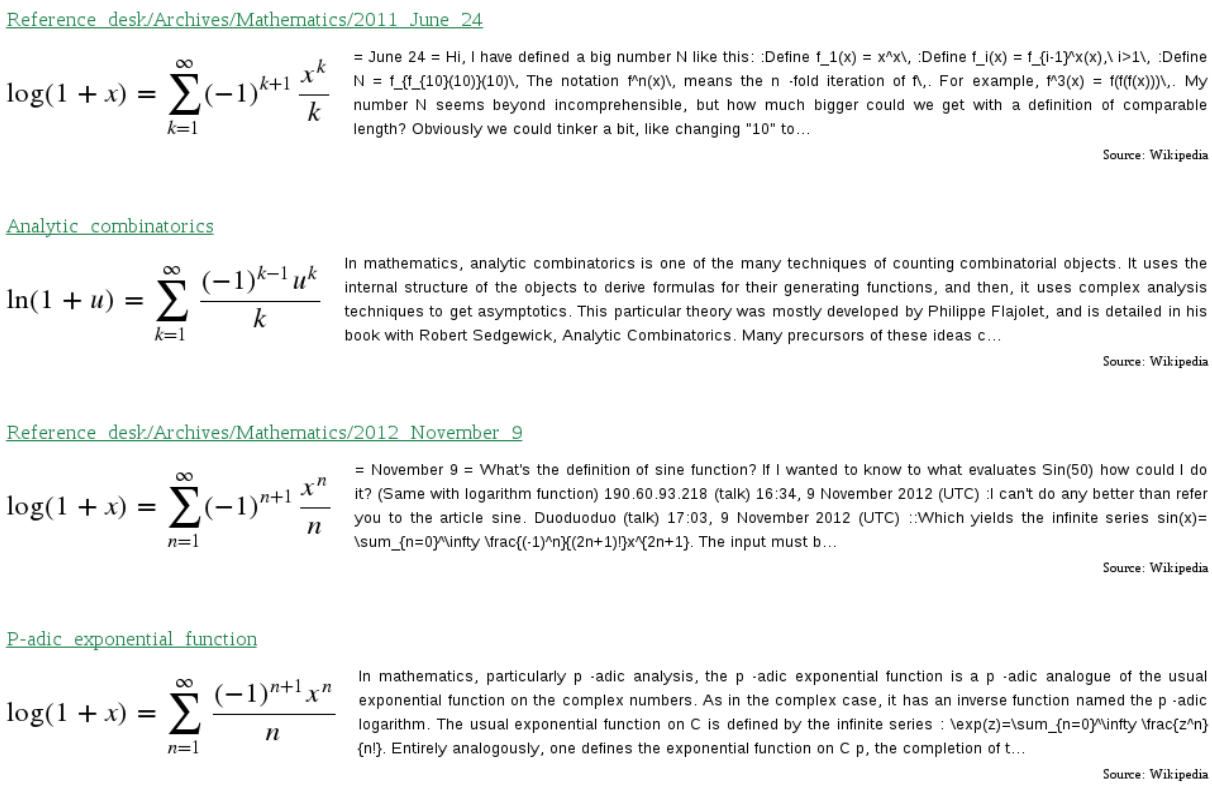


Figura 27: 4 primeiros resultados apresentados após uma consulta da fórmula (1) na *SearchOnMath* versão *web*, comprovando a igualdade dos resultados com o aplicativo desenvolvido.

Ao pressionar algum elemento da lista de resultados o usuário era redirecionado para uma tela do site que contém a equação pesquisada ou similar a ela.

4.3 Experiência do Usuário

4.3.1 Barra de Progresso

Um importante componente utilizado para garantir experiência do usuário foi a barra de progresso em formato circular. Ela foi exibida para toda operação que exige espera do usuário por uma operação do aplicativo. Na tela de pesquisa, ela é exibida a cada vez que o usuário pressiona o botão de pesquisa e o aplicativo inicia a busca de resultados no *WebService*. Na tela de resultados, a barra de progresso é

exibida a cada consulta de uma nova página de resultados ao *WebService.* A Figura 28 a seguir exibe a barra de progresso em ação na tela de pesquisa e na tela de resultados:

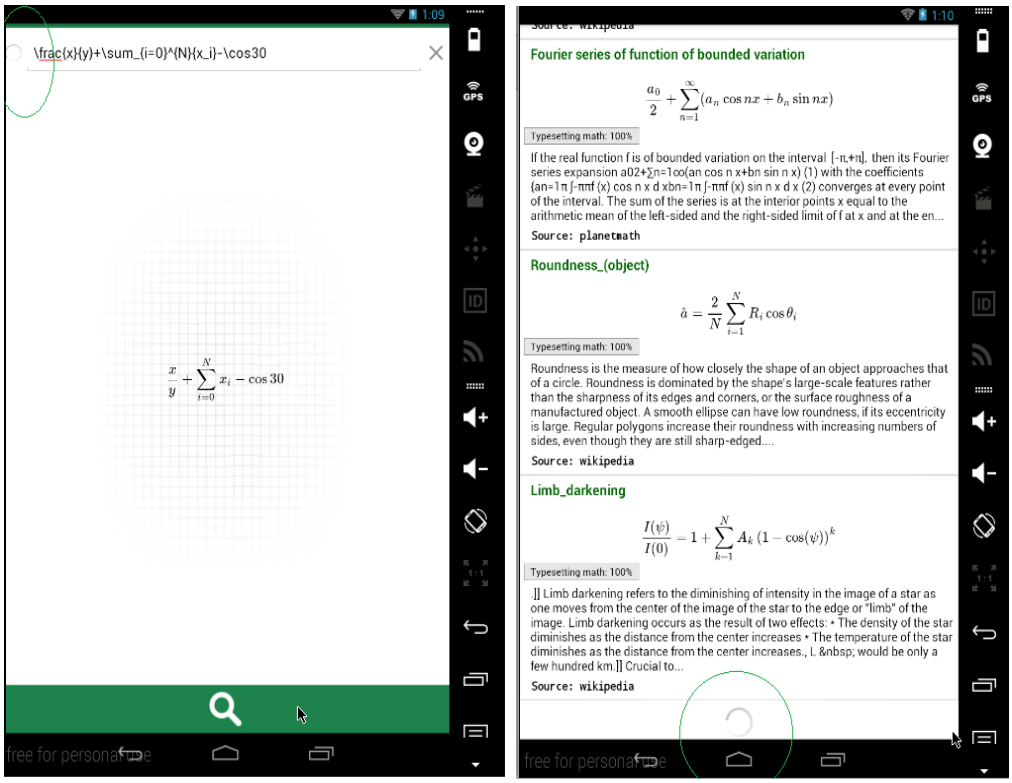


Figura 28: Barra de progresso em destaque na tela de pesquisa e tela de resultados.

4.3.2 Resposta Visual a Eventos

Outro importante recurso apresentado pelo aplicativo *SearchOnMath* visando melhorar a experiência do usuário foi a capacidade de alterar a cor das *Views* conforme o foco fosse modificado ou conforme alguma *View* fosse pressionada. Na Figura 29 é exibida a modificação da cor de um elemento da lista de resultados ao ser pressionado:

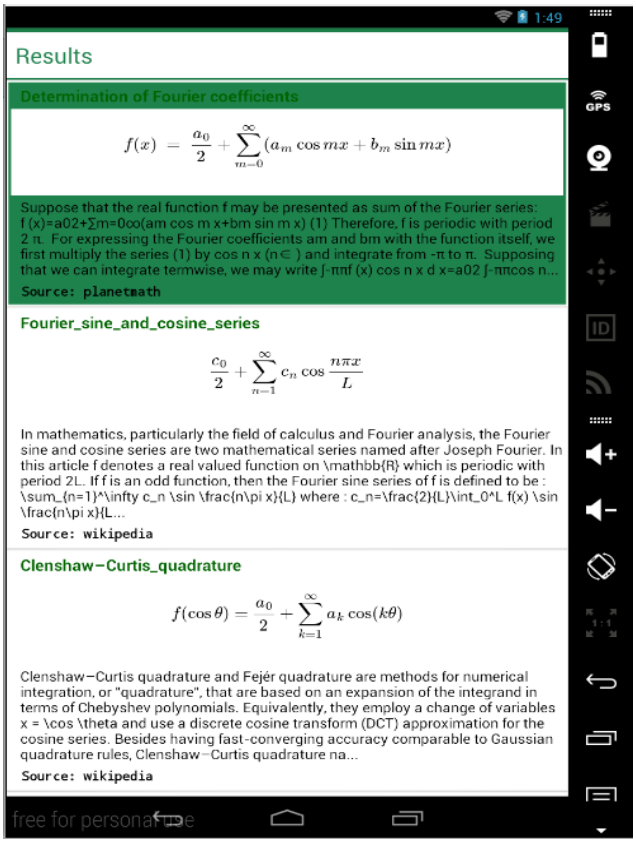


Figura 29: Alteração da cor da *View* ao ser pressionada.

4.3.3 Rápido Acesso a Símbolos Matemáticos

Uma característica importante do aplicativo *SearchOnMath* foi a facilidade na navegação e o rápido acesso aos símbolos matemáticos utilizados para construir uma fórmula matemática. O aplicativo permite que um máximo de duas teclas pressionadas seja o suficiente para o usuário ter acesso ao símbolo matemático escolhido e a partir daí, construir a fórmula. Este comportamento foi possível devido a utilização de teclas no teclado que permitem fácil navegabilidade de avanço e de retorno para novos teclados.

5 Conclusões e Trabalhos Futuros

Este capítulo apresenta algumas discussões envolvidas nesta dissertação. É organizado da seguinte forma: Na Seção5.1, é apresentada de forma sucinta a proposta, as principais limitações do projeto, os resultados alcançados e as vantagens do projeto, e na Seção 5.2 são discutidas propostas para trabalhos futuros

5.1 Conclusões

O propósito deste projeto foi o desenvolvimento de um aplicativo móvel para a ferramenta de busca por fórmulas matemáticas *SearchOnMath* disponível para a plataforma *Android.*

O aplicativo possibilita que o usuário, a partir de um dispositivo móvel com sistema operacional *Android,* construa sua fórmula matemática em formato TeX e pesquise na *web* por páginas que contenham referências a fórmula ou similares a ela.

Para a construção da fórmula matemática, o aplicativo oferece ao usuário a exibição de um novo teclado que contém aspectos já existentes nos teclados convencionais e também inovações de novas teclas. A principal inovação foi a existência de um novo teclado de subconjuntos matemáticos, que quando pressionado suas teclas, redireciona o usuário para um novo teclado contendo símbolos específicos do subconjunto. Possibilitando a construção de matrizes, o novo teclado oferece recursos para a construção de diferentes tipos. Para cada tipo de matriz é possível definir sua dimensão, construí-la com diferentes elementos e visualizá-la. Esta abordagem oferece ao usuário a facilidade na construção de fórmulas em formato TeX.

A partir da criação da fórmula matemática em formato TeX, o aplicativo disponibiliza ao usuário uma visualização amigável da fórmula criada, podendo inclusive adicionar *zoom* a visualização.

Para dar início à pesquisa pela fórmula criada, é disponibilizado na tela de pesquisa um botão *search,* que quando pressionado inicia a consulta ao *WebService* e recupera os dados referentes aos resultados.

Os resultados são apresentados aos usuários em formato de listas paginadas. Cada lista contém 10 resultados e para cada lista um novo acesso ao *WebService* é realizado. O resultado apresentado possui informações como o título, a descrição e a url do site que contém a fórmula matemática bem como a fórmula matemática ou alguma similar a ela. Quando pressionado um elemento da lista de resultados, o usuário é redirecionado para a página que contém a fórmula matemática.

Algumas vantagens obtidas a partir dos resultados são:

- Capacidade de criar diferentes fórmulas matemáticas em formato TeX de forma rápida através da disponibilização dos ícones para novos teclados;
- Permite visualizar a fórmula matemática em formato TeX de maneira amigável e processada;
- Permite manter o usuário consciente de cada ação realizada por ele ou pelo aplicativo, através de técnicas visando a experiência do usuário;
- Permite encontrar referências a fórmula matemática ou sua similar por toda web.

Uma limitação apresentada pelo aplicativo está relacionada ao novo método de entrada (IME) criado. Foi observado em testes com dispositivos reais, que ao selecionar o novo IME criado, apenas o aplicativo do *SearchOnMath* é que reconhecia o novo teclado criado, comportamento não esperado visto que ao trocar de IME todos aplicativos devem reconhecer o novo teclado.

5.2 Trabalhos Futuros

Tem-se como trabalhos futuros:

- Correções e reparos do novo método de entrada criado;
- Modificações em telas visando melhorar a experiência do usuário;
- Realização de teste de estresse visando encontrar *bugs* e *crashs;*
- Disponibilização do aplicativo *SearchOnMath* aos usuários através das lojas de aplicativos.

6 Referências Bibliográficas

ARVIND, Rangaswamy; C., Lee Giles; SERESB, Silvija. **A Strategic Perspective on Search Engines: Thought Candies for Practitioners and Researchers.** 2009. Disponível em: <http://clgiles.ist.psu.edu/pubs/JIM2009.pdf>. Acesso em: 03 set. 2014.

CHAVAN, Amey; NAIK, A. **Linear equation solver in Android using OCR.** IOSR Journal of Engeneering (IOSRJEN), p. 42-44, 5 may 3013.

DEVELOPERS. **Activities.** 2015_g. Disponível em: <http://developer.android.com/guide/components/activities.html>. Acesso em: 05 mar. 2015.

DEVELOPERS. **BaseAdapter.** 2015_n. Disponível em: <http://developer.android.com/reference/android/widget/BaseAdapter.html>. Acesso em: 11 mar. 2015.

DEVELOPERS. **Handling Keyboard Input.** 2015_a. Disponível em: <http://developer.android.com/training/keyboard-input/index.html>. Acesso em: 19 jun. 2015.

DEVELOPERS. **Intents and Intent Filters.** 2015_i. Disponível em: <http://developer.android.com/guide/components/intents-filters.html>. Acesso em: 7 maio 2015.

DEVELOPERS. **KeyboardView.OnKeyboardActionListener.** 2015_c. Disponível em: <http://developer.android.com/reference/android/inputmethodservice/KeyboardView.OnKeyboardActionListener.html>. Acesso em: 3 jun. 2015.

DEVELOPERS. **KeyboardView:** Class Overview. 2015_b. Disponível em: <http://developer.android.com/reference/android/inputmethodservice/KeyboardView.html>. Acesso em: 10 abr. 2015.

DEVELOPERS. **Layouts.** 2015_d. Disponível em: <http://developer.android.com/guide/topics/ui/declaring-layout.html>. Acesso em: 1 jun. 2015.

DEVELOPERS. **LinearLayout.** 2015_f. Disponível em: <http://developer.android.com/reference/android/widget/LinearLayout.html>. Acesso em: 4 fev. 2015.

DEVELOPERS. **ListView.** 2015_m. Disponível em: <http://developer.android.com/guide/topics/ui/layout/listview.html>. Acesso em: 11 mar. 2015.

DEVELOPERS. **RelativeLayout.** 2015_e. Disponível em: <http://developer.android.com/reference/android/widget/RelativeLayout.html>. Acesso em: 1 fev. 2015.

DEVELOPERS. **ScrollView.** 2015_l. Disponível em: <l http://developer.android.com/reference/android/widget/ScrollView.html>. Acesso em: 11 mar. 2015.

DEVELOPERS. **TextFields.** 2015_i. Disponível em: <http://developer.android.com/guide/topics/ui/controls/text.html>. Acesso em: 1 mar. 2015.

DEVELOPERS. **Transmitting Network Data Using Volley.** 2015_k. Disponível em: <https://developer.android.com/training/volley/index.html>. Acesso em: 24 mar. 2015.

DEVELOPERS. **WebView.** 2015_h. Disponível em: <http://developer.android.com/reference/android/webkit/WebView.html>. Acesso em: 02 fev. 2015.

EASYCOOKING.IN. **The greatest recipe search app ever.** 2012. Disponível em: <http://www.easycooking.in/>. Acesso em: 03 nov. 2014.

ECMA INTERNATIONAL. **The JSON Data Interchange Format:** Introduction. 2013. Disponível em: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>. Acesso em: 21 nov. 2014.

GONZAGA, Flávio Barbieri. **Recuperação de Informação Orientada ao Domínio da Matemática**. Universidade Federal do Rio de Janeiro, 2013.

GOOGLE PLAY. **Agri Precision - Agricultura.** 2015_a. Disponível em: <https://play.google.com/store/apps/details?id=agriprecision.pck>. Acesso em: 28 jun. 2015.

GOOGLE PLAY. **DuckDuckGo Search & Stories.** 2015_b. Disponível em: <https://play.google.com/store/apps/details?id=com.duckduckgo.mobile.android&hl=en>. Acesso em: 01 fev. 2015.

GOOGLE PLAY. **Image Search.** 2014. Disponível em: <https://play.google.com/store/apps/details?id=com.wagachat.imagesearch&hl=en>. Acesso em: 03 maio 2015.

GOOGLE PLAY. **Job Search Indeed Jobs.** 2015_d. Disponível em: <https://play.google.com/store/apps/details?id=com.indeed.android.jobsearch&hl=en>. Acesso em: 03 fev. 2015.

GOOGLE PLAY. **Math Formulas.** 2015e. Disponível em: <https://play.google.com/store/apps/details?id=com.whitesof.mathformulas&hl=en>. Acesso em: 29 jun. 2015.

GOOGLE PLAY. **Pre-Algebra.** 2012. Disponível em: <https://play.google.com/store/apps/details?id=com.yourteacher.prealgebra&hl=en>. Acesso em: 03 dez. 2014.

GOOGLE PLAY. **Search Engine - All Search.** 2015_c. Disponível em: <https://play.google.com/store/apps/details?id=com.devmine.en.smartsearch&hl=en>. Acesso em: 01 mar. 2015.

GOOGLE PLAY. **Symbolab.** 2015f. Disponível em: <https://play.google.com/store/apps/details?id=com.whitesof.mathformulas&hl=en>. Acesso em: 29 jun. 2015.

GOOGLE PLAY. **Travel Search Engine.** 2013. Disponível em: <https://play.google.com/store/apps/details?id=com.TravelSearchEngine&hl=en>. Acesso em: 12 set. 2014.

HATHIBELAGAL, Ashraff. **Create A Custom Keyboard on Android.** 2014_a. Disponível em: <http://code.tutsplus.com/tutorials/create-a-custom-keyboard-on-android--cms-22615>. Acesso em: 03 jun. 2015.

HATHIBELAGAL, Ashraff. **Create A Custom Keyboard on Android:** Create method.xml. 2014_b. Disponível em: <http://code.tutsplus.com/tutorials/create-a-custom-keyboard-on-android--cms-22615>. Acesso em: 12 abr. 2015.

HATHIBELAGAL, Ashraff. **Create A Custom Keyboard on Android:** Create a Service Class. 2014_c. Disponível em: <http://code.tutsplus.com/tutorials/create-a-custom-keyboard-on-android--cms-22615>. Acesso em: 2 jun. 2015.

IDC - ANALIZE THE FUTURE. **Estudo da IDC mostra recorde nas vendas de smartphones no terceiro trimestre de 2013.** 2013. Disponível em: <http://www.idcbrasil.com.br/releases/news.aspx?id=1547>. Acesso em: 02 maio 2015.

KHALAF, Simon. **Mobile Use Grows 115% in 2013, Propelled by Messaging Apps.** 2014. Disponível em: <http://flurrymobile.tumblr.com/post/115191226770/mobile-use-grows-115-in-2013-propelled-by#.U3iuBnVdW2U>. Acesso em: 18 jun. 2014.

KNUTH, Donald E.. **The TeXbook.** Stanford: Addison-wesley Professional, 1984. 496 p.

KRAUTZBERGER, Peter. **Guide: MathJax in Android apps.** 2014. Disponível em: <https://github.com/mathjax/MathJax-docs/wiki/Guide:-MathJax-in-Android-apps>. Acesso em: 10 nov. 2014.

LEE, Wei-Meng. Introdução ao Desenvolvimento de Aplicativos para Android. 1. ed. Rio de Janeiro: Editora Ciência Moderna Ltda, 2011.

MARIM, Neto. **Comunicação Web em apps Android - Parte I.** 2014. Disponível em: <http://codigo-google.blogspot.com.br/2014/02/comunicacao-web-em-apps-android-parte-i.html>. Acesso em: 10 nov. 2014.

MATHJAX. **Beautiful math in all browsers.** 2015_a. Disponível em: <http://www.mathjax.org/>. Acesso em: 12 mar. 2015.

MATHJAX. **The SVG output processor.** 2015_b. Disponível em: <http://docs.mathjax.org/en/latest/options/SVG.html>. Acesso em: 20 jun. 2015.

MEDSCAPE MOBILE. **The Medscape App is Now Better Than Ever.** 2015. Disponível em: <http://www.medscape.com/public/android>. Acesso em: 28 maio 2015.

ORACLE. **The Java EE 6 Tutorial:** What Are Web Services?. 2013. Disponível em: <https://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf>. Acesso em: 20 jun. 2015.

UDACITY. **Developing Android Apps:** ListView and Recycling. 2015. Disponível em: <https://www.udacity.com/course/viewer#!/c-ud853/l-1395568821/m-1601259313>. Acesso em: 11 mar. 2015.

WILSON, Jesse. **Android's HTTP Clients.** 2011. Disponível em: <http://android-developers.blogspot.com.br/2011/09/androids-http-clients.html>. Acesso em: 10 dez. 2014.

YAMASANI, Amith. **Creating an Input Method.** 2009. Disponível em: <http://android-developers.blogspot.com.br/2009/04/creating-input-method.html>. Acesso em: 03 jun. 2015.