UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

APLICATIVO ANDROID BASEADO EM REALIDADE AUMENTADA PARA RECOMENDAÇÕES DE LOCAIS

BRUNO KEWITZ DEMARCHI

BLUMENAU
2013

2013/1-09

BRUNO KEWITZ DEMARCHI

APLICATIVO ANDROID BASEADO EM REALIDADE AUMENTADA PARA RECOMENDAÇÕES DE LOCAIS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Marcel Hugo, Mestre - Orientador

BLUMENAU
2013

2013/1-09

APLICATIVO ANDROID BASEADO EM REALIDADE AUMENTADA PARA RECOMENDAÇÕES DE LOCAIS

Por

BRUNO KEWTIZ DEMARCHI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Marcel Hugo, Mestre – Orientador, FURB
Membro:	Prof. Dalton Solano dos Reis, M.Sc. - FURB
Membro:	Prof. Aurélio Faustino Hoppe, M.Sc. - FURB

Blumenau, 09 de julho de 2013

Dedico este trabalho a todas as pessoas que me ajudaram na realização deste, desde professores a amigos.

AGRADECIMENTOS

À minha família, por estarem sempre presentes.

Ao meu primo e amigo, Leonardo K. Kewitz, por todo o apoio.

À minha namorada, Roberta E. Geisler, pela confiança e ajuda.

Ao meu orientador, Marcel Hugo, por ter me guiado durante a realização deste trabalho.

Uma vida sem desafios não vale a pena ser vivida.

Sócrates

RESUMO

Este trabalho apresenta uma aplicação Android para recomendações de locais utilizando georreferenciamento e realidade aumentada. Na aplicação é possível cadastrar um perfil, manter uma lista de amigos, cadastrar locais, recomendar locais e procurar por locais recomendados por outros usuários. O resultado da busca de locais recomendados é visto através de realidade aumentada, cuja implementação utiliza recursos do dispositivo móvel tais como bússola e câmera. A renderização de gráficos na realidade aumentada é feita por meio da biblioteca OpenGL ES. A aplicação se comunica com um servidor que possui diversos serviços REST publicados, onde estão implementadas as regras de negócio. Por fim, a aplicação é apresentada sendo utilizada em um dispositivo móvel.

Palavras-chave: Realidade aumentada. Android. Rede social. Recomendação de locais.

ABSTRACT

This paper presents an Android application for place recommendations based on georeferencing and augmented reality. This application allows to create a profile, maintain a friend's list, register places, recommend places and search for places recommended by other users. The result of the search is seen through augmented reality, which implementation uses device's features such as compass and camera. The graphics rendering is made using the OpenGL ES library. The application communicates with a server that has several REST services published, where the business requirements are implemented. Finally, the application is shown running on a mobile device.

Key-words: Augmented reality. Android. Social network. Places recommendation.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADT – Android *Development Tools*

API – *Application Programming Language*

AVA – Ambiente Virtual de Aprendizagem

DAO – *Data Accesss Object*

FBC – Filtragem Baseada em Conteúdo

FC – Filtragem Colaborativa

FI – Filtragem de Informação

FURB – Universidade Regional de Blumenau

GB – *Giga Byte*

GHz – *Giga Hertz*

GPS – *Global Positioning System*

HTTP – *HyperText Transfer Protocol*

IP – Internet *Protocol*

J2EE – Java 2 *Platform Enterprise Edition*

JDBC – *Java DataBase Connectivity*

JNDI – *Java Naming and Directory Interface*

JNI – Java *Native Interface*

NDC – *Normalized Device Coordinates*

RA – Realidade Aumentada

REST – *REpresentational State Transfer*

RF – Requisito Funcional

RN – Regra de Negócio

RNF – Requisito Não-Funcional

SOA – *Service-oriented architecture*

UML – *Unified Modeling Language*

Wi-Fi – *Wireless Fidelity*

LISTA DE SÍMBOLOS

% – por cento

º – graus

SUMÁRIO

1 INTRODUÇÃO

Os dispositivos móveis estão cada vez mais potentes e com isso tem propiciado não só o desenvolvimento de novas tecnologias como a sua aplicação em áreas onde torna-se cada vez mais inadequado o uso de tecnologias móveis da geração anterior tais como *notebooks* e *netbooks*.

Na sua maioria, os dispositivos móveis já vêm preparados para acessar a internet, possuem câmera, *Global Positioning System* (GPS), acelerômetro e bússola. Este conjunto de recursos permite o desenvolvimento de soluções de Realidade Aumentada (RA), onde há uma mesclagem entre imagens da realidade e imagens produzidas por um programa (AZUMA, 1997, p. 257).

A mobilidade propiciada pelos dispositivos permite ao usuário maior controle do espaço que terá a sua Realidade Aumentada (VASSELAI, 2010, p. 18). Neste caso o conceito mais usado é o de *video see-through* (visão através de vídeo), no qual a câmera captura o vídeo em tempo real que é sobreposto por objetos gráficos antes de ser visualizado pelo usuário (BIMBER; RASKAR, 2005, p. 79).

Esta mobilidade que os dispositivos oferecem também é uma das razões pela popularidade dos sites de redes sociais. Pode-se estar conectado e interagindo com outras pessoas em praticamente qualquer lugar. Segundo ComScore (2013), os consumidores de internet brasileiros passam cerca de 36% do seu tempo *on-line* em sites de redes sociais.

Com base nas informações anteriores, este trabalho teve por objetivo o desenvolvimento de um aplicativo para Android que permita adicionar opiniões e recomendações sobre locais, utilizando o posicionamento GPS do dispositivo. Com este aplicativo, o usuário pode, utilizando a câmera do dispositivo, visualizar a distância dos locais recomendados pelos outros usuários, onde são destacados na tela somente os locais que estiverem na direção apontada pelo dispositivo. Além disso, é possível visualizar textualmente os comentários realizados sobre um local selecionado.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver um aplicativo Android para recomendações de locais que utilize georreferenciamento e realidade aumentada.

Os objetivos específicos do trabalho são:

a) permitir o cadastro de opiniões sobre locais, utilizando o posicionamento GPS do dispositivo;
b) permitir adicionar outros usuários a uma lista de amigos;
c) permitir visualizar a distância dos locais recomendados pelos amigos utilizando RA;
d) permitir filtrar as recomendações por nota, amigos específicos, distância máxima e categoria do local (pontos turísticos, bares, cinemas, etc.);
e) permitir visualizar textualmente as opiniões dos usuários ao selecionar um local.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está apresentada em quatro capítulos, sendo que o segundo contém a fundamentação teórica necessária para o entendimento deste trabalho.

O terceiro capítulo apresenta como foi desenvolvida a aplicação para a plataforma Android, as descrições dos casos de uso, diagramas de classe e a especificação que define a aplicação. No terceiro capítulo também são apresentadas as partes principais da implementação juntamente com resultados e discussões que aconteceram durante o desenvolvimento do projeto.

Por fim, o quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

As próximas seções contêm a fundamentação teórica necessária para o entendimento deste trabalho. Primeiramente é descrito sobre sistemas de recomendação. Em seguida, é explanado sobre as redes sociais e sua importância nos dias atuais. Posteriormente, trata-se da plataforma móvel Android e o uso da RA para imersão do usuário nos softwares. Por último, são apresntados os trabalhos correlatos à ferramenta proposta.

2.1 SISTEMAS DE RECOMENDAÇÃO

Nos dias atuais as pessoas estão expostas a uma enorme quantidade de informações, o que proporciona um grande número de opções para diversas situações da vida e do cotidiano. Muitas vezes um indivíduo possui pouca ou quase nenhuma experiência pessoal para realizar escolhas entre as várias alternativas que lhe são apresentadas (REATEGUI; CAZELLA, 2005).

Diante de tantas possibilidades buscam-se meios de reduzir esta quantidade de opções em busca da melhor alternativa. Uma forma de fazê-la é confiar nas recomendações passadas por outras pessoas. Segundo Reategui e Cazella (2005), os sistemas de recomendação auxiliam no aumento da capacidade e eficácia deste processo de indicação.

Existem diversas técnicas que fundamentam os sistemas de recomendação. Dentre elas pode-se citar a Filtragem de Informação (FI), que é uma variedade de processos que envolvem a entrega de informação para as pessoas que realmente necessitam delas. Para isto geralmente mantém um perfil dos interesses do usuário, ou seja, baseia-se nas preferências dos mesmos (REATEGUI; CAZELLA, 2005).

A FI pode ainda ser dividida em Filtragem Baseada em Conteúdo (FBC) e Filtragem Colaborativa (FC). Na FBC, os interesses do usuário são obtidos através de informações fornecidas por ele próprio ou através de ações, como seleção e aquisição de itens (REATEGUI; CAZELLA, 2005). Ainda segundo Reategui e Cazella (2005), a FC se diferencia da FBC por não exigir a compreensão ou reconhecimento do conteúdo dos itens, visto que a filtragem é realizada com base nas avaliações feitas pelos próprios usuários.

2.2 REDES SOCIAIS

Redes sociais são, antes de tudo, relações entre pessoas (AGUIAR, 2007, p. 2). Com o surgimento da internet estas relações puderam ser passadas para o universo tecnológico, onde as pessoas comunicam-se e interagem através de sistemas em lugar da presença física.

Segundo Boyd e Ellison (2008, p. 2), um *site* de rede social é um serviço que permite um indivíduo: (1) a criar um perfil público ou semi-público; (2) atrelar uma lista de outros usuários com os quais se compartilha uma conexão; e (3) visualizar e utilizar a sua lista de conexões e as listas feitas por outros usuários dentro do sistema. Na internet estão disponíveis diversos sistemas de rede social, onde é possível diferenciá-los pelos seus objetivos e tipos de interação. Existem sistemas voltados ao mundo profissional, como o LinkedIn (2013), onde pode-se cadastrar seus conhecimentos e habilidades, procurar vagas de emprego, etc. Também existem redes sociais voltadas à área acadêmica, como o Ambiente Virtual de Aprendizagem (AVA) da Universidade Regional de Blumenau (FURB), onde pode-se, entre outras funcionalidades, postar materiais acadêmicos e criar fóruns de discussões (FURB, 2013). Além destes, pode-se citar os sistemas voltados à política, como o PoliticaBook (2013), cujo objetivo é instigar o debate político por meio de postagens de notícias da área. Por fim, pode-se citar os sistemas voltados ao relacionamento social, como o Facebook (2013), onde é possível compartilhar e curtir os mais diversos tipos de mídias postadas por outros usuários, como fotos, vídeos, textos e *links* externos. Neste último tipo de rede social é comum que entre os indivíduos exista a interação através de recomendações. Estas recomendações podem variar entre os mais diversos assuntos, desde jogos, tecnologia e música até restaurantes, casas noturnas e viagens.

As redes sociais tornaram-se parte do cotidiano das pessoas. De acordo com uma pesquisa da ComScore (2013), os consumidores no Brasil passam a maior parte do tempo *on-line* - 36% - em sites de redes sociais. Uma das razões pela utilização das redes sociais no cotidiano das pessoas é o fato delas estarem o tempo inteiro conectadas ao mundo virtual e muito disso se deve a recente popularização dos *smartphones,* os quais comumente vêm equipados com recursos que possibilitam a conexão à internet através de tecnologias como 3G ou *Wireless Fidelity* (Wi-Fi).

2.3 ANDROID

Android é uma plataforma completa para dispositivos móveis que disponibiliza um sistema operacional, uma camada intermediária e diversas aplicações chave para o desenvolvimento de software (OPEN HANDSET ALLIANCE, 2013). Segundo a Open Handset Alliance (2013), Android possui código aberto e utiliza uma máquina virtual customizada, que foi desenhada para otimizar o uso de memória e outros recursos de *hardware* para o ambiente móvel.

A seguir são detalhadas algumas informações sobre o desenvolvimento para Android. Primeiramente é explanada a arquitetura da plataforma. Depois, é apresentado o arquivo `AndroidMainfest.xml`. Em seguida, são explicadas as classes `Activity` e `Intent` e como elas trabalham juntas. Posteriormente, é apresentada a classe `AsyncTask` para execução de processos concorrentes. Além disso, é falado sobre os serviços de localização disponíveis. No tópico seguinte, é tratada a utilização da *Application Programming Language* (API) dos sensores do dispositivo. Posteriormente é explanado sobre a utilização de câmeras. Por último, é apresentada a biblioteca OpenGL ES para renderização de gráficos.

2.3.1 Arquitetura

A arquitetura do Android é dividida basicamente em quatro camadas: o *kernel* Linux, as bibliotecas nativas, o *framework* para desenvolvimento e as aplicações da plataforma (ELINUX, 2011). A primeira camada tem a função de abstrair os recursos de hardware para as demais camadas, executando as tarefas de gerenciamento de memória, gerenciamento de processos e gerenciamento de rede (VASSELAI, 2010, p. 20). Segundo a Elinux (2011), a segunda camada foi escrita em C e C++ e disponibiliza diversas implementações de baixo nível. Além disso, na segunda camada está a máquina virtual Dalvik, desenvolvida para customizar a execução de aplicações Java em dispositivos com recursos de *hardware* limitados (DALVIKVM, 2008). A terceira camada é o *framework* para desenvolvimento. Segundo Vasselai (2010, p. 20), esta camada foi desenvolvida em Java e invoca as bibliotecas da segunda camada utilizando Java *Native Interface* (JNI). Já a última camada possui as aplicações padrão da plataforma, que podem ser reutilizados para suprir ações que necessitem de calendário, contatos, entre outros.

2.3.2 O arquivo `AndroidMainfest.xml`

Segundo Android (2013h), toda aplicação deve possuir obrigatoriamente o arquivo `AndroidManifest.xml` no seu diretório raíz. Esse *manifest* contém informações essenciais sobre a aplicação e sobre o que o dispositivo deve possuir para executá-la.

Entre as informações contidas nesse arquivo, pode-se citar a versão mínima do sistema operacional, o identificador único da aplicação, a descrição de cada componente da aplicação, as permissões que a aplicação deve possuir para acessar partes de outras APIs e interagir com outras aplicações; além das permissões que outras aplicações devem possuir para acessar os componentes criados por ela (ANDROID, 2013h).

2.3.3 `Activity` e `Intent`

`Activity` é um componente Android que fornece uma tela com a qual o usuário pode interagir. Cada `Activity` possui uma janela que normalmente preenche a tela inteira do dispositivo, onde pode-se montar os formulários conforme necessário (ANDROID, 2013a).

Segundo Android (2013a), uma aplicação normalmente possui diversos componentes `Activity` que devem ser fracamente acoplados, cada um com seu objetivo específico - sendo um definido como o *main*, ou seja, é a `Activity` apresentada ao usuário quando a aplicação é iniciada pela primeira vez.

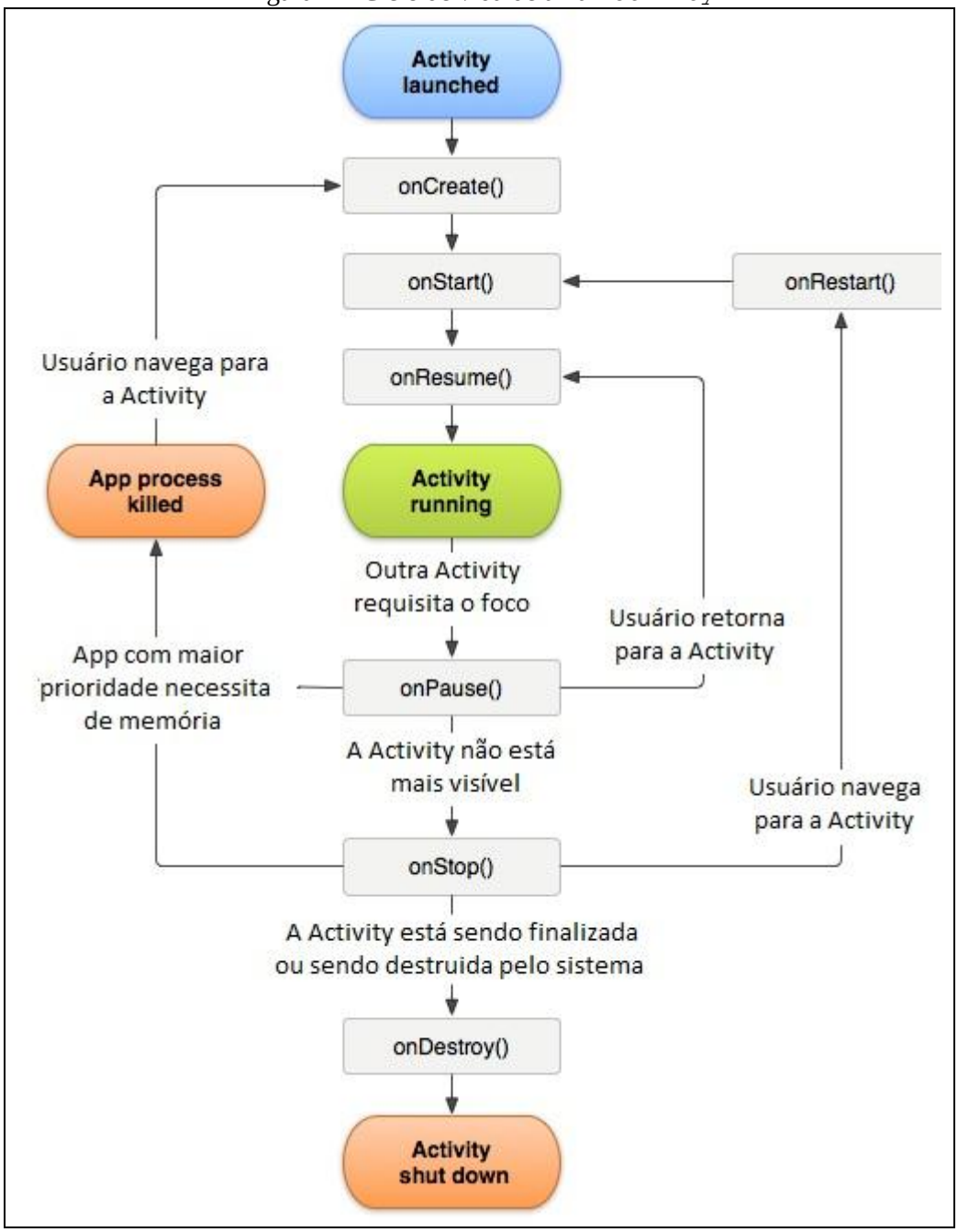
Cada `Activity` pode criar e iniciar outra `Activity` para executar diferentes ações. Segundo Android (2013a), cada vez que uma nova `Activity` é iniciada, a anterior é parada e a nova pede o foco para ser mostrada na tela do dispositivo. Além disso, a nova `Activity` é colocada no topo de uma pilha. Toda vez que uma `Activity` é terminada ou o botão voltar do dispositivo é pressionado, a `Activity` no topo da pilha é destruída e removida da mesma, dando lugar à `Activity` anterior (ANDROID, 2013a).

Toda `Activity` possui um ciclo de vida onde seus eventos são executados e podem ser sobrescritos pelo desenvolvedor. O ciclo de vida de uma `Activity` é composto pelos seguintes eventos (ANDROID, 2013a):

a) `onCreate()`: chamado quando a `Activity` é criada;
b) `onStart()`: chamado momento antes da `Activity` estar visível;
c) `onResume()`: chamado quando a `Activity` se tornou visível;
d) `onPause()`: chamado quando outra `Activity` requisitou o foco do dispositivo;
e) `onStop()`: chamado quando a `Activity` não está mais vísivel, ou seja, está parada;
f) `onDestroy()`: chamado momento antes da `Activity` ser destruída e removida da pilha de atividades.

O fluxograma do ciclo de vida de uma `Activity` pode ser visualizado na Figura 1.

Figura 1 – Ciclo de vida de uma `Activity`



Fonte: adaptado de Android (2013a).

Para iniciar outra `Activity` deve-se utilizar o método `startActivity()` que recebe uma instância da classe `Intent`, a qual contém a especificação de qual `Activity` deve ser criada (ANDROID, 2013a).

Segundo Android (2013a), quando deseja-se abrir uma `Activity` da sua própria aplicação basta criar uma `Intent` que contém explicitamente a `Activity` que deve ser criada através do nome da sua classe. Porém, existem casos onde você precisa executar ações que não estão implementadas na sua aplicação, como por exemplo mandar um *e-mail* ou bater uma foto: nesses casos pode-se criar uma `Intent` que descreve qual tipo de ação deve ser

executada e então o próprio sistema operacional se encarrega de escolher a aplicação mais adequada (ANDROID, 2013a).

Segundo Android (2013a), é possível passar dados como parâmetros para outras `Activities` através do método `putExtra()` da classe `Intent` ou até mesmo receber o resultado de outra `Activity`. Para tal deve-se utilizar o método `startActivityForResult()` e implementar o método `onActivityResult()` que será chamado assim que a `Activity` requisitada for finalizada.

2.3.4 `AsyncTask`

O Android possui no pacote `android.os` a classe `AsyncTask` que permite a execução de processos concorrentes de forma simples, sem tratamento de `Thread` e sincronismo de objetos, onde o resultado do processo é publicado no processo principal - também conhecido como processo da interface do usuário (ANDROID, 2013b). Segundo Android (2013b), a `AsyncTask` deve ser utilizada para operações curtas (com no máximo alguns segundos).

A classe `AsyncTask` é abstrata, ou seja, não se pode obter uma instância dela diretamente: para tanto deve-se criar uma outra classe que estenda `AsyncTask` e que implemente o método abstrato `doInBackground(Params...)`. Segundo Android (2013b), um componente `AsyncTask` é definido por três tipos genéricos: `Params`, `Progress` e `Result`. O primeiro define o tipo de parâmetro que será enviado ao processo concorrente. O segundo define o tipo de unidade de progresso que é publicado durante a execução do processo. Já o terceiro define o tipo do resultado que será retornado ao processo principal (ANDROID, 2013b).

Segundo Android (2013b), a execução de um `AsyncTask` é composta por quatro passos que podem ser sobrescritos:

a) `onPreExecute()`: executado no processo principal antes da tarefa concorrente ser executada. Este passo normalmente é utilizado para preparar a execução do processo - por exemplo montar uma barra de progresso na tela da aplicação;
b) `doInBackground(Params...)`: executado imediatamente após o final do passo `onPreExecute()`. O código deste passo é executado de forma concorrente. Os parâmetros para a execução do processo são passados para este passo. Além disso, o resultado do processo deve ser retornado neste passo. Uma opção é realizar a chamada do método `publishProgress(Progress...)` o qual será publicado no processo principal, mais especificamente no próximo passo;

c) `onProgressUpdate(Progress...)`: executado no processo principal depois de uma chamada para o `publishProgress(Progress...)` no passo anterior. Este passo normalmente é utilizado para atualizar uma barra de progresso;
d) `onPostExecute(Result)`: executado no processo principal logo após o processo concorrente terminar. O resultado do processo concorrente é passado como parâmetro para este evento.

2.3.5 Serviços de localização

O Android possui o pacote `android.location` o qual fornece classes para acesso aos serviços de localização geográfica disponíveis no dispositivo (ANDROID, 2013d). Segundo Android (2013d), o componente central para o *framework* de localização é a classe `LocationManager` a qual fornece diversas APIs para obter-se a posição geográfica do usuário.

Para obter-se uma instância da classe `LocationManager` é necessário realizar a chamada `getSystemService(Context.LOCATION_SERVICE)` (ANDROID, 2013d). Segundo Android (2013d), através dessa instância é possível obter-se a lista dos fornecedores de localização disponíveis no dispositivo, representados pela classe `LocationProvider`. Além disso, é possível inscrever-se em determinado fornecedor para receber atualizações periódicas da mudança de localização do dispositivo ou quando a localização do dispositivo está próxima de uma posição geográfica específica (ANDROID, 2013d).

Segundo Android (2013e), para obter-se a localização do usuário têm-se duas opções de fornecedores: GPS e rede. Embora o GPS seja mais preciso do que a rede, ele só funciona em ambientes a céu aberto, além de consumir rapidamente a bateria do dispositivo e ter um tempo de resposta mais longo (ANDROID, 2013e). Já o fornecedor baseado na rede determina a localização do dispostivo baseado na torre de celular e sinais Wi-Fi, funcionando tanto em ambientes fechados quanto a céu aberto, além de obter a resposta mais rapidamente e com menos consumo de bateria quando comparado ao GPS (ANDROID, 2013e).

Segundo Android (2013e), para registrar-se em algum fornecedor com o objetivo de obter a localização do dispositivo deve-se chamar o método `requestLocationUpdates(String,long,float,LocationListener)` da classe `LocationManager`, onde o primeiro parâmetro é o tipo de fornecedor a ser utilizado: pode-se utilizar a contante `LocationManager.GPS_PROVIDER` para o GPS e a constante `LocationManager.NETWORK_PROVIDER` para a rede. O segundo e terceiro parâmetros definem a periodicidade com que a localização deve ser atualizada, sendo o segundo parâmetro o intervalo de tempo mínimo em milissegundos e o terceiro parâmetro a distância mínima em

metros (ANDROID, 2013e). Já o último parâmetro é um objeto cuja classe deve implementar a interface `LocationListener`, sendo que o método `onLocationChanged(Location)` será executado toda vez que o fornecedor atualizar a localização do dispositivo (ANDROID, 2013e).

2.3.6 Sensores

O Android disponibiliza como parte do pacote `android.hardware` um *framework* para obter-se valores de sensores dos dispositivos, tais como bússola, acelerômetro, sensor de luz, medidor de temperatura, entre outros (ANDROID, 2013g). Segundo Android (2013g), o componente central para este *framework* é a classe `SensorManager`, cuja instância pode ser obtida através da chamada `getSystemService(Context.SENSOR_SERVICE)`.

Para obter-se a lista de todos os sensores disponíveis no dispositivo, deve-se usar o método `getSensorList(Sensor.TYPE_ALL)` (ANDROID, 2013g). Ainda segundo Android (2013g), para obter-se o principal sensor de determinado tipo deve-se utilizar o método `SensorManager.getDefaultSensor(int type)`. Para receber as atualizações de um sensor têm-se o método `registerListener(SensorEventListener, Sensor, int)` da classe `SensorManager`, sendo o segundo parâmetro o sensor obtido anteriormente e o terceiro parâmetro um valor que define a frequência da atualização. Já o primeiro parâmetro deve ser um objeto cuja classe implementa a interface `SensorEventListener`, sendo que o método `onSensorChanged(SensorEvent)` será executado toda vez que o sensor atualizar os seus valores. Estes valores podem ser obtidos através do parâmetro `SensorEvent` no atributo `SensorEvent.values`, assim como a precisão no atributo `SensorEvent.accuracy`, o próprio sensor no atributo `SensorEvent.sensor` e o tempo das medições no atributo `SensorEvent.timestamp` (ANDROID, 2013g).

2.3.7 Câmera

O *framework* Android inclui suporte para desenvolvimento com diversos recursos das câmeras do dispostivo, permitindo a captura de imagens e vídeos através da API da câmera ou até mesmo através do uso do `Intent` (ANDROID, 2013c).

Segundo Android (2013c), as principais classes para o uso da câmera são:

a) `Camera`: a principal para controlar as câmeras do dispositivo. Utilizada para capturar imagens e vídeos em aplicativos que necessitam de câmera;
b) `SurfaceView`: utilizada para mostrar o *preview* da câmera ao usuário, ou seja, as imagens da câmera em tempo real;

c) `MediaRecorder`: utilizada para gravar o vídeo da câmera;

d) `Intent`: uma `Intent` criada com os tipos `MediaStore.ACTION_IMAGE_CAPTURE` ou `MediaStore.ACTION_VIDEO_CAPTURE` pode ser utilizada para capturar imagens e vídeos utilizando outras aplicações - desta forma não se faz necessário utilizar o objeto `Camera` diretamente.

Conforme Android (2013c), para utilizar a câmera em uma aplicação Android é necessário que a mesma possua algumas permissões declaradas no *manifest*. As duas declarações fundamentais são: (1) do tipo `permission`, `android.permission.CAMERA;` e (2) do tipo `feature`, `android.hardware.camera`. A primeira requere permissão para o uso da câmera, enquanto que a segunda define o *hardware* da câmera como necessário para a execução da aplicação, podendo ser alterada para opcional através do valor `false` no atributo `android:required` (ANDROID, 2013c). Segundo Android (2013c), as permissões opcionais são para gravação em disco da mídia obtida (`android.permission.WRITE_EXTERNAL_STORAGE`), gravação de áudio junto à gravação de vídeo (`android.permission.RECORD_AUDIO`) e anexo da posição GPS junto às imagens capturadas (`android.permission.ACCESS_FINE_LOCATION`).

2.3.8 OpenGL ES

O OpenGL é uma API gráfica multi-plataforma que disponibiliza uma interface de *software* para processamento gráfico 2D e 3D, sendo que o OpenGL ES é uma sub-seção do OpenGL destinado a dispositivos embarcados (ANDROID, 2013f).

Segundo Khronos (2013), o OpenGL ES é a única API verdadeiramente aberta e independente de fornecedor, sendo que qualquer pessoa pode baixar a especificação OpenGL ES e fazer a sua própria implementação. Com o uso de uma API como o OpenGL ES, é possível focar na implementação da aplicação propriamente dita ao invés de se preocupar com os recursos gráficos, visto que a API possui um alto nível de abstração.

Os recursos dos dispositivos móveis variam muito, e para isso o OpenGL ES foi desenhado para suportar estas diferenças, sendo necessário o mínimo de espaço para armazenamento de dados e o mínimo de instruções e tráfego de dados (KHRONOS, 2013).

A principal diferença entre o OpenGL e o OpenGL ES é que a versão embarcada não possui diversas funcionalidades presentes no OpenGL, como por exemplo as chamadas `glBegin` e `glEnd` (EDGELIB, 2013).

Segundo Android (2013f), duas classes são fundamentais para o desenvolvimento com a API do OpenGL ES para Android. A primeira classe é a `GLSurfaceView`, que define uma

`View` que receberá o renderizador do OpenGL ES, sendo que se deve estender essa classe caso seja necessário capturar eventos de toque na tela (ANDROID, 2013f). Já a segunda classe é a interface `GLSurfaceView.Renderer`, a qual define os métodos básicos que se deve implementar para renderizar gráficos na `GLSurfaceView`. O objeto `GLSurfaceView` recebe a instância do renderizador através do método `setRenderer()` (ANDROID, 2013f). Segundo Android (2013f), os métodos básicos que a interface `GLSurfaceView.Renderer` define são:

a) `onSurfaceCreated()`: chamado apenas uma vez, quando a `GLSurfaceView` é criada. É utilizado normalmente para configurar os parâmetros do ambiente e inicializar os objetos que serão renderizados;
b) `onSurfaceChanged()`: chamado toda vez que a geometria da `GLSurfaceView` é alterada, o que inclui mudanças no seu tamanho ou na mudança de orientação da tela do dispositivo;
c) `onDrawFrame()`: chamado toda vez que a `GLSurfaceView` é renderizada. É aqui onde os gráficos devem ser literalmente desenhados ou re-desenhados.

Segundo Android (2013f), existem duas versões do OpenGL ES que são a `1.0` e a `2.0`. Ambas oferecem alta performance em renderização de gráficos, mas deve-se considerar alguns fatores ao escolher qual delas utilizar (ANDROID, 2013f). Na sua maioria, a versão `2.0` possui desempenho melhor na renderização de gráficos, porém isso pode variar dependendo do dispositivo em que a aplicação executa (ANDROID, 2013f). Além disso, a versão `2.0` só está disponível a partir da versão `2.2` do Android (ANDROID, 2013f). Segundo Android (2013f), desenvolvedores sem experiência em OpenGL podem achar o desenvolvimento com a versão `1.0` mais rápida e conveniente. Por último, a versão `2.0` oferece maior controle dos gráficos renderizados a partir do uso de *shaders*, tornando possível a criação de efeitos que dificilmente seriam alcançados utilizando a versão `1.0` (ANDROID, 2013f).

2.4 REALIDADE AUMENTADA

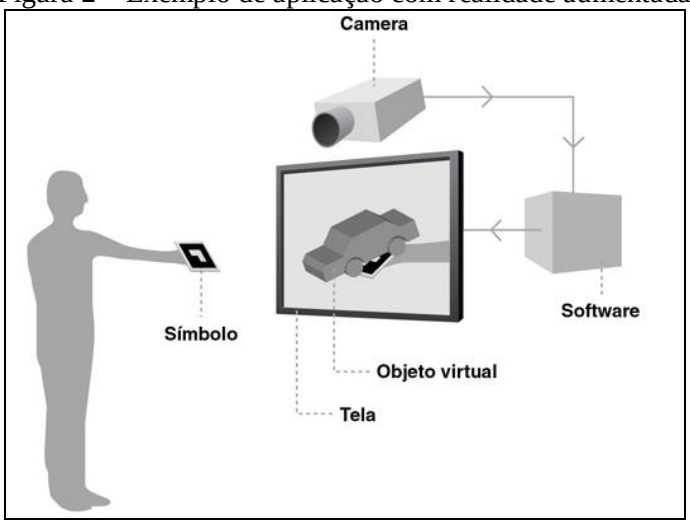
Segundo Bimber e Raskar (2005, p. 20), a RA procura inserir elementos virtuais ao mundo real, ou em um vídeo em tempo real deste mundo. Em outras palavras, aumentar a realidade é somar ao mundo real alguma informação virtual de maneira a complementá-lo, permitindo alguma interação entre o usuário e o ambiente gerado (VASSELAI, 2010, p. 18).

Um sistema de RA deve possuir três características básicas: combinar objetos reais e objetos virtuais em um ambiente real, executar interativamente em tempo real e alinhar

objetos reais e virtuais entre si (AZUMA, 1997, p. 257). Além disso, Azuma (1997, p. 357) também afirma que RA permite uma interação homem-máquina mais fácil e natural.

A Figura 2 exemplifica uma aplicação de realidade aumentada utilizando marcadores.

Figura 2 – Exemplo de aplicação com realidade aumentada



Fonte: Agência DDA (2013).

A aplicação exemplificada na Figura 2 funciona da seguinte forma: o usuário está posicionado em frente a uma câmera com um símbolo, também conhecido como marcador. O software processará as imagens obtidas pela câmera e ao identificar o marcador irá desenhar um carro virtual utilizando como base o posicionamento do símbolo, gerando a saída na tela. Pelo fato de ser inserido um objeto virtual a uma imagem do mundo real, esta aplicação é considerada como realidade aumentada.

Outro exemplo de RA, porém sem marcadores, é da marca de óculos Rayban a qual publicou em seu site uma aplicação chamada Rayban Virtual Mirror. Esta aplicação captura a imagem do rosto do usuário por meio da *webcam* e, através de pontos chave da face do usuário, insere virtualmente o óculos escolhido - fazendo com que o cliente possa provar os óculos sem tê-los em mãos (DIGITAL BUZZ, 2010).

2.5 TRABALHOS CORRELATOS

A seguir são apresentados os trabalhos correlatos ao aplicativo desenvolvido. Primeiramente é apresentada a rede social Foursquare (FOURSQUARE, 2013). Em seguida são apresentadas algumas informações sobre o aplicativo ainda em desenvolvimento What is Up App (WHAT IS UP APP, 2013). Além disso, é apresentado o aplicativo TripAdvisor Augmented Reality para recomendações de locais (BLACKBERRY OS, 2013). Por fim, é

apresentada uma ferramenta Android baseada em realidade aumentada e serviços baseados em localização usando notificações (RAMPELOTTI, 2011).

2.5.1 Foursquare

O Foursquare é um aplicativo gratuito para dispositivos móveis, lançado pela empresa Foursquare em 2009 (FOURSQUARE, 2013). Seu objetivo é ajudar os usuários a descobrir novos lugares, e para isso são utilizadas informações como a posição geográfica atual, locais já visitados por amigos, locais populares, tipos de localidade e recomendações de outras pessoas com o perfil parecido com o do usuário.

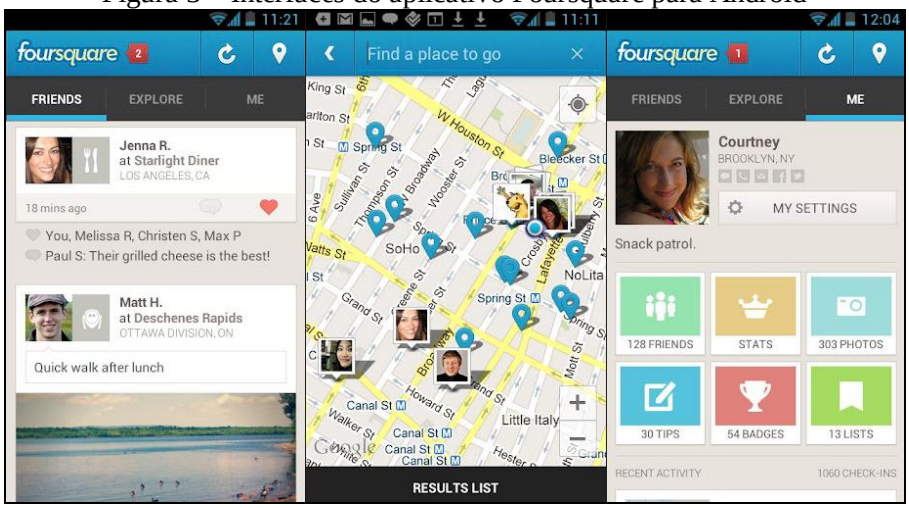
O servidor possui um amplo cadastro de locais de diversos tipos que podem ser cadastrados pelos próprios usuários, tais como restaurantes, shoppings, residências, empresas, casas noturnas, etc. É possível cadastrar suas opiniões através de mensagens que serão anexadas ao local em questão. Além disso, é possível realizar o *check-in* em um local, indicando que você está naquele lugar no momento.

Segundo Foursquare (2013), o sistema possui mais de 20 milhões de usuários em todo o mundo, com mais de 2,5 bilhões de *check-ins* ao total, número que cresce cerca de um milhão por dia. O software possui aplicações disponíveis para diversos sistemas operacionais, como Android, iOS, BlackBerry, WebOS, Symbian e Windows Phone, além de poder ser acessado via web através das versões *desktop* e *mobile* (FOURSQUARE, 2013).

Os aplicativos não possuem mecanismos de realidade aumentada, sendo que a interação com o usuário é realizada utilizando mapas e formulários.

A Figura 3 apresenta algumas interfaces da aplicação Foursquare para Android.

Figura 3 – Interfaces do aplicativo Foursquare para Android



Fonte: adaptado de Google (2013).

2.5.2 What is Up App

O What is Up App é uma aplicação ainda em desenvolvimento cujo objetivo é somar Realidade Aumentada com as informações de locais cadastradas no Foursquare (WHAT IS UP APP, 2013). O aplicativo está sendo desenvolvido para rodar apenas no sistema operacional iOS.

A Figura 4 apresenta a aplicação sendo executada, com os pontos de interesse do Foursquare destacados sobre as imagens captadas pela câmera do aparelho.

Figura 4 – Execução do aplicativo What is Up App

Fonte: What is Up App (2013).

2.5.3 TripAdvisor Augmented Reality

O TripAdvisor Augmented Reality é um aplicativo gratuito para o sistema operacional BlackBerry e foi lançado em agosto de 2012 pela Wikitude (BLACKBERRY OS, 2013). O aplicativo utiliza as informações de locais cadastrados no *site* de viagens TripAdvisor e as apresenta utilizando realidade aumentada, onde os locais aparecem sobre as imagens da câmera.

Ao selecionar um local é possível acessar informações detalhadas sobre o mesmo, tais como opiniões enviadas por outros usuários, preços, etc. Também há opção *Bring me there*, que indica as direções para o usuário chegar até o local selecionado.

A Figura 5 apresenta a aplicação TripAdvisor Augmented Reality sendo executada.

Figura 5 – Execução do aplicativo TripAdvisor Augmented Reality

Fonte: BlackBerry OS (2013).

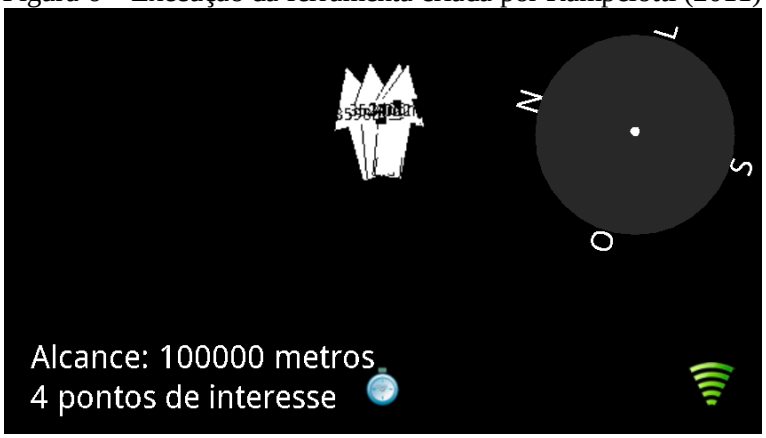
2.5.4 Ferramenta Android baseada em realidade aumentada e serviços baseados em localização usando notificações

A ferramenta proposta por Rampelotti (2011) tem como objetivo disponibilizar uma aplicação para auxiliar o cotidiano do aluno no ambiente universitário da FURB através de serviços baseados em localização utilizando o conceito de *push* para dispositivos móveis da plataforma Android.

Nesta ferramenta é possível visualizar e comunicar-se com os membros de uma disciplina, visualizar o mapa da universidade com os eventos criados, além de ter a opção de visualizar a direção dos eventos com RA. Ainda, a ferramenta utiliza um mecanismo de notificações *push* para alertar o aluno quando ele está próximo do local de um evento ou caso o professor tenha cadastrado algum novo evento, nota ou freqüência (RAMPELOTTI, 2011).

A Figura 6 apresenta a tela que utiliza RA para indicar a direção dos eventos na ferramenta criada por Rampelotti (2011).

Figura 6 – Execução da ferramenta criada por Rampelotti (2011)



Fonte: Rampelotti (2011).

2.5.5 Comparação entre os trabalhos correlatos

A partir dos dados levantados anteriormente é possível comparar algumas características e funcionalidades dos trabalhos apresentados. Esta comparação pode ser visualizada no Quadro 1.

Quadro 1 – Comparação entre trabalhos correlatos

Características / Funcionalidades	Foursquare	What is Up App	TripAdvisor Augmented Reality	TCC Rampelotti
desenvolvimento finalizado	sim	não	sim	sim
recomendação de locais	sim	sim	sim	não
disponível para Android	sim	não	não	sim
realidade aumentada	não	sim	sim	sim

Nota-se que a única aplicação ainda em desenvolvimento é o What is Up App. Já o trabalho de Rampelotti (2011) é a única aplicação cujo objetivo não é recomendar locais. Percebe-se também que as aplicações Foursquare e o trabalho de Rampelotti (2011) são os únicos trabalhos disponíveis para Android. Por fim, constata-se que o Foursquare é o único aplicativo que não possui a funcionalidade de RA.

3 DESENVOLVIMENTO

Este capítulo demonstra o desenvolvimento da aplicação desenvolvida. São apresentados os requisitos, especificação e implementação da aplicação.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A aplicação Android para recomendações de locais baseada em RA e seu servidor foram construídas com base nos seguintes Requisitos Funcionais (RF) e Requisitos Não-Funcionais (RNF):

a) o sistema deve permitir o cadastro de usuários e seus respectivos perfis (RF01);
b) o sistema deve permitir adicionar outros usuários a uma lista de amigos (RF02);
c) o sistema deve permitir o cadastro de locais utilizando o posicionamento GPS do dispositivo (RF03);
d) o sistema deve permitir o cadastro de opiniões sobre locais utilizando o posicionamento GPS do dispositivo (RF04);
e) o sistema deve permitir visualizar a direção e a distância dos locais recomendados utilizando RA (RF05);
f) o sistema deve permitir filtrar as recomendações visíveis (RF06);
g) o sistema deve permitir visualizar textualmente as opiniões dos usuários e mais informações de um local ao selecioná-lo (RF07);
h) o sistema deve permitir visualizar a rota a um determinado local (RF08);
i) o sistema deve permitir alterar as configurações de privacidade do perfil, tornando mais dados visíveis a usuários desconhecidos (RF09);
j) o sistema deve utilizar senhas de acesso para garantir a segurança da aplicação (RNF01);
k) o sistema deve ser implementado em Java (RNF02);
l) o sistema deve ser compatível com o sistema operacional Android 4.0.4 ou posterior (RNF03);
m) o sistema deve armazenar os dados no servidor utilizando o banco de dados PostgreSQL 9 ou posterior (RNF04);
n) o sistema deve utilizar a biblioteca OpenGL ES para implementação da RA (RNF05);
o) o sistema deve disponibilizar uma interface *Web Service* para acesso aos seus dados e regras (RNF06);

p) os *web services* devem estar disponíveis na internet através do servidor de aplicação JBoss 7.1 (RNF07);
q) a aplicação deve utilizar a API do Google Maps para disponibilizar as rotas aos locais (RNF08).

3.2 ESPECIFICAÇÃO

A especificação foi realizada a partir da ferramenta Enterprise Architect utilizando a linguagem de modelagem *Unified Modeling Language* (UML). Para a modelagem foram utilizados os diagramas de casos de uso, classe e arquitetura.

3.2.1 Regras de Negócio

A aplicação e o servidor foram desenvolvidos respeitando às seguintes Regras de Negócio (RN):

a) para se cadastrar, o usuário deverá fornecer obrigatoriamente seu nome, data de nascimento, senha e e-mail (RN01);
b) o e-mail deve ser único para cada usuário (RN02);
c) a idade mínima do usuário deve ser 16 anos (RN03);
d) após se cadastrar, o usuário deve confirmar o cadastro através do link enviado para o e-mail informado no cadastro (RN04);
e) ao adicionar um amigo, a amizade só será confirmada após o usuário adicionado aceitá-la (RN05);
f) por padrão, as únicas informações visíveis de perfis de usuários que não fazem parte da lista de amigos são o nome e foto do perfil (RN06);
g) o usuário pode alterar as configurações de privacidade em duas opções:
   - deixando todos os dados visíveis;
   - deixando somente o nome e a foto vísiveis a usuários desconhecidos (RN07);
h) ao cadastrar um local, o ponto geográfico do mesmo deve estar no máximo a 1.000 metros do posicionamento GPS do usuário (RN08);
i) as informações obrigatórias para o cadastro de um local são: endereço, nome, categoria e foto. Além disso, deve ser feita obrigatoriamente uma recomendação do local (RN09);
j) ao recomendar um local, o ponto geográfico do mesmo deve estar no máximo a 1.000 metros do posicionamento GPS do usuário (RN10);

k) as informações obrigatórias para recomendar um local são: nota da recomendação e opinião (RN11);
l) a nota da recomendação deve ser um número inteiro entre 0 e 10 (RN12);
m) as recomendações visíveis na tela da RA devem estar no máximo a 45º a esquerda ou 45º a direita em relação à direção da bússola do dispositivo (RN13);
n) as opções de filtro para as recomendações visíveis são: distância máxima, nota mínima, amigos específicos ou categoria do local (RN14);
o) para o filtro do local, a distância máxima é uma informação obrigatória assim como a nota mínima e deve possibilitar um valor entre 500 e 10.000 metros. Além disso, o usuário deve informar ao menos um amigo específico ou ao menos uma categoria (RN15);
p) ao selecionar um local, a recomendação do criador deve ser a primeira a ser mostrada, seguida pelas recomendações dos amigos e depois as demais recomendações (RN16);
q) os tipos de rotas possíveis são: a pé ou de carro (RN17).

3.2.2 Arquitetura do sistema

A arquitetura definida para o sistema resultou na definição de três projetos: `TCC WAR`, `TCC Aplicação Android` e `TCC Common`.

O projeto `TCC WAR` contém o pacote web que rodará no servidor de aplicação JBoss. O foco desse projeto é disponibilizar serviços que encapsulam as regras de negócio do sistema em *REpresentational State Transfer* (REST), tornando tais serviços consumíveis em qualquer plataforma: os mesmos serviços utilizados por uma aplicação Android poderiam ser consumidos por uma aplicação para iOS ou Windows Phone, por exemplo. Esses serviços REST foram configurados para consumir e retornar dados no compacto formato JavaScript *Object Notation* (JSON), gerando menos tráfego de dados: o que é importante em aplicações para dispositivos móveis os quais possuem em sua maioria conexões com a internet de baixa velocidade ou limites de tráfego de dados periódicos. Para a implementação destes serviços foi utilizada a API RestEasy que por meio de anotações em classes e métodos Java disponibiliza serviços em REST, gerando alta produtividade onde o desenvolvedor pode se concentrar mais na implementação do negócio do que na manutenção da tecnologia. Além disso, esse projeto também é o responsável pela comunicação com os serviços de localização do Google Maps, utilizados para consulta de endereços através de posições geográficas e vice-versa. Ainda, este projeto possui a incumbência de se comunicar com o banco de dados PostgreSQL onde são salvos os dados do sistema.

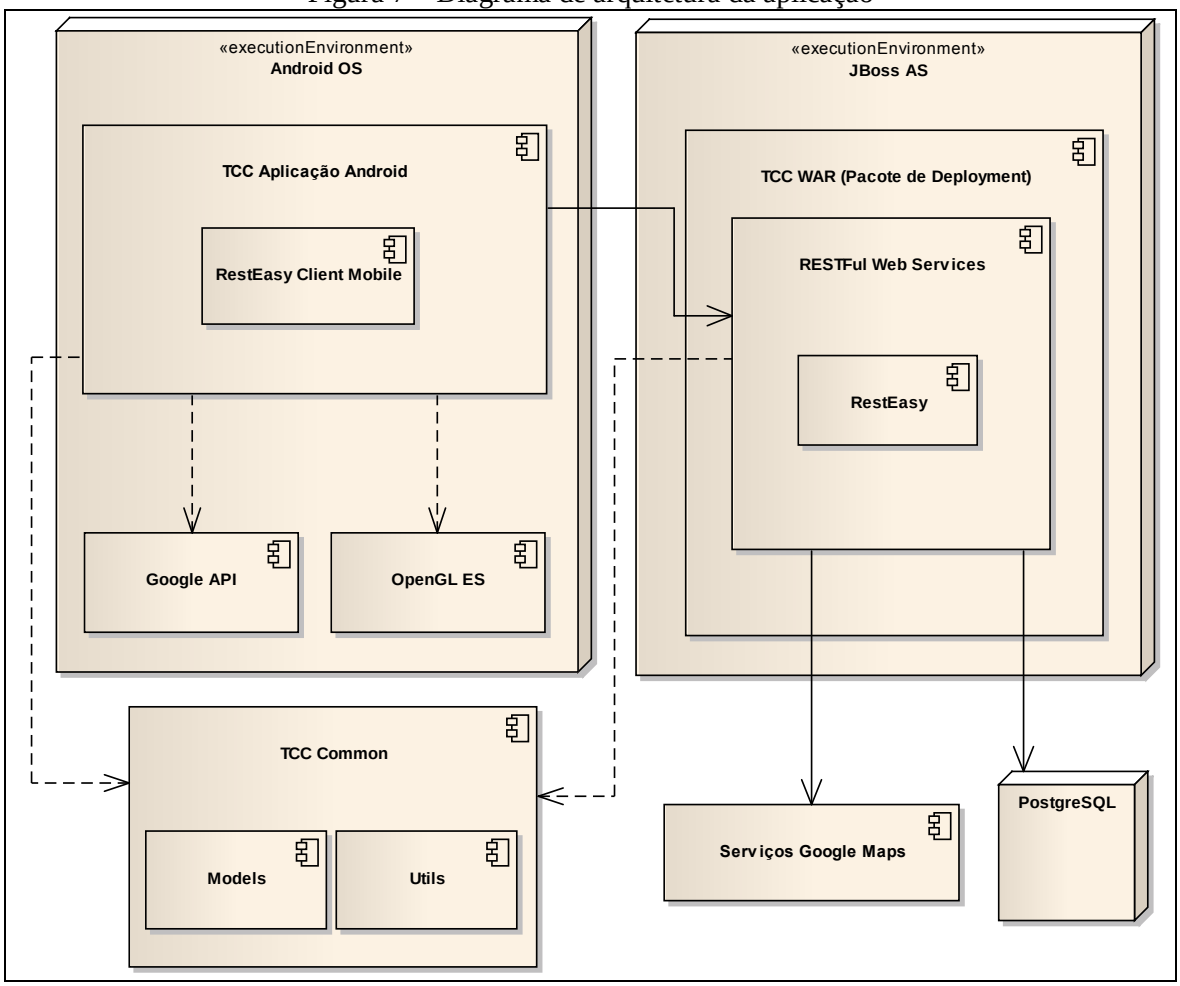
Já o projeto `TCC Aplicação Android` contém a aplicação que rodará no sistema operacional Android e com a qual o usuário irá interagir, contendo: formulários, listas, mapas e a tela de RA. Para a implementação de formulários e listas foi utilizada a Google API. Já para a integração com mapas, foi utilizada a API do Google Maps a qual também faz parte do Google API. Para o desenvolvimento da tela de RA foi empregada a biblioteca OpenGL ES com o objetivo de renderizar elementos gráficos com alto desempenho. O projeto `TCC Aplicação Android` se comunica com o módulo de serviços através da biblioteca RestEasy Client Mobile, responsável por abstrair o consumo de *web services* em REST, sendo uma versão reduzida da API RestEasy Client, criada especialmente para ser utilizada em aplicações de dispositivos móveis.

Por último, o projeto `TCC Common` possui as classes de modelo, interfaces dos serviços REST e classes com métodos utilitários: como por exemplo tratamento de datas, números, etc. Esses conteúdos são utilizados tanto pelo projeto da aplicação Android quanto pelo projeto

web. Desta forma, esse projeto foi criado com o objetivo de centralizar os códigos compartilhados entre os projetos.

O diagrama de arquitetura do sistema pode ser visto na Figura 7.

Figura 7 – Diagrama de arquitetura da aplicação

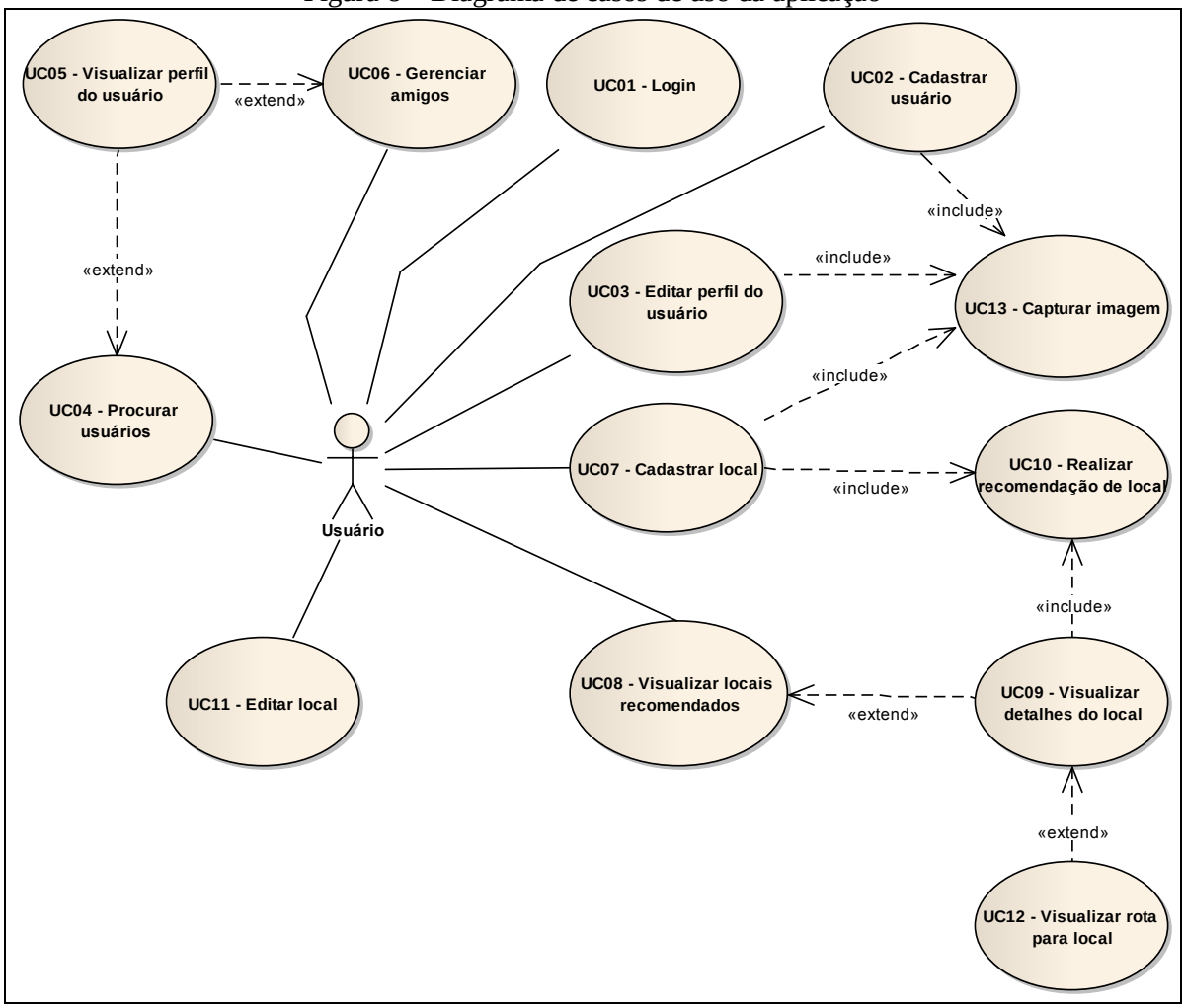


3.2.3 Casos de uso

A partir dos requisitos elicitados, obteve-se 13 casos de uso os quais são desempenhados pelo ator `Usuário`, que representa o utilizador da aplicação.

O diagrama de caso de uso foi desenvolvido observando os padrões da UML. Todos os casos de uso foram detalhados em cenários e vinculados a pelo menos um RF, facilitando a identificação do propósito do caso de uso e justificando a sua existência. A Figura 8 contém o diagrama de casos de uso da aplicação.

Figura 8 – Diagrama de casos de uso da aplicação



Os detalhes dos casos de uso da aplicação, como seus objetivos, requisitos atendidos e cenários de caso de uso podem ser visualizados no Apêndice A.

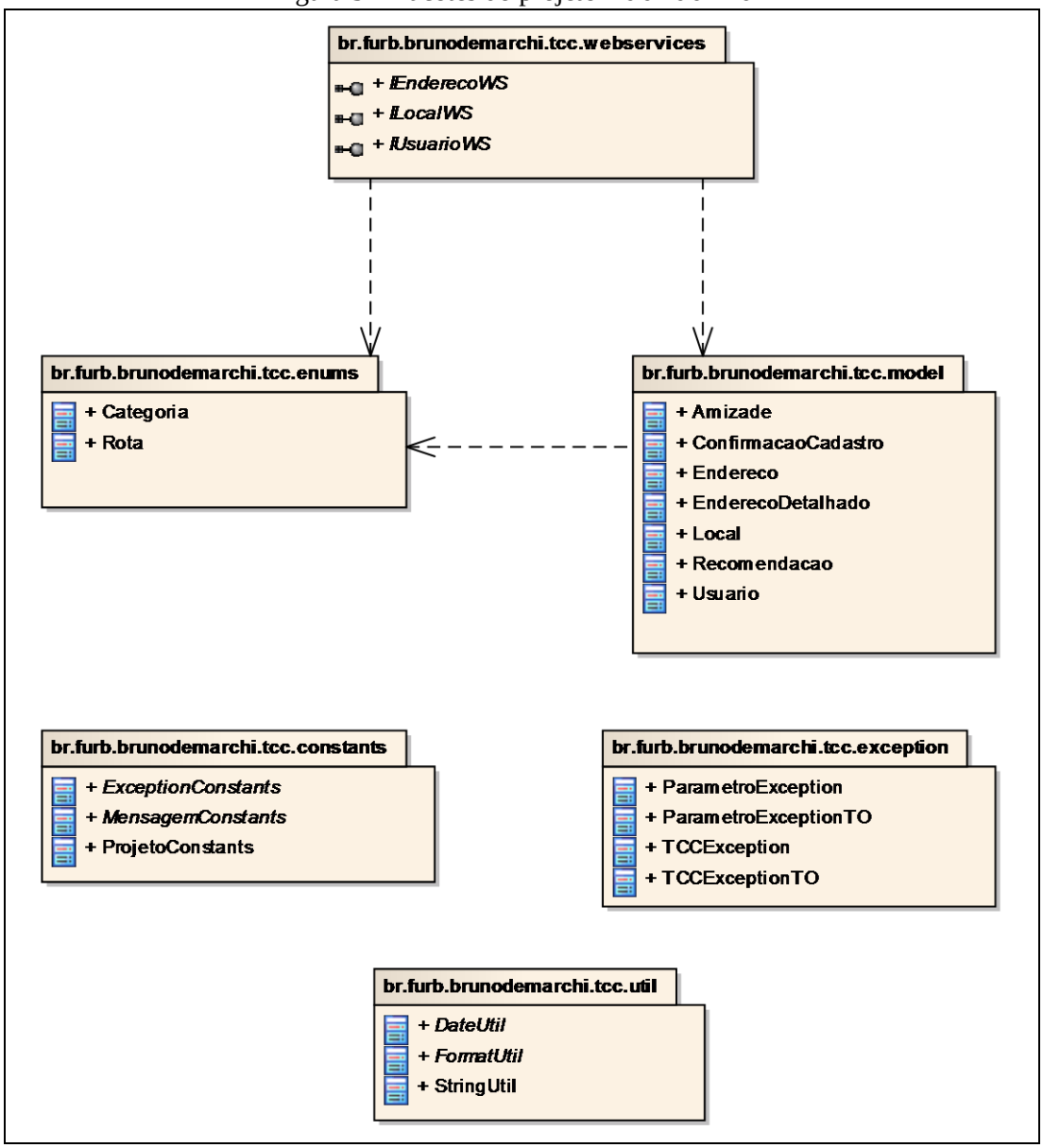
3.2.4 Diagramas de classe

A seguir são mostrados os diagramas de classe do projeto `TCC Common`, posteriormente do `TCC WAR` e por último do projeto `TCC Aplicação Android`.

3.2.4.1 Diagrama de classe do projeto `TCC Common`

Este projeto possui as classes de modelo, enumeradores, classes de exceção, constantes, interfaces dos serviços REST e classes com métodos utilitários. Os pacotes do projeto `TCC Common` podem ser vistos na Figura 9.

Figura 9 – Pacotes do projeto `TCC Common`



3.2.4.1.1 Pacote `br.furb.brunodemarchi.tcc.enums`

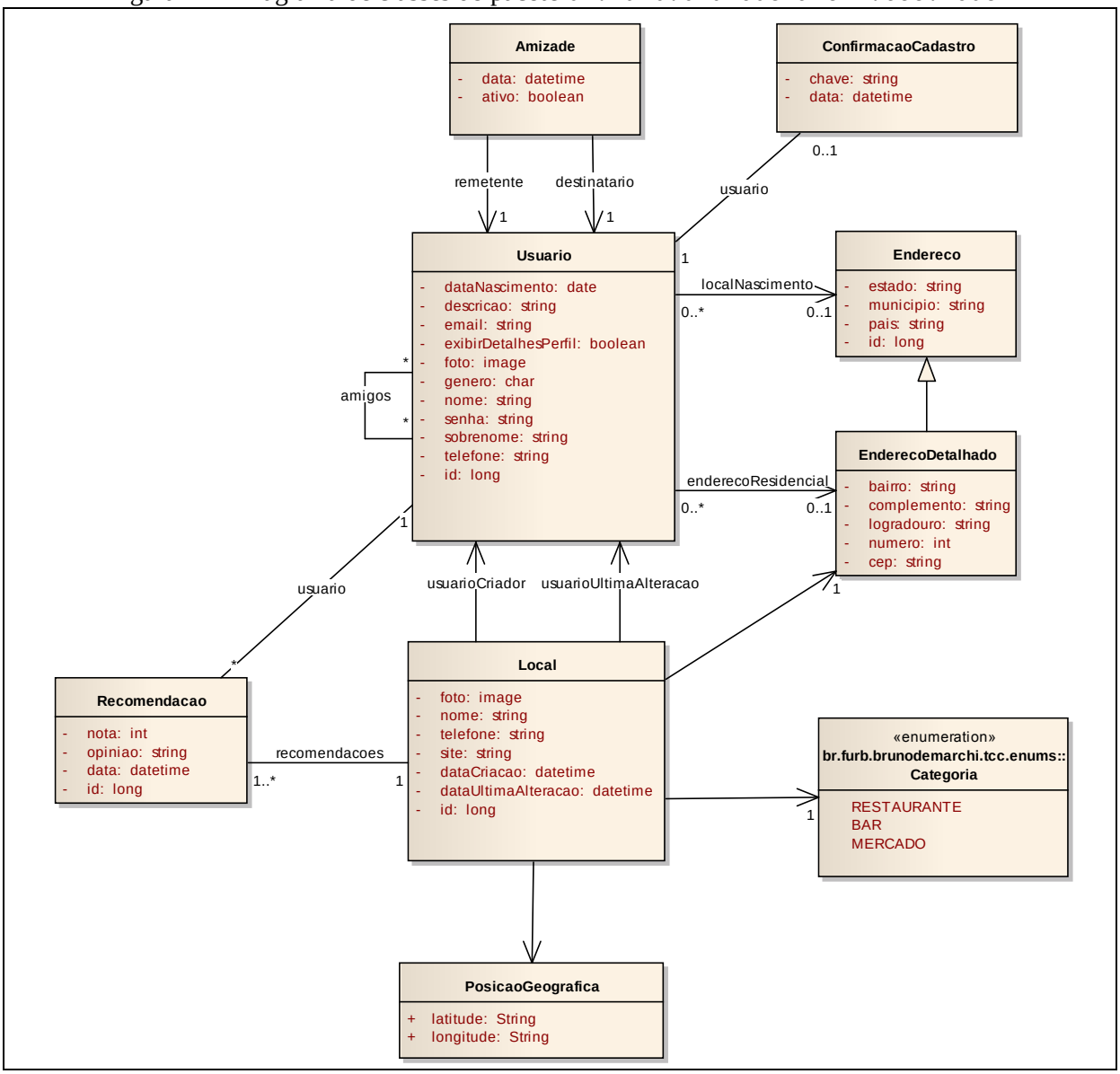
Este pacote possui os enumeradores utilizados no sistema e pode ser visto na Figura 10.

Figura 10 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.enums`

3.2.4.1.2 Pacote `br.furb.brunodemarchi.tcc.model`

Este pacote (Figura 11) possui as classes de modelo que representam os dados utilizados no sistema.

Figura 11 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.model`



3.2.4.1.3 Pacote `br.furb.brunodemarchi.tcc.webservices`

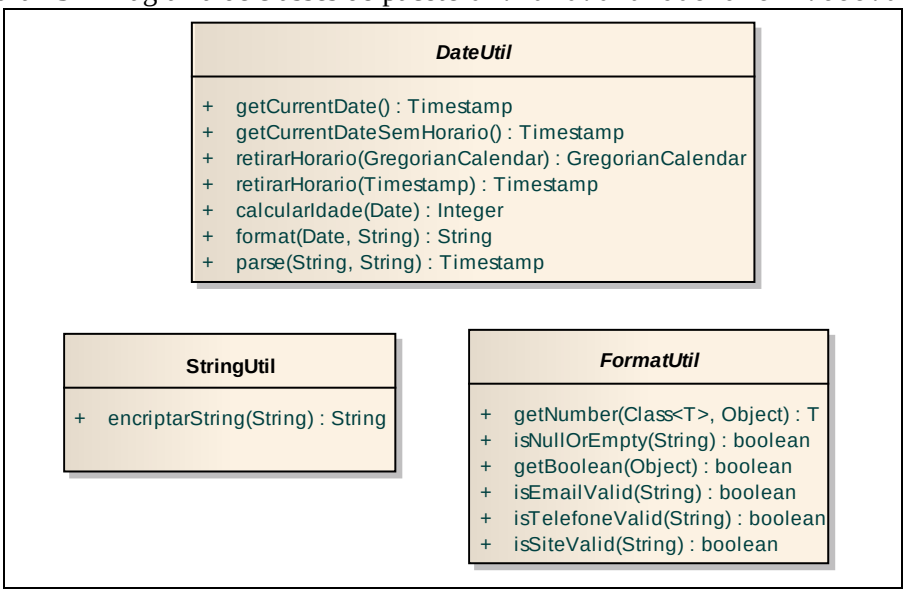
Este pacote possui as interfaces dos *web services* do sistema. Essas interfaces são implementadas pelo projeto `TCC WAR` e são utilizadas como interface de consumo por parte do projeto `TCC Aplicação Android`. O diagrama de classes desse pacote pode ser visualizado na Figura 12.

Figura 12 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.webservices`

3.2.4.1.4 Pacote `br.furb.brunodemarchi.tcc.util`

Este pacote (Figura 13) contém classes com método utilitários para tratamento de datas, números, *strings* e outros.

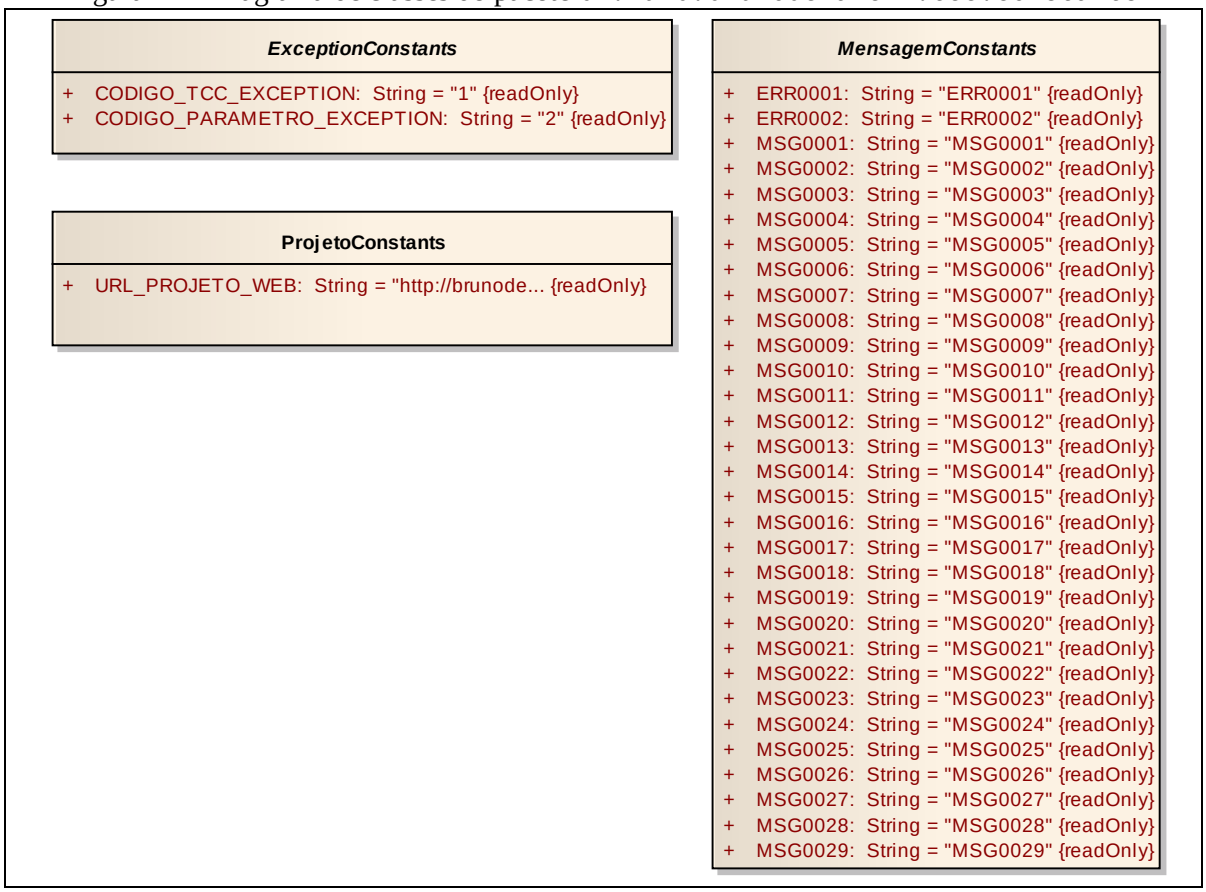
Figura 13 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.util`



3.2.4.1.5 Pacote `br.furb.brunodemarchi.tcc.constants`

Este pacote contém as constantes utilizadas no sistema, tais como códigos de mensagens, endereços *web* e códigos de exceção. Este diagrama de classes pode ser visualizado na Figura 14.

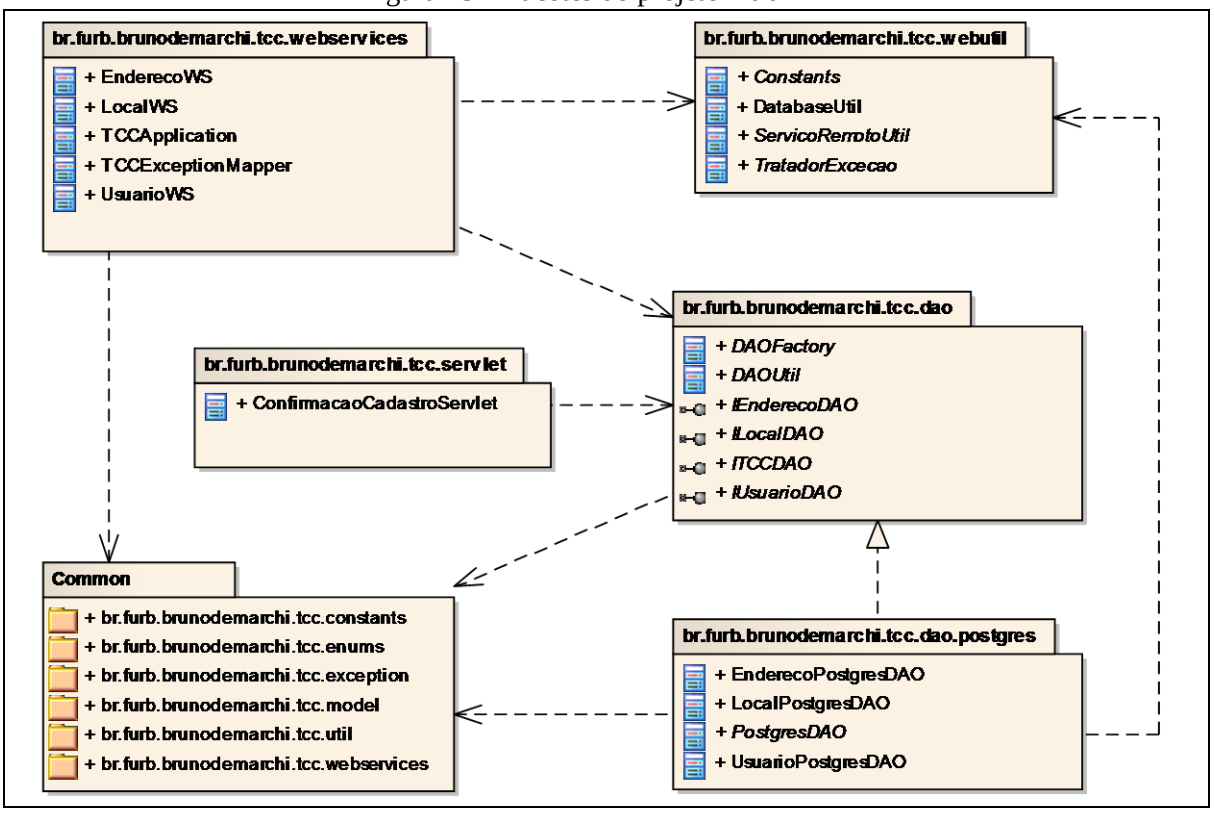
Figura 14 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.constants`



3.2.4.2 Diagrama de classe do servidor web

Este projeto possui toda a camada web necessária para a aplicação, incluindo implementação e disponibilização de *web services* que contém as regras de negócio do sistema, além de possuir as classes que realizam a manipulação das informações no banco de dados utilizando os padrões de projeto *Data Access Object* (DAO) e *Abstract Factory*. Por último, este projeto disponibiliza uma página web para que a confirmação do cadastro do usuário possa ser realizada em qualquer tecnologia que possua acesso a internet - não obrigatoriamente um dispositivo com Android. Os pacotes do projeto `TCC WAR` podem ser visualizados na Figura 15.

Figura 15 – Pacotes do projeto `TCC WAR`

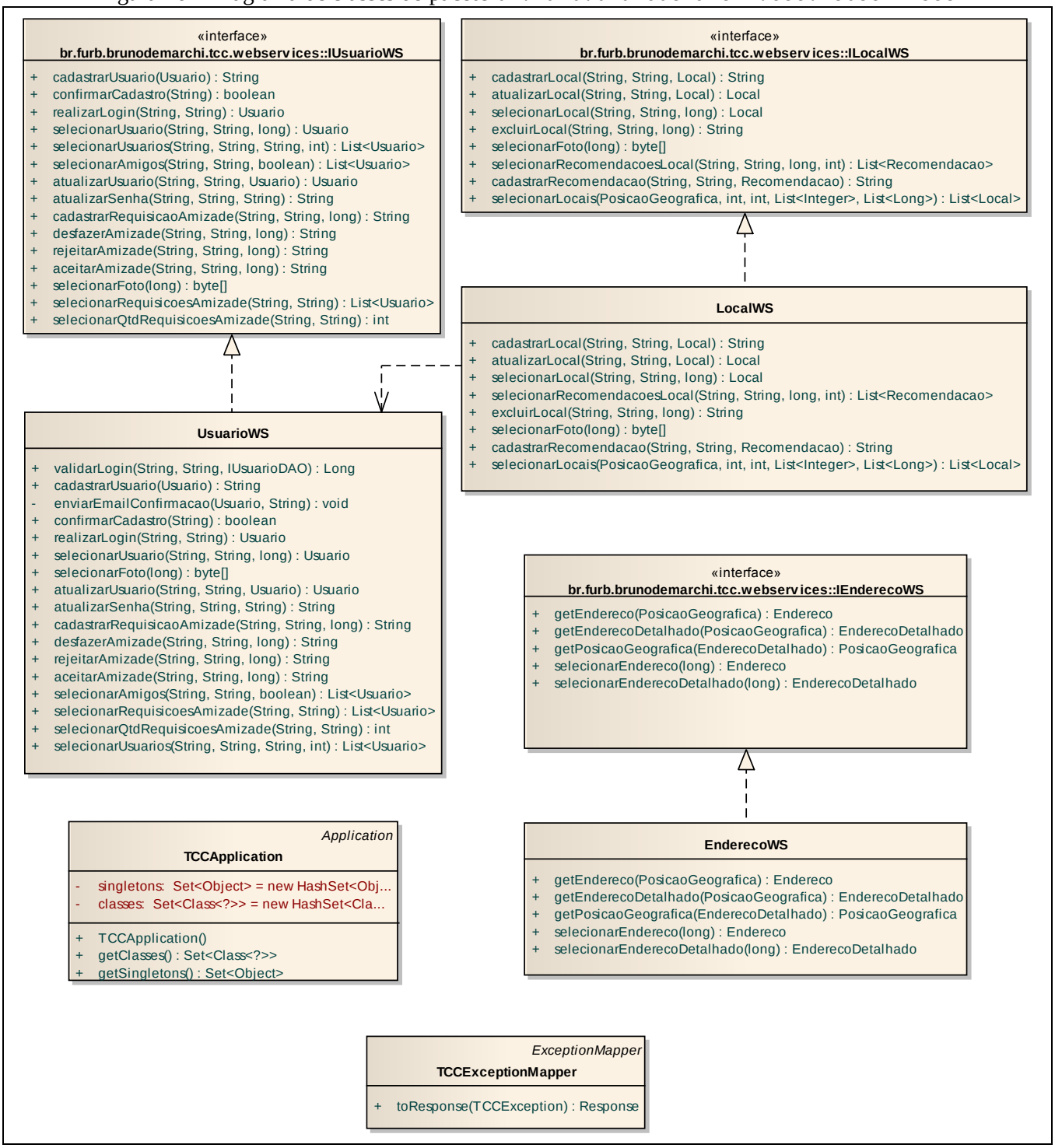


3.2.4.2.1 Pacote `br.furb.brunodemarchi.tcc.webservices`

Este pacote contém as implementações dos *web services* definidos no projeto `TCC Common`. Estes *web services* foram separados em três classes, cada uma responsável por tratar tipos de informações específicas. A classe `UsuarioWS` possui tratamentos relacionados ao usuário, enquanto que a classe `LocalWS` trata sobre locais e recomendações. Por último, a classe `EnderecoWS` tem como objetivo manipular endereços.

Este pacote também contém as classes `TCCApplication`, responsável por cadastrar os *web services* publicados, e `TCCExceptionMapper`, cujo objetivo é tratar o tipo de resposta *HyperText Transfer Protocol* (HTTP) em casos de exceções. Este diagrama de classes pode ser visualizado na Figura 16.

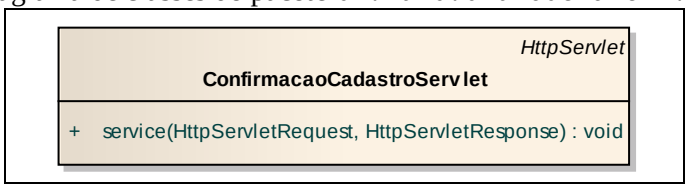
Figura 16 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.webservices`



3.2.4.2.2 Pacote `br.furb.brunodemarchi.tcc.servlet`

Este pacote (Figura 17) contém uma `Servlet` responsável por tratar as confirmações de cadastro de usuários.

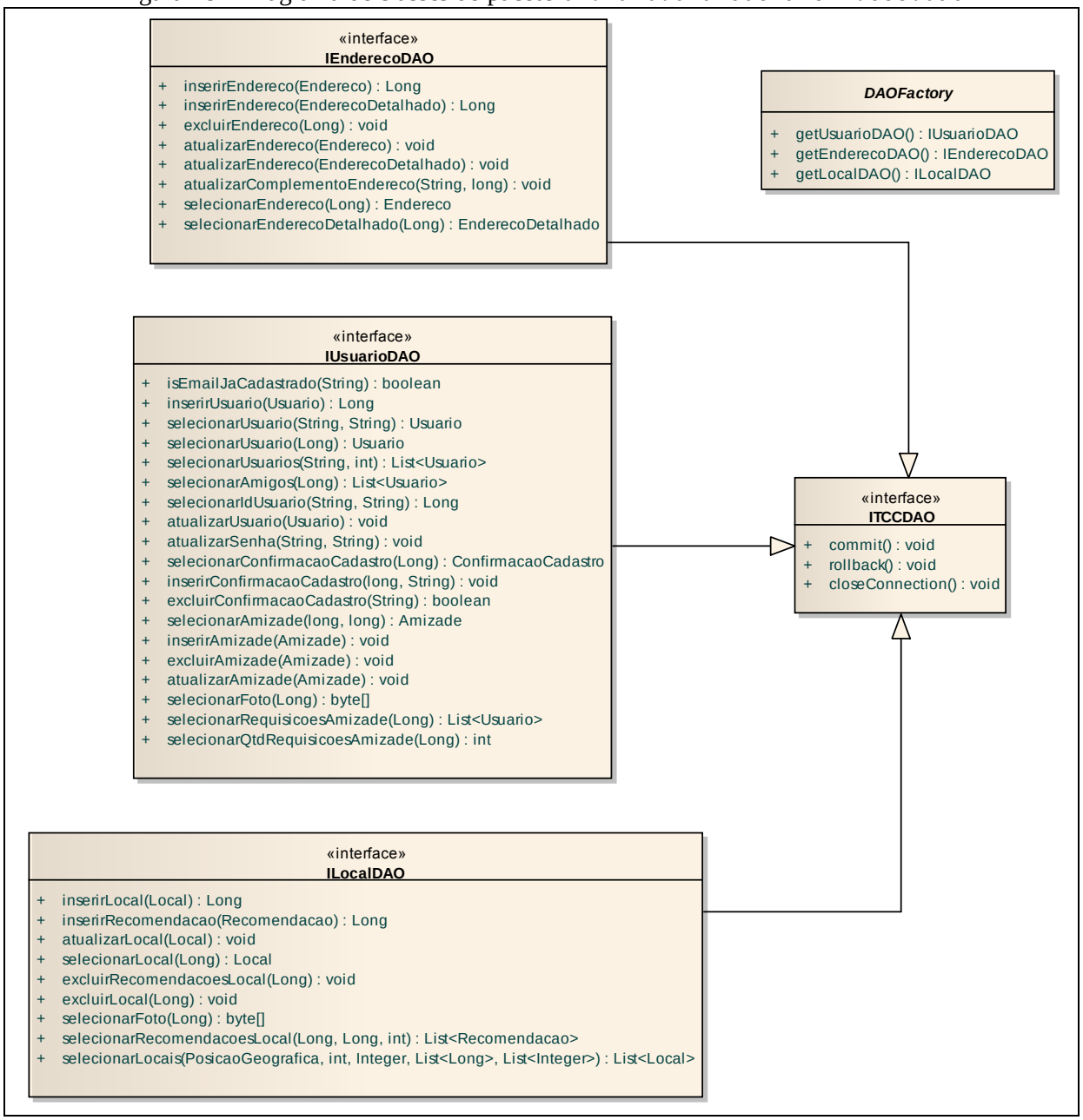
Figura 17 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.servlet`



3.2.4.2.3 Pacote `br.furb.brunodemarchi.tcc.dao`

Este pacote contém as interfaces que definem os métodos responsáveis pela manipulação das informações no banco de dados. Além disso possui a classe `DAOFactory`, responsável por abstrair a obtenção de instâncias das classes DAO. Este diagrama de classes pode ser visualizado na Figura 18.

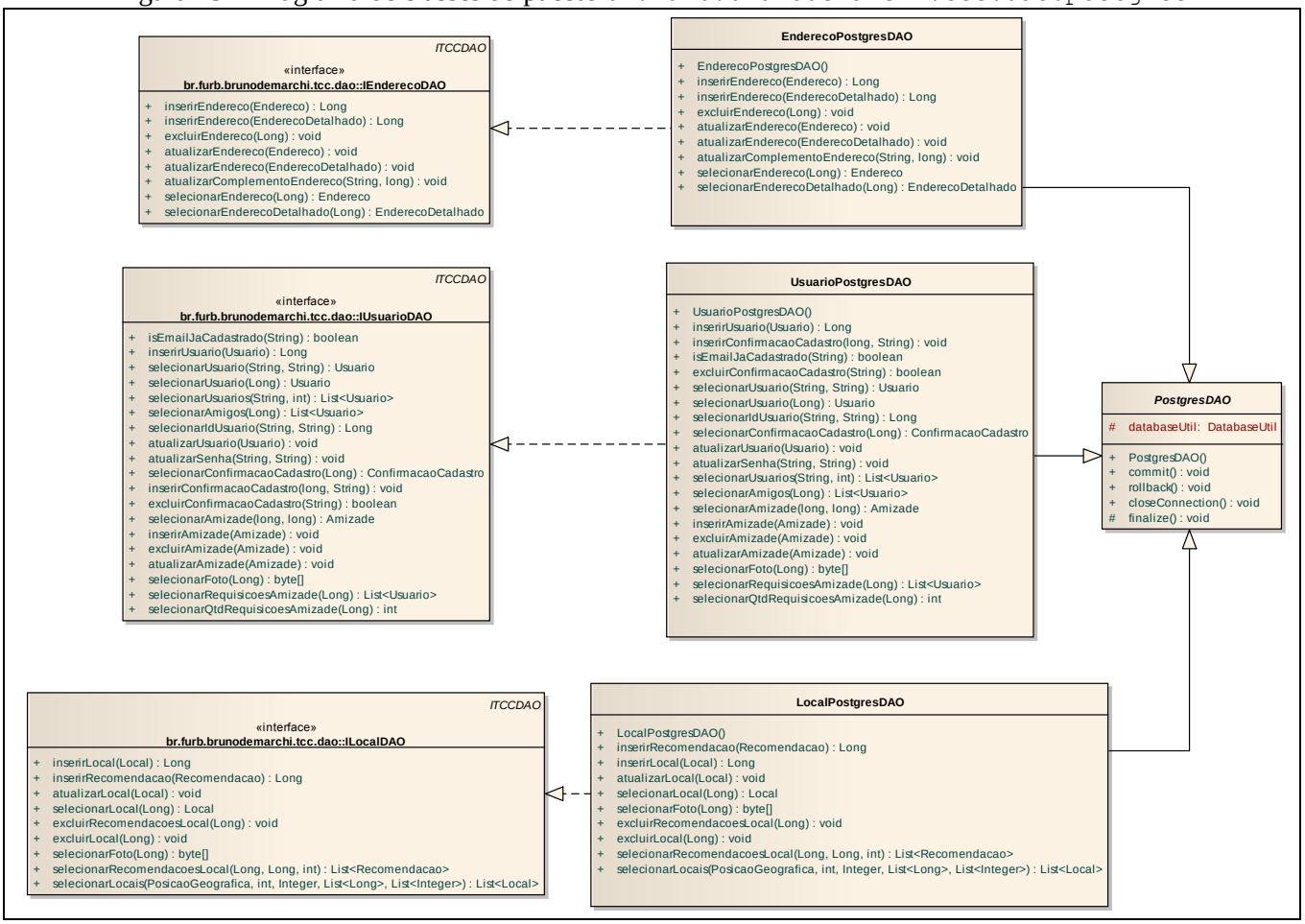
Figura 18 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.dao`



3.2.4.2.4 Pacote `br.furb.brunodemarchi.tcc.dao.postgres`

Este pacote possui as implementações para manipulação das informações no banco de dados PostgreSQL, seguindo as interfaces especificadas no pacote anterior. Este diagrama de classes pode ser visualizado na Figura 19.

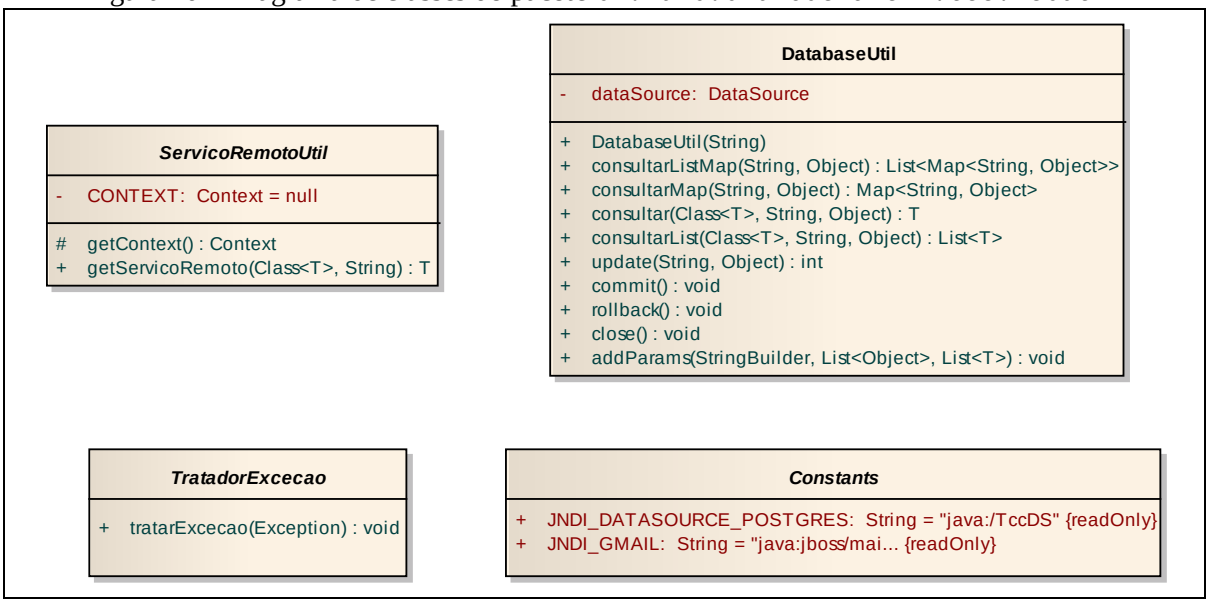
Figura 19 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.dao.postgres`



3.2.4.2.5 Pacote `br.furb.brunodemarchi.tcc.webutil`

Este pacote (Figura 20) contém classes com métodos utilitários específicos para o projeto web. A classe `ServicoRemotoUtil` realiza a busca de instâncias de objetos disponíveis no servidor de aplicação através dos seus nomes *Java Naming and Directory Interface* (JNDI), estes definidos na classe `Constants`. A classe `TratadorExcecao` tem como objetivo tratar exceções inesperadas e transformá-las em exceções do tipo `TCCException`. Por último, a classe `DatabaseUtil` possui métodos para abstrair o uso da biblioteca Apache DBUtils e realizar consultas e alterações em um determinado `DataSource`.

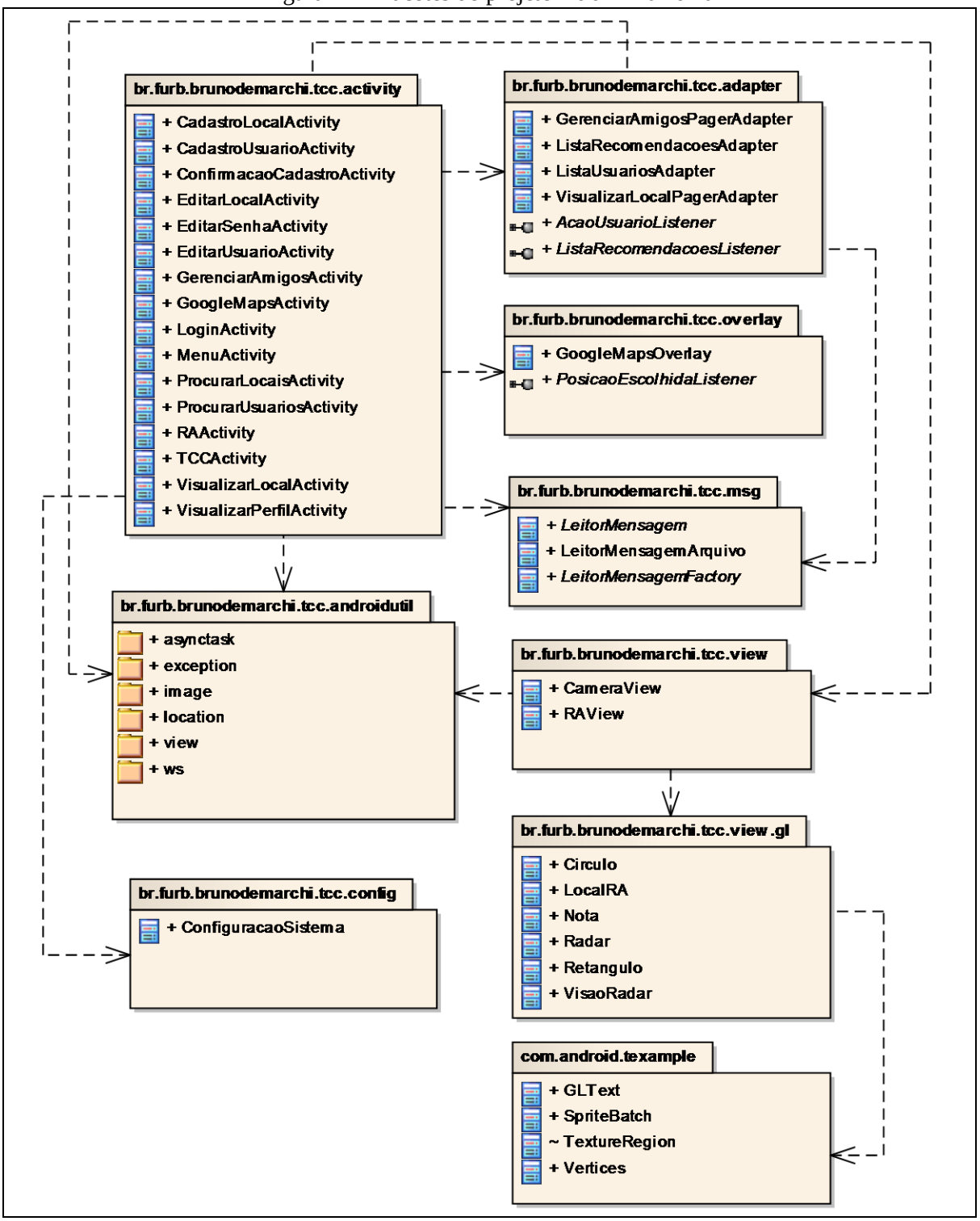
Figura 20 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.webutil`



3.2.4.3 Diagrama de classe da aplicação Android

Este projeto possui todos os pacotes necessários para a implementação da aplicação Android, incluindo os componentes `Activity`, `Adapter`, `Overlay` e `View`. Este projeto também possui um pacote com diversos recursos utilitários que serão detalhados mais a frente, além de leitores de mensagem, configurações do sistema, objetos do OpenGL ES e também o pacote `com.android.texample`, no qual há código de terceiros utilizado para abstração na renderização de texto com o OpenGL ES. Os pacotes deste projeto podem ser visualizados na Figura 21.

Figura 21 – Pacotes do projeto `TCC Android`

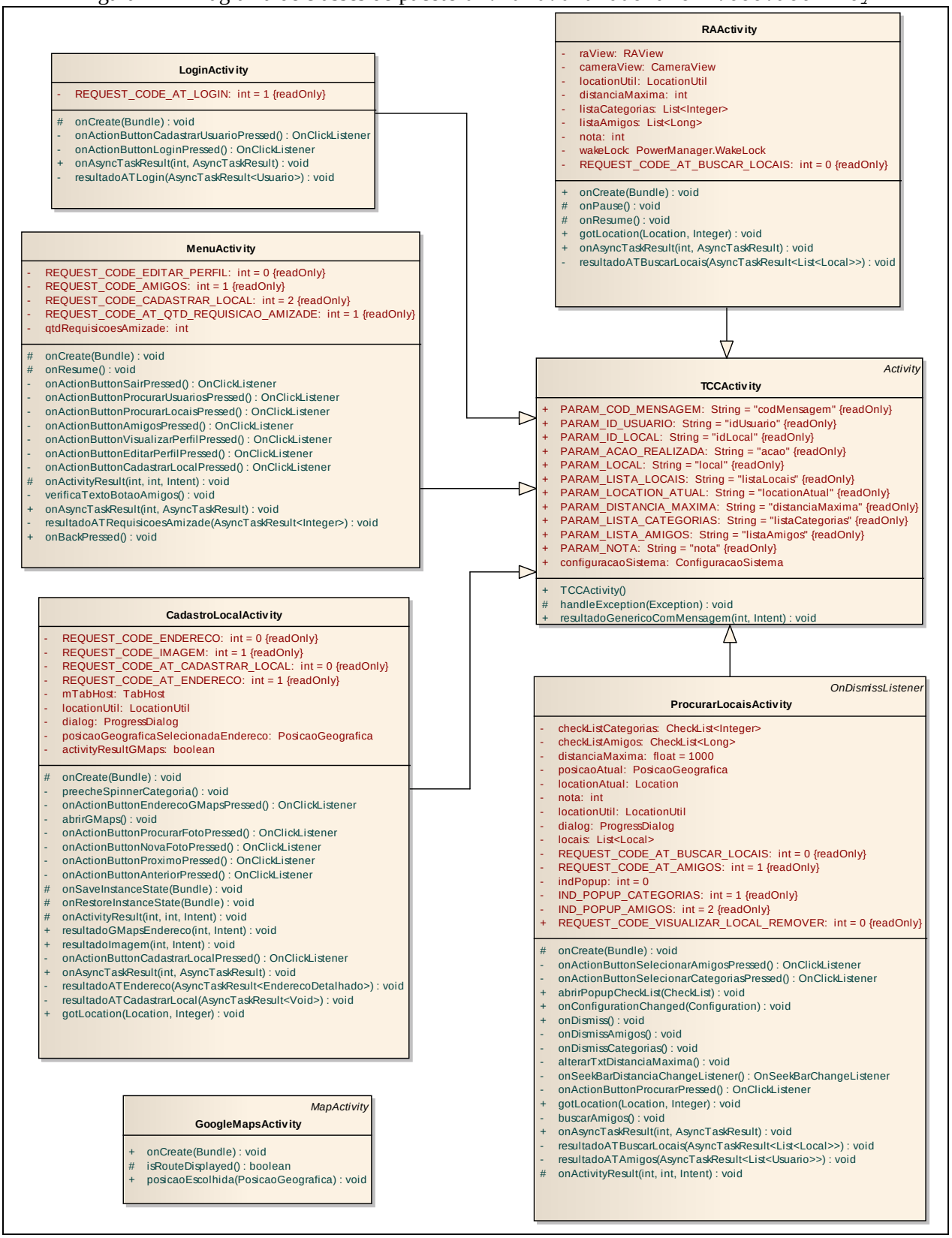


3.2.4.3.1 Pacote `br.furb.brunodemarchi.tcc.activity`

Este pacote possui as classes com os componentes `Activity` que fornecem as telas da aplicação com as quais o usuário irá interagir. Com exceção da `GoogleMapsActivity`, utilizada para apresentar ao usuário o mapa do Google Maps, todas as atividades estendem de `TCCActivity`, criada para centralizar comportamentos comuns entre os componentes

`Activity` da aplicação. Na Figura 22, é possível observar as principais atividades da aplicação, sendo que as demais foram ocultadas para melhor visualização do diagrama.

Figura 22 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.activity`



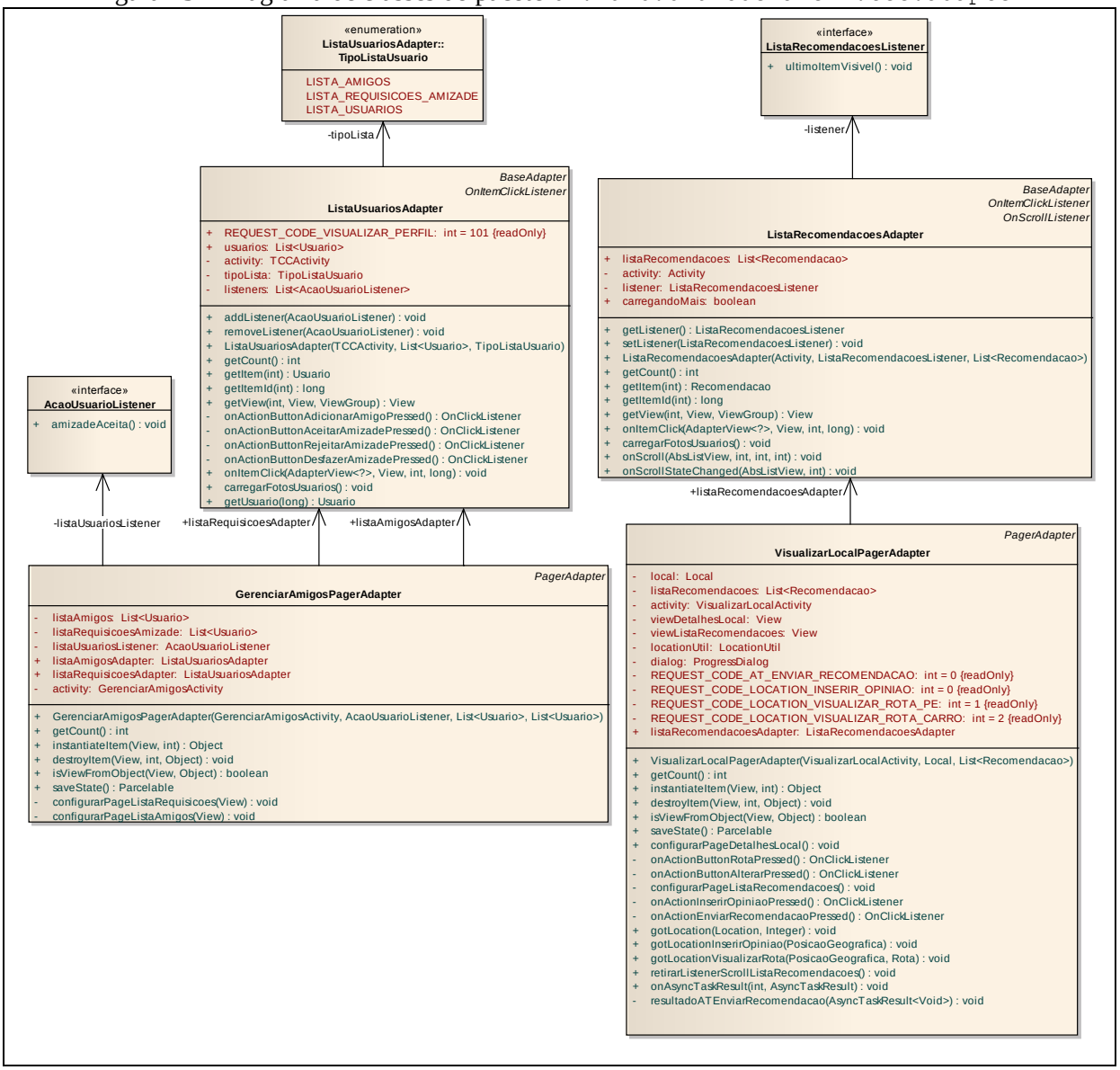
3.2.4.3.2 Pacote `br.furb.brunodemarchi.tcc.adapter`

Este pacote (Figura 23) contém os adaptadores necessários para utilização de componentes como o `PagerAdapter` e o `BaseAdapter`.

O `PagerAdapter` possibilita a utilização de duas páginas, sendo que o usuário pode passar de uma para outra deslizando a tela horizontalmente. Este componente foi extendido no `GerenciarAmigosPagerAdapter`, onde em uma página há a lista de requisições de amizade e na outra há a lista de amigos do usuário. Este componente aciona o evento `amizadeAceita()` dos seus observadores (`AcaoUsuarioListener`) quando o usuário aceita uma requisição de amizade. Além disso, o `PagerAdapter` foi estendido também no `VisualizarLocalPagerAdapter`, onde em uma página há informações sobre o local, enquanto na outra há a lista de recomendações realizadas sobre o local.

O `BaseAdapter` possibilita a implementação de uma lista onde você pode customizá-la de forma a ter uma visualização e eventos próprios. Este componente foi estendido pelo `ListaUsuarioAdapter`, que provê uma lista de usuários em três formas diferentes: lista de usuários, lista de amigos e lista requisições de amizade. O `BaseAdapter` também foi estendido pelo `ListaRecomendacoesAdapter`, que fornece uma lista das recomendações realizadas sobre um local e aciona o evento `ultimoItemVisivel()` dos seus observadores (`ListaRecomendacoesListener`) quando o usuário deslizou até o final da lista.

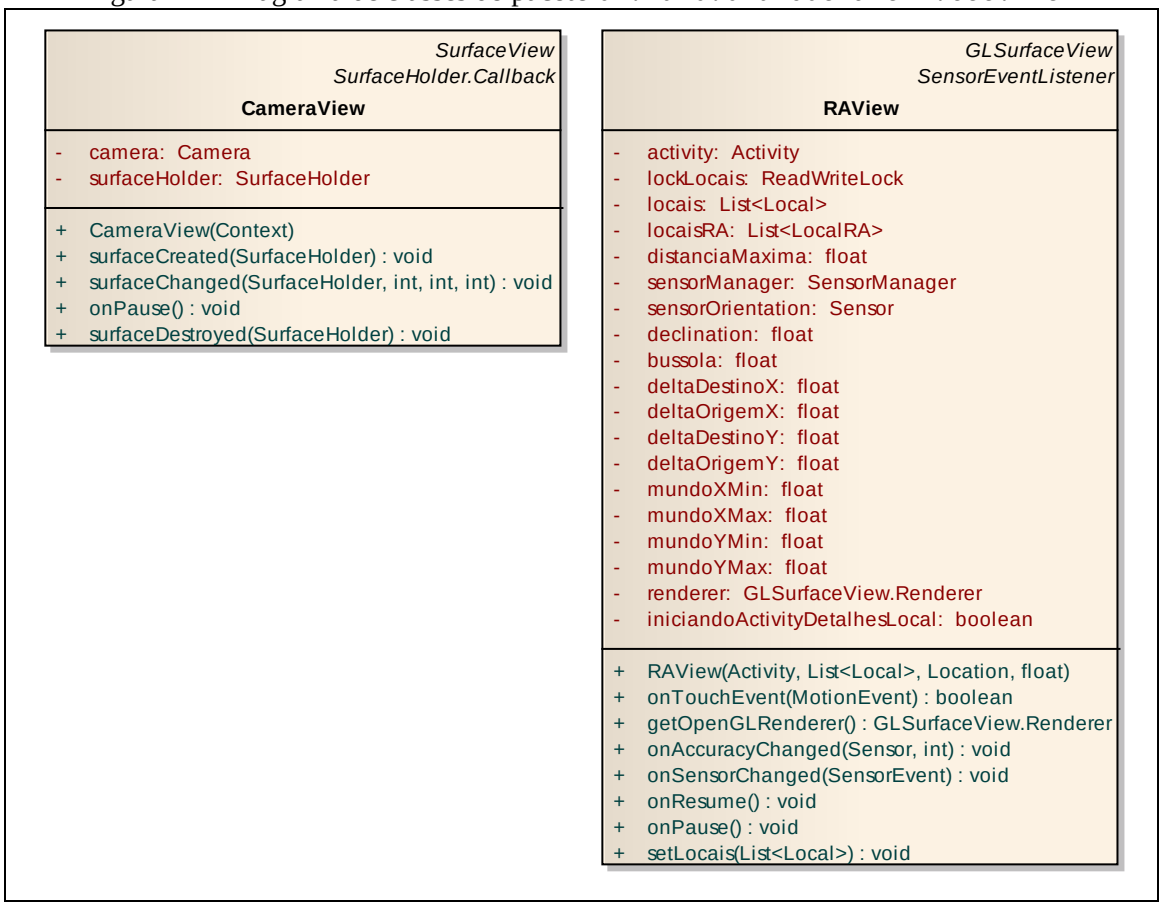
Figura 23 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.adapter`



3.2.4.3.3 Pacote `br.furb.brunodemarchi.tcc.view`

Este pacote contém as telas as quais são criadas dinâmicamente e que são utilizadas para montagem da `RAActivity`, responsável pela apresentação dos locais utilizando RA. A `CameraView` tem como objetivo apresentar a imagem da câmera do dispositivo em tempo real, enquanto que a `RAView` tem como objetivo renderizar os objetos do OpenGL ES de acordo com as atualizações obtidas da bússola do dispositivo. Este diagrama de classes pode ser visualizado na Figura 24.

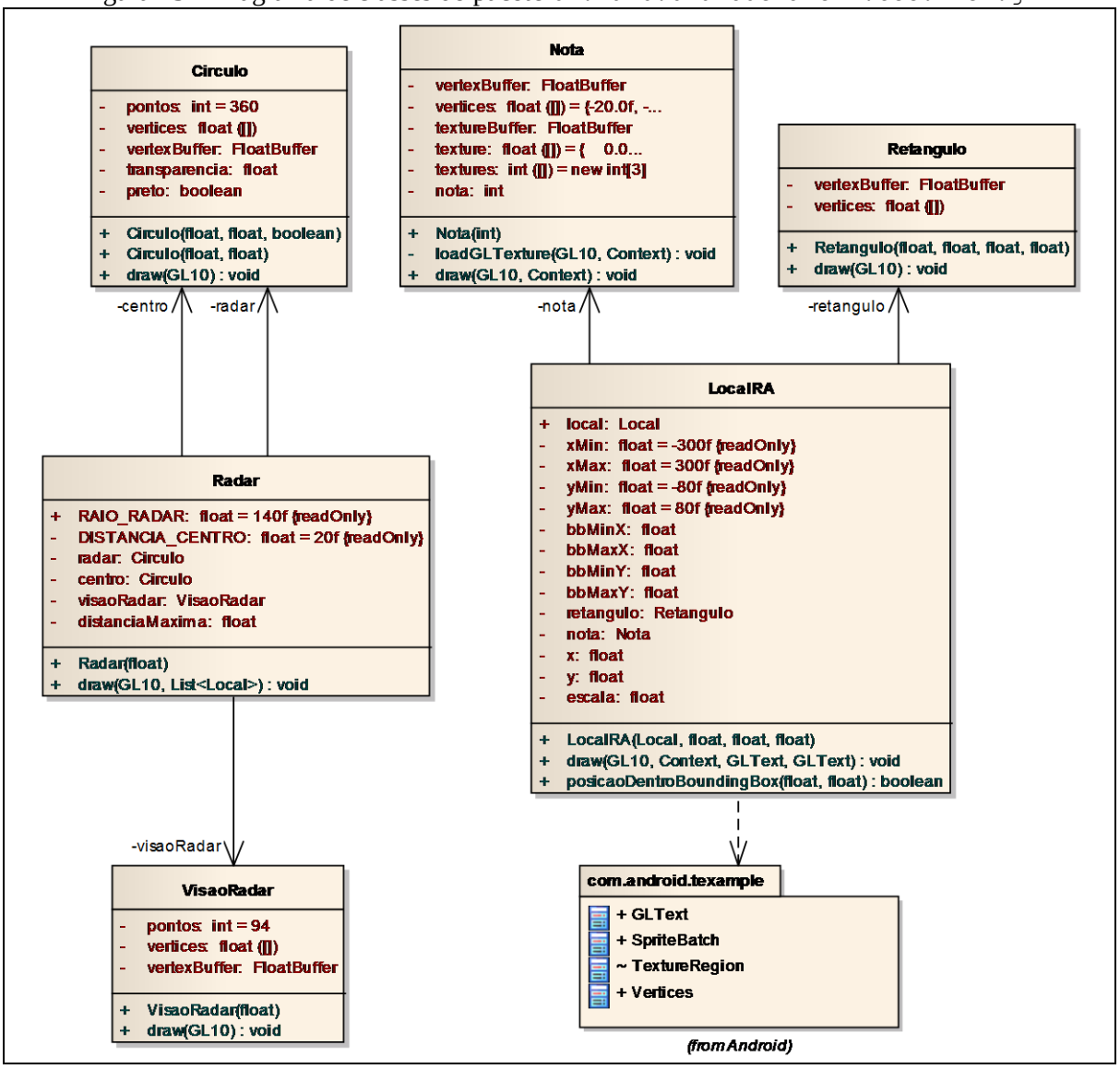
Figura 24 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.view`



3.2.4.3.4 Pacote `br.furb.brunodemarchi.tcc.view.gl`

Este pacote contém os objetos OpenGL que são renderizados pela `RAView` apresentada anteriormente. O objeto `Radar` é responsável por renderizar um `Circulo` que contém os locais encontrados através de pontos, com um visor (`VisaoRadar`) indicando quais deles estão visíveis em um determinado momento. Já o objeto `LocalRA` representa graficamente um local encontrado, apresentando um `Retangulo` sobreposto pela `Nota` do local e algumas informações textuais como nome e categoria. Para a renderização desses textos foi utilizado o pacote `com.android.texample` - o qual possui códigos de terceiros. Este pacote pode ser visualizado na Figura 25.

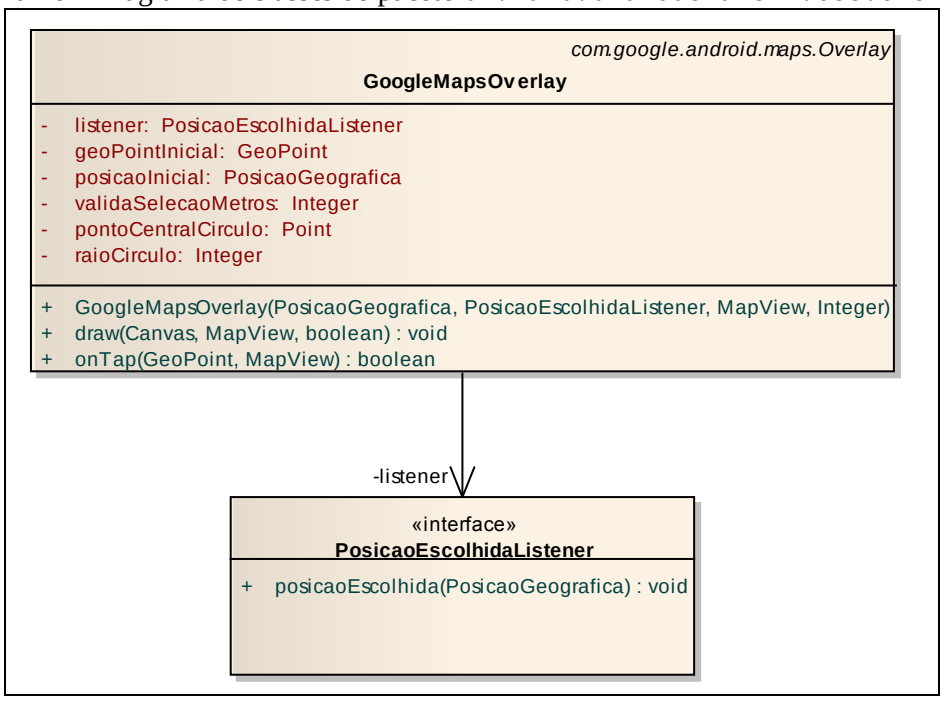
Figura 25 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.view.gl`



3.2.4.3.5 Pacote `br.furb.brunodemarchi.tcc.overlay`

Este pacote (Figura 26) contém a classe `GoogleMapsOverlay`, que tem como objetivo tratar a visualização e os eventos que ocorrem quando o usuário está utilizando o Google Maps dentro de alguma `Activity` da aplicação. Quando uma posição no mapa é escolhida, é chamado o método `posicaoEscolhida(PosicaoGeografica)` do observador (`PosicaoEscolhidaListener`).

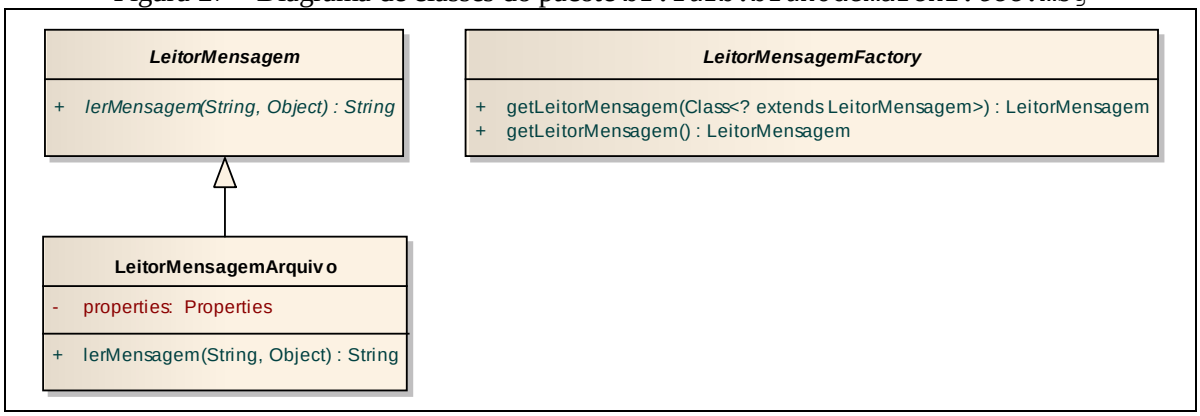
Figura 26 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.overlay`



3.2.4.3.6 Pacote `br.furb.brunodemarchi.tcc.msg`

Este pacote possui as classes responsáveis por obter as mensagens da aplicação de acordo com o código requisitado. Foi utilizado o padrão de projeto *Abstract Factory* para abstrair a obtenção de uma instância da interface `LeitorMensagem`. A classe `LeitorMensagemArquivo` implementa esta interface, tendo como fonte das mensagens um arquivo texto com a relação chave/valor. Este diagrama de classes pode ser visualizado na Figura 27.

Figura 27 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.msg`

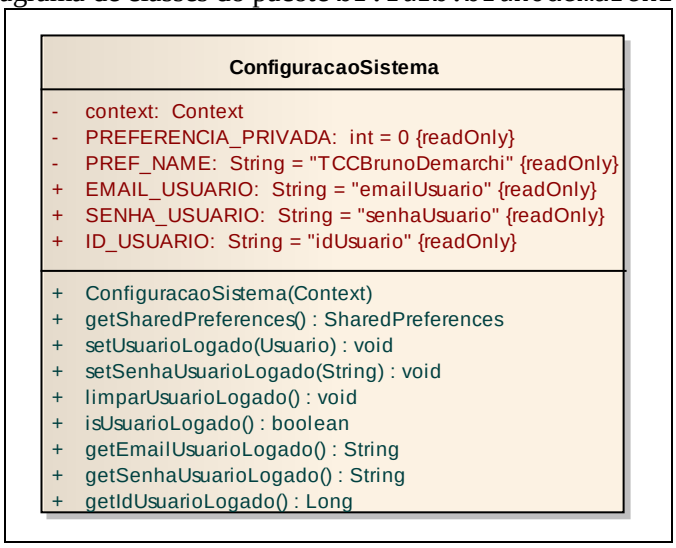


3.2.4.3.7 Pacote `br.furb.brunodemarchi.tcc.config`

Este pacote contém a classe `ConfiguracaoSistema`, responsável por persistir as configurações da aplicação para que quando o usuário saia e entre da aplicação não tenha que

informar, por exemplo, seu usuário e senha após já ter realizado o login na aplicação. O diagrama pode ser visualizado na Figura 28.

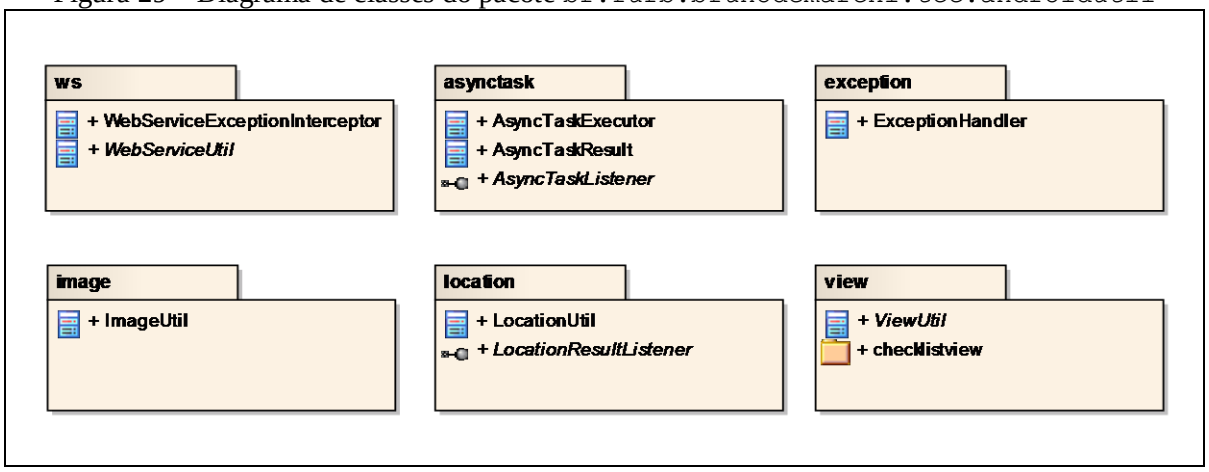
Figura 28 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.config`



3.2.4.3.8 Pacote `br.furb.brunodemarchi.tcc.androidutil`

Este pacote (Figura 29) possui classes utilitárias específicas para o projeto Android e foi dividido em subpacotes para melhor organização.

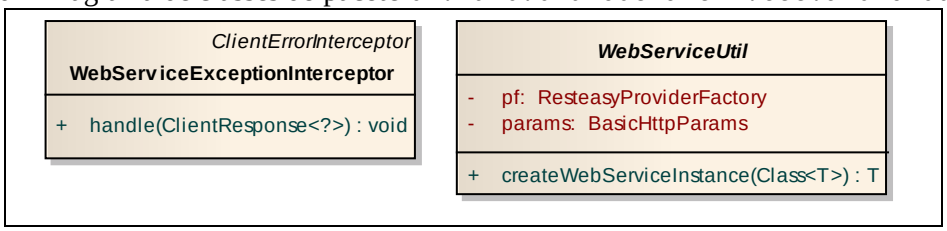
Figura 29 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.androidutil`



3.2.4.3.8.1 Pacote `br.furb.brunodemarchi.tcc.androidutil.ws`

Este pacote, visível na Figura 30, possui a classe `WebServiceUtil` que tem como objetivo abstrair a obtenção das instâncias para chamadas dos *web services* do projeto `TCC WAR` utilizando a biblioteca RestEasy Mobile. Já a classe `WebServiceExceptionInterceptor` é responsável por receber respostas HTTP que contém erros e transformá-las em exceções do sistema.

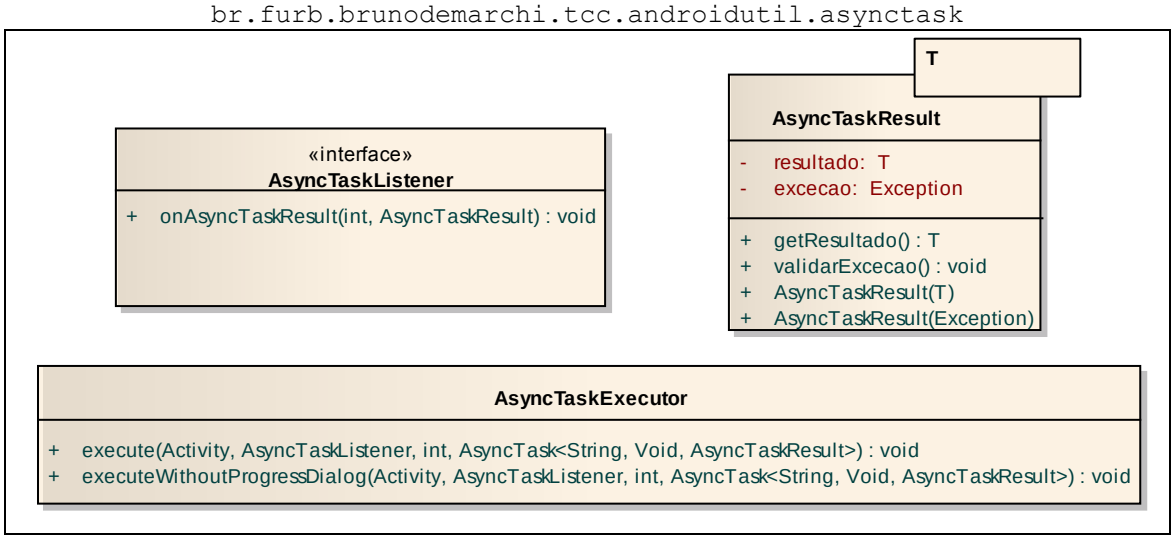
Figura 30 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.androidutil.ws`



3.2.4.3.8.2 Pacote `br.furb.brunodemarchi.tcc.androidutil.asynctask`

Este pacote (Figura 31) adiciona mais uma camada na execução de processos concorrentes, onde o código concorrente é passado para o método `AsyncTaskExecutor.execute()` ou `AsyncTaskExecutor.executeWithoutProgressDialog()`, que se encarrega de executá-lo e, no caso do primeiro, mostrar um diálogo informando o usuário da execução do processo concorrente. Quando o processo concorrente chega ao fim é chamado o método `onAsyncTaskResult()` do observador (`AsyncTaskListener`), passando como parâmetro o resultado encapsulado em um objeto `AsyncTaskResult`. Esse objeto `AsyncTaskResult` possui o método `validarExcecao()` que lançará uma `Exception` caso o processo concorrente tenha finalizado de forma inesperada.

Figura 31 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.androidutil.asynctask`

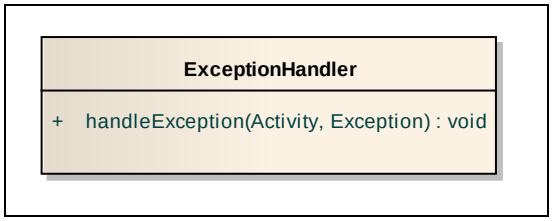


3.2.4.3.8.3 Pacote `br.furb.brunodemarchi.tcc.androidutil.exception`

Este pacote, visível na Figura 32, possui a classe `ExceptionHandler` que possui o método `handleException()` cujo objetivo é apresentar a mensagem adequada ao usuário em caso de exceções: para exceções do tipo `ParametroException` e `TCCException`, as

mensagens estão dentro da própria exceção; enquanto que para as demais exceções é apresentada uma mensagem padrão.

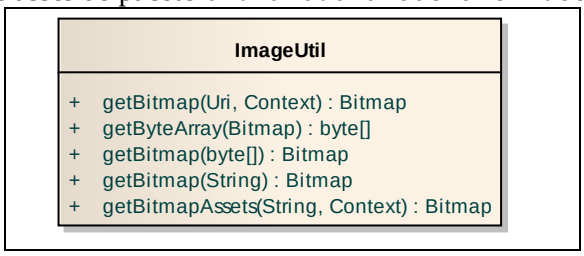
Figura 32 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.androidutil.exception`



3.2.4.3.8.4 Pacote `br.furb.brunodemarchi.tcc.androidutil.image`

Este pacote possui a classe `ImageUtil` que contém métodos utilitários para tratamento de imagens e pode ser visualizado na Figura 33.

Figura 33 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.androidutil.image`

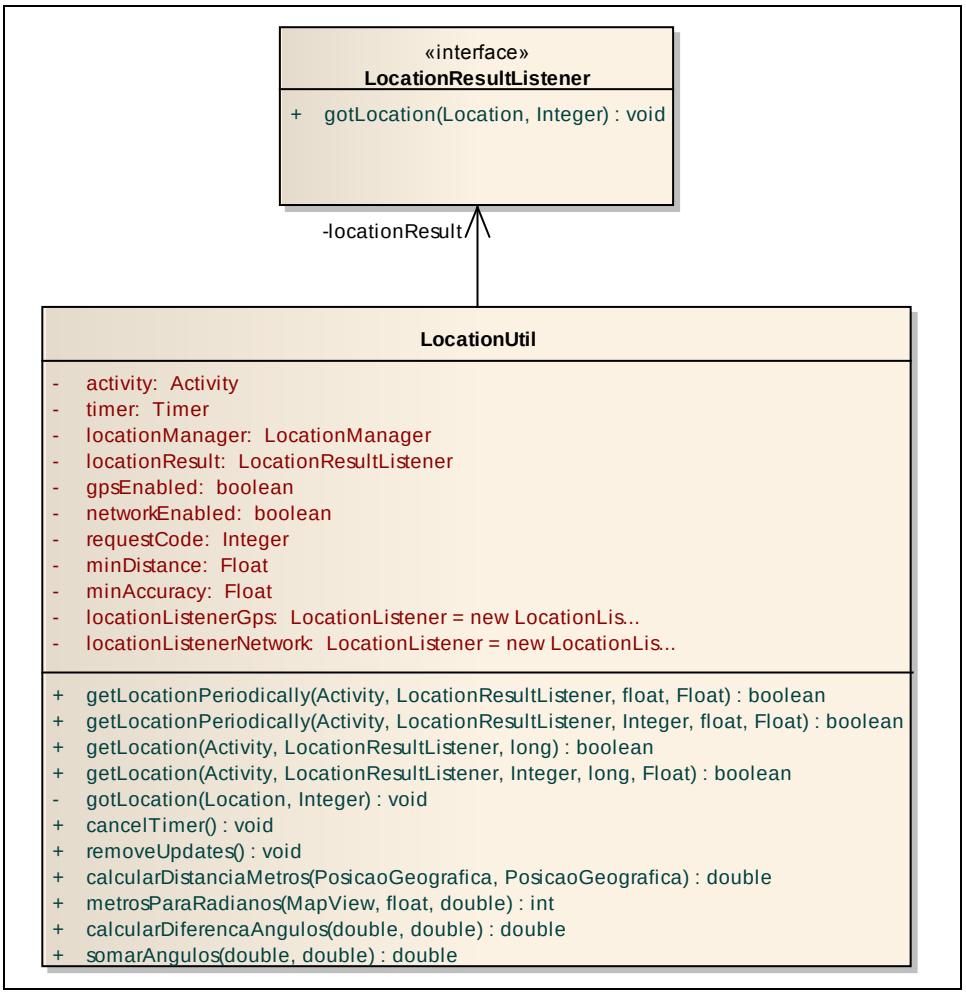


3.2.4.3.8.5 Pacote `br.furb.brunodemarchi.tcc.androidutil.location`

Este pacote (Figura 34) possui a classe `LocationUtil`, que tem como objetivo abstrair o uso da API para obtenção da localização do dispositivo e também prover métodos utilitários de cálculos envolvendo distâncias e ângulos.

Os métodos `getLocation()` tentam obter a localização através do GPS por um determinado tempo, onde caso o tempo passar e a posição não tiver sido obtida, utilizará a ultima localização recebida pelo provedor da rede. Já os métodos `getLocationPeriodically()` utilizam somente um provedor: caso ativado, o GPS; se não, a rede. A atualização da posição para esse método ocorre toda vez que o dispositivo se movimentar uma quantidade de metros - até que o método `removeUpdates()` seja chamado. Ambos os métodos chamam o evento `gotLocation()` do observador (`LocationResultListener`) quando a posição é obtida ou atualizada.

Figura 34 – Diagrama de classes do pacote
`br.furb.brunodemarchi.tcc.androidutil.location`



3.2.4.3.8.6 Pacote `br.furb.brunodemarchi.tcc.androidutil.view`

Este pacote (Figura 35) possui a classe `ViewUtil` com métodos utilitários para obter-se valores de diversos tipos de componentes do Android, como por exemplo o `TextEdit` e o `RatingBar`. Além desta classe, também possui o pacote `checklistview`, que pode ser visualizado na Figura 36. O pacote `checklistview` contém o componente `CheckList`, que recebe uma lista de `CheckableItem` e os mostra na tela - opcionalmente em uma *pop-up* - para que o usuário possa selecionar os itens que desejar. Para obter-se a lista dos itens selecionados têm-se o método `CheckList.getSelectedItems()`, enquanto que o método `CheckList.getSelectedValues()` retorna somente os valores dos itens selecionados.

Figura 35 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.androidutil.view`

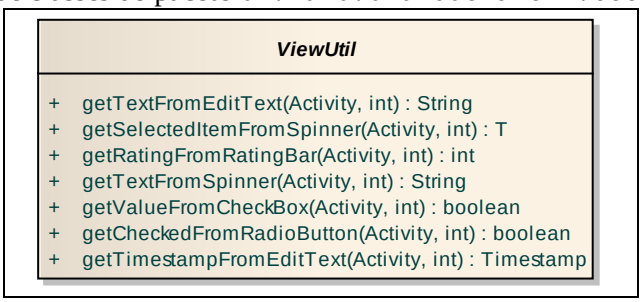
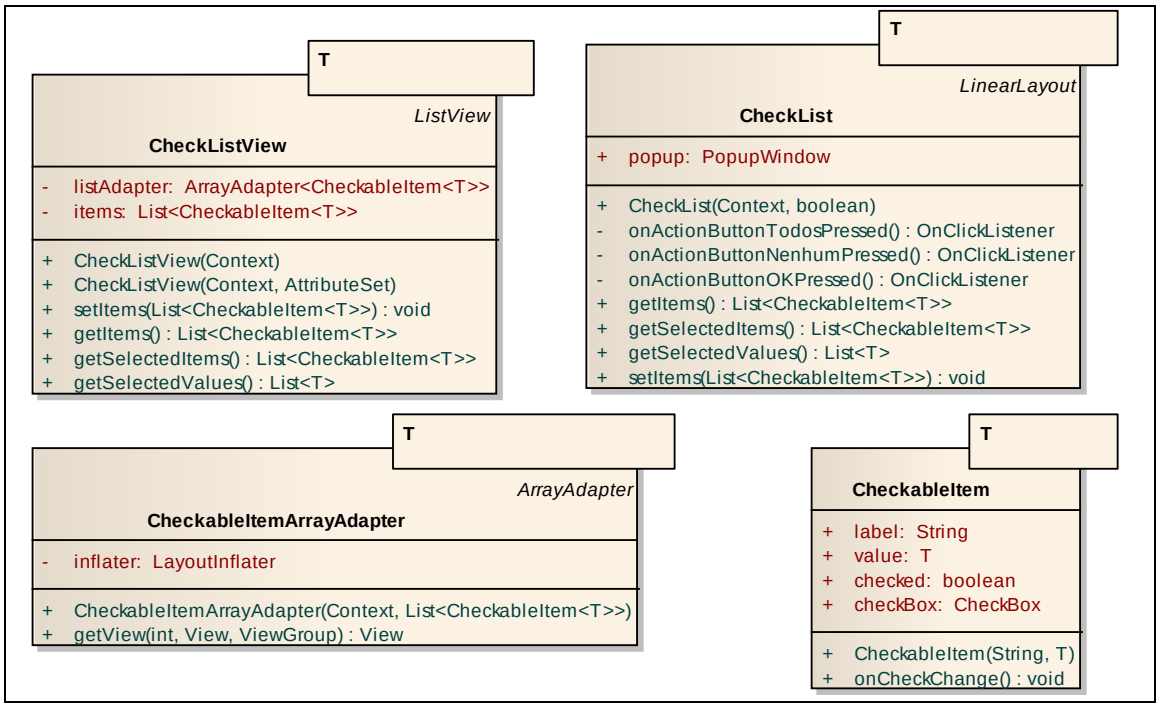


Figura 36 – Diagrama de classes do pacote `br.furb.brunodemarchi.tcc.androidutil.view.checklistview`



3.3 IMPLEMENTAÇÃO

A seguir são descritas as técnicas e ferramentas utilizadas na implementação, os detalhes das principais classes e rotinas da aplicação, assim como a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O sistema foi desenvolvido de forma que as regras de negócio fossem encapsuladas em um projeto com uma interface web com requisições e respostas no formato JSON, resultando em um sistema que utiliza o paradigma *Service-oriented architecture* (SOA). Para o desenvolvimento dos *web services* foi utilizada a biblioteca RestEasy que, através de anotações em métodos, publica serviços REST. Para o consumo destes *web services* por parte da aplicação Android foi utilizada a biblioteca RestEasy Client Mobile. Já para a renderização dos gráficos na tela de RA foi utilizada a biblioteca OpenGL ES.

O ambiente de desenvolvimento utilizado foi o Eclipse Juno em conjunto com os *plugins* do Android *Development Tools* (ADT). Para a aplicação Android, foi utilizada a API de desenvolvimento do Android na versão 4.0.4. Já o projeto web foi desenvolvido utilizando a especificação Java 2 *Platform Enterprise Edition* (J2EE).

Para execução e depuração da aplicação Android foi utilizado o *smartphone* Sony Xperia S LT26i cujo sistema operacional é o Android 4.0.4. Segundo Sony (2013), este dispositivo possui um processador Qualcomm Dual Core de 1.5 *Giga Hertz* (GHz), memória de 1 *Giga Byte* (GB) e resolução de 1280x720.

Para execução do projeto web foi utilizado o servidor de aplicação JBoss 7.1 com abertura para a web. Para abstração do endereço Internet *Protocol* (IP) do servidor web foi utilizado o programa No-IP. As informações do sistema são persistidos no banco de dados PostgreSQL 9, e a comunicação do sistema web com este banco de dados foi implementado utilizando a biblioteca Apache DBUtils, que por sua vez é uma camada de abstração da interface *Java DataBase Connectivity* (JDBC).

3.3.2 Implementação do `AsyncTaskExecutor`

Como tratado anteriormente, a classe `AsyncTaskExecutor` foi criada com o intuito de adicionar mais uma camada na execução de processos concorrentes onde o método `execute()`, que pode ser visualizado no Quadro 2, se encarrega de executar o trecho de código em paralelo e mostrar o diálogo informando o usuário da execução do processo concorrente.

Este método recebe quatro parâmetros. O parâmetro `activity` é utilizado para que o processo volte a executar na `Thread` principal quando o código paralelo tenha terminado. O parâmetro `listener` é o objeto que irá receber o resultado do processamento paralelo. Já o parâmetro `requestCode` é utilizado somente para controle do `listener` pois ele pode acionar vários processamentos paralelos de uma só vez com valores diferentes para este parâmetro. Desta forma, o `listener` pode receber vários resultados simultaneamente, mas com o `requestCode` sendo devolvido é possível saber de qual processamento está recebendo o resultado. Por último, o parâmetro `asyncTask` contém o trecho de código a ser executado concorrentemente.

Quadro 2 – Código do método `AsyncTaskExecutor.execute()`

```
/**
* Método responsável por executar uma AsyncTask e mostrar um
* ProgressDialog. O retorno da AsyncTask será devolvido ao
* listener informado.
* @param activity
* @param listener
* @param requestCode
* @param asyncTask
*/
public static void execute(
      final Activity activity
      , final AsyncTaskListener listener
      , final int requestCode
      , final AsyncTask<String, Void, AsyncTaskResult> asyncTask) {

  //mostra o dialog utilizando a Thread principal
  final ProgressDialog dialog = ProgressDialog.show(activity, "Tp²",
                                          "Executando..", false, false);

  dialog.setIcon(R.drawable.ic_launcher);

  new Thread(new Runnable() {

    @Override
    public void run() {

      try {

        //executa a AsyncTask e espera o resultado ('travando' a
        //Thread atual: por isso é criada uma nova Thread para a
        //execução da AsyncTask)
        final AsyncTaskResult result = asyncTask.execute("").get();

        //fecha o dialog
        dialog.dismiss();

        //devolve o resultado executando na Thread principal
        activity.runOnUiThread(new Runnable() {

          @Override
          public void run() {
            listener.onAsyncTaskResult(requestCode, result);
          }

        });
      } catch (Exception e) {
        e.printStackTrace();
      }

    }
  }).start();
}
```

3.3.3 Obtendo a localização periodicamente

Para obter a atualização da localização do dispositivo periodicamente, ou seja, toda vez que a pessoa se movimentar uma certa distância, foi implementado o método `LocationUtil.getLocationPeriodically()` que verifica se o dispositivo possui o GPS ativado: caso sim, o utiliza como fornecedor; caso não, utiliza a rede de dados como fornecedor. Este trecho de código pode ser visualizado no Quadro 3.

Quadro 3 – Fragmento do método `LocationUtil.getLocationPeriodically()`

```
...

if(locationManager == null) {
  locationManager = (LocationManager) activity
            .getSystemService(Context.LOCATION_SERVICE);
}

try{gpsEnabled=locationManager.isProviderEnabled(
        LocationManager.GPS_PROVIDER);}catch(Exception ex){}
try{networkEnabled=locationManager.isProviderEnabled(
        LocationManager.NETWORK_PROVIDER);}catch(Exception ex){}

//retorna false caso nao tenha fornecedor disponível
if(!gpsEnabled && !networkEnabled) {
  return false;
}

//se o GPS está ativo utilizará somente o GPS para atualizar
if(gpsEnabled) {
  locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER
            , 0, minDistance, locationListenerGps);
} else {
  locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER
            , 0, minDistance, locationListenerNetwork);
}

return true;
```

Toda vez que o fornecedor enviar uma atualização da posição, o observador da classe `LocationUtil` - identificado no atributo `locationResult` - será notificado através do método `gotLocation()` na `Thread` principal (assim como no `AsyncTaskExecutor`). As atualizações serão realizadas até que o método `removeUpdates()` seja acionado, pois este informa os fornecedores para não notificarem mais as mudanças de localização. Estes fragmentos de código podem ser visualizados no Quadro 4.

Quadro 4 – Fragmentos de código da classe `LocationUtil`

```
private LocationResultListener locationResult;

...

private LocationListener locationListenerGps = new LocationListener() {
    public void onLocationChanged(Location location) {
      if(minAccuracy == null || location.getAccuracy() <= minAccuracy) {
      LocationUtil.this.gotLocation(location, requestCode);
      }
    }
    ...
};

private LocationListener locationListenerNetwork = new LocationListener()
{
    public void onLocationChanged(Location location) {
      if(!gpsEnabled) {
        LocationUtil.this.gotLocation(location, requestCode);
      }
    }
    ...
};

private void gotLocation(final Location location, final Integer
                                                    requestCode) {
    ...

    //devolve o resultado executando na Thread principal
    activity.runOnUiThread(new Runnable() {

      @Override
      public void run() {
        locationResult.gotLocation(location, requestCode);
      }

  });
}

public void removeUpdates() {
  if(locationManager != null) {
    locationManager.removeUpdates(locationListenerGps);
    locationManager.removeUpdates(locationListenerNetwork);
  }
}
```

3.3.4 Implementação da RA

A implementação da RA foi realizada na atividade `RAActivity` e na tela `RAView`.

3.3.4.1 Atividade `RAActivity`

No evento `onCreate()` foram inicializados e configurados os componentes necessários e foi realizada a sobreposição dos gráficos da RA na imagem em tempo real da câmera do dispositivo - através das visões `CameraView` e `RAView`. A inicialização da `RAActivity` pode ser visualizada no Quadro 5.

Quadro 5 – Configuração da `RAActivity`

```
private RAView raView;
private CameraView cameraView;
private LocationUtil locationUtil;

private int distanciaMaxima;
private List<Integer> listaCategorias;
private List<Long> listaAmigos;
private int nota;

//utilizado para deixar a tela ligada durante a execução da activity
private PowerManager.WakeLock wakeLock;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  //define a orientação da tela como Portrait (sempre)
  setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
  //recebe os locais encontrados e os parâmetros utilizados na procura
  List<Local> locais = (List<Local>)getIntent()
            .getSerializableExtra(PARAM_LISTA_LOCAIS);
  Location locationAtual = (Location)getIntent()
            .getParcelableExtra(PARAM_LOCATION_ATUAL);
  this.distanciaMaxima = FormatUtil.getNumber(
            Integer.class,
            getIntent().getFloatExtra(PARAM_DISTANCIA_MAXIMA, 0f));
  this.listaCategorias = (List<Integer>)getIntent()
            .getSerializableExtra(PARAM_LISTA_CATEGORIAS);
  this.listaAmigos = (List<Long>)getIntent()
            .getSerializableExtra(PARAM_LISTA_AMIGOS);
  this.nota = getIntent().getIntExtra(PARAM_NOTA, 0);

  //inicializa as view da realidade aumentada e da câmera
  this.raView = new RAView(this, locais, locationAtual
            , getIntent().getFloatExtra(PARAM_DISTANCIA_MAXIMA, 0f));
  this.cameraView = new CameraView(this);

  //cria um layout, adicionando as views da Camera e da Realidade
  //Aumentada - para que os gráficos da RA sobreponham a imagem da câmera
  final FrameLayout frame = new FrameLayout(this);
  frame.addView(cameraView);
  frame.addView(raView);

  //define o conteúdo da Activity como o Layout criado anteriormente
  setContentView(frame);

  //obtém a instância do wakeLock
  final PowerManager pm =
            (PowerManager)getSystemService(Context.POWER_SERVICE);
  this.wakeLock = pm.newWakeLock(
            PowerManager.SCREEN_DIM_WAKE_LOCK, this.getClass().getName());

  //cria a instâcia do LocationUtil
  this.locationUtil = new LocationUtil();
}
```

Após a configuração dos componentes, foram utilizados os eventos `onResume()` e `onPause()` (Quadro 6) para iniciar e parar a execução dos componentes, respectivamente. Foram utilizados estes eventos de forma com que caso o usuário navegue, por exemplo, para

visualizar os detalhes de um local a `RAActivity` tenha o evento `onPause()` acionado e então deixe de escutar as atualizações na localização e também revogar o pedido para ficar com a tela ligada, economizando bateria no uso de luz e GPS/rede. Assim que o usuário voltar da tela dos detalhes do local para a tela de RA, o método `onResume()` será acionado e os componentes de luz e obtenção de localização voltarão a ser utilizados.

Quadro 6 – Eventos `onResume()` e `onPause()` da `RAActivity`

```
@Override
protected void onResume() {
  super.onResume();

  //aciona a view da RA
  raView.onResume();

  //adquiri a permissão para deixar a tela ativada
  wakeLock.acquire();

  //começa a escutar as atualizações de local
  locationUtil.getLocationPeriodically(this, this, 5f, 5f);
}

@Override
protected void onPause() {
  super.onPause();

  //para as views da camera e da RA
  raView.onPause();
  cameraView.onPause();

  //revoga a permissão para deixar a tela ativada
  this.wakeLock.release();

  //para de escutar as atualizações do local
  locationUtil.removeUpdates();
}
```

A partir do momento em que o evento `onResume()` é executado, a `RAActivity` começa a escutar as atualizações de localização a cada 5 metros, no mínimo. As atualizações são recebidas no método `gotLocation()` - definido na interface `LocationResultListener` - que com a nova posição irá executar a busca dos locais atualizados em *background* utilizando o `AsyncTaskExecutor`. A implementação do método `gotLocation()` pode ser visualizado no Quadro 7.

Quadro 7 – Método `gotLocation()` da `RAActivity`

```
@Override
public void gotLocation(Location location, Integer requestCode) {
  if(location == null) {
    return;
  }

  //converte a posição obtida para o modelo PosicaoGeografica
  final PosicaoGeografica posicaoGeografica =
       new PosicaoGeografica(Double.toString(location.getLatitude()),
                             Double.toString(location.getLongitude()));

  //executa o código de forma concorrente, porém sem mostrar o progress
  //dialog
  AsyncTaskExecutor.executeWithoutProgressDialog(
    this, this
    , REQUEST_CODE_AT_BUSCAR_LOCAIS

    //cria o AsyncTask que contém o código concorrente no método
    //doInBackGround
    , new AsyncTask<String, Void, AsyncTaskResult>() {

      @Override
      protected AsyncTaskResult<List<Local>> doInBackground(String...
                                                              params) {
        try {

          //obtem a instância do web service ILocalWS
          final ILocalWS localWS =
               WebServiceUtil.createWebServiceInstance(ILocalWS.class);

          //executa a busca dos locais
          List<Local> locais = localWS.selecionarLocais(
              posicaoGeografica
              , distanciaMaxima
              , nota
              , listaCategorias
              , listaAmigos
          );

          //retorna a lista de locais como resultado
          return new AsyncTaskResult<List<Local>>(locais);

        } catch(Exception ex) {
          //retorna a exceção em caso de erro
          return new AsyncTaskResult<List<Local>>(ex);
        }
      }
    }
  );
}
```

Assim que os locais forem obtidos, o resultado será enviado ao método `onAsyncTaskResult()` - definido na interface `AsyncTaskListener` - que então acionará o método `resultadoATBuscarLocais()` o qual enviará a lista atualizada dos locais para a `RAView`, responsável por renderizar os gráficos da RA. A implementação dos métodos `onAsyncTaskResult()` e `resultadoATBuscarLocais()` pode ser visualizada no Quadro 8.

Quadro 8 – Métodos `onAsyncTaskResult()` e `resultadoATBuscarLocais()` da `RAActivity`

```
@Override
public void onAsyncTaskResult(int requestCode, AsyncTaskResult resultado)
{
  switch (requestCode) {
    //verifica se este requestCode é o da execução da busca de locais
    case REQUEST_CODE_AT_BUSCAR_LOCAIS:
      //chama o método responsável por tratar o resultado
      resultadoATBuscarLocais(resultado);
      break;
  }
}

private void resultadoATBuscarLocais(AsyncTaskResult<List<Local>>
                                                          resultado) {
  try {
    //verifica se houve algum problema na busca dos locais
    resultado.validarExcecao();
    //atualiza a lista de locais na RAView
    raView.setLocais(resultado.getResultado());

  } catch (Exception e) {
    //se ocorreu exceção, simplesmente não atualiza os locais
    e.printStackTrace();
  }
}
```

3.3.4.2 Visão `RAView`

A classe `RAView` tem como objetivo renderizar os locais fornecidos pela `RAActivity`, utilizando OpenGL ES, de acordo com as suas distâncias para a localização do usuário assim como a angulação quando comparado com a bússola do dispositivo.

A distância do local para a localização atual é retornada pelo *web service* `LocalWS.selecionarLocais()`. Já a captura dos valores da bússola do dispositivo utilizam a estratégia dos eventos `onResume()` e `onPause()`, onde no primeiro inicia-se obtenção dos valores e no segundo ocorre a pausa na obtenção do mesmo. Quando há uma atualização no valor da bússola é chamado o evento `onSensorChanged()`. Este método leva em consideração a declinação magnética obtida no construtor da `RAView` de acordo com a posição geográfica do dispositivo. Estes fragmentos de código podem ser visualizados no Quadro 9.

Quadro 9 – Fragmentos para obtenção do valor da bússola na `RAView`

```
public RAView(Activity activity, List<Local> locais
 , Location   locationAtual, float distanciaMaxima){
  super(activity);

  ...

  //obtém o valor da declinação magnética de acordo com a posição
  this.declination = new GeomagneticField(
                       (float)locationAtual.getLatitude()
                     , (float)locationAtual.getLongitude()
                     , (float)locationAtual.getAltitude()
                     , locationAtual.getTime()
                   ).getDeclination();

  ...

}

@Override
public void onSensorChanged(SensorEvent event) {
  //se for uma atualização do sensor de orientação
  if(event.sensor.getType() == Sensor.TYPE_ORIENTATION) {
    //atualiza o angulo da bússola levando em consideração a declinação
    bussola = event.values[0] + declination;

    //ajusta o valor da bússola para não ficar com angulação negativa
    if(bussola < 0) {
      bussola += 360;
    }

    //requere a renderização da view
    requestRender();
  }
}
```

O renderizador do OpenGL é obtido através do método `getOpenGLRenderer()`, que retorna uma instância da classe `GLSurfaceView.Renderer`. A lógica da renderização dos gráficos se encontra no método `onDrawFrame(GL10 gl)`, sendo que o parâmetro `gl` é utilizado para manipular as matrizes e também é ele que literalmente renderiza os objetos. O método `onDrawFrame()`, que pode ser visualizado no Quadro 10, define o tamanho e a posição do local na tela no eixo Y de acordo com a distância dos mesmos para a localização do dispositivo, como também define a posição na tela no eixo X de acordo com a diferença de ângulo do local para a direção onde a bússola do dispositivo está apontando. Além disso, este método também é responsável por chamar a renderização do radar informando os locais a serem mostrados juntamente com os seus ângulos em relação à bússola do dispositivo. O radar é posicionado sempre no canto superior direito da tela.

Quadro 10 – Método `onDrawFrame()` do `Renderer` utilizado na `RAView`

```
public void onDrawFrame(GL10 gl) {
   gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
   gl.glMatrixMode(GL10.GL_MODELVIEW);
   //limpa a lista de locais desenhados
   locaisRA.clear();

   //calcula a diferença do ângulo em relacao ao norte
   double diferencaBussola = LocationUtil.calcularDiferencaAngulos(bussola, 360);

   //cria uma lista que será utilizada na construção do radar
   List<Local> locaisRadar = new ArrayList<Local>();

   //obtém o lock para que a lista não possa ser modificada enquanto
   //está sendo utilizada para a renderização
   lockLocais.readLock().lock();

   for(Local local : locais) {
      //calcula a diferença do ângulo da bussola para o local
     double diferencaAngulo = LocationUtil
               .calcularDiferencaAngulos(bussola, local.anguloGraus);

     //cria a instância do local que será utilizado para a construção do radar
     Local localRadar = new Local();
     localRadar.distanciaMetros = local.distanciaMetros;
     //define o angulo do local em relação à direção apontada pela bússola
     localRadar.anguloGraus = LocationUtil
               .somarAngulos(local.anguloGraus, diferencaBussola);
     //adiciona na lista de locais que será utilizada no radar
     locaisRadar.add(localRadar);

     //somente desenha o local na tela caso esteja 45° a esquerda ou
     //a direita em relação à direção apontada pela bússola
     if((diferencaAngulo >= 0 && diferencaAngulo <= 45)
               || (diferencaAngulo < 0 && diferencaAngulo >= -45)) {
       //define a escala do local com base na distância:
       //quanto mais longe, menor será o local renderizado
       float escala = 1.0f - (FormatUtil.getNumber(Float.class,
                 String.format("%.1f",
                           local.distanciaMetros / (distanciaMaxima*2))));
       //define a posição X do local de acordo com a diferença do ângulo
       float x = (float)(diferencaAngulo*10);

       //define a posição no eixo Y de acordo com a distancia do local
       float escalaY = 1.0f - (FormatUtil.getNumber(Float.class,
                 String.format("%.1f", local.distanciaMetros / distanciaMaxima)));
       float y = (escalaY > 0.5
                     ? escalaY * (mundoYMin)
                     : 1-escalaY * (mundoYMax)) + deltaOrigemY/5;

       //adiciona o local na lista de locais renderizados
       LocalRA localRA =  new LocalRA(local, x, y, escala);
       locaisRA.add(localRA);

       //renderiza o local
       localRA.draw(gl, activity, glTextTitulo, glTextOutros);
     }
   }

   //libera o lock
   lockLocais.readLock().unlock();

   gl.glPushMatrix();
   gl.glLoadIdentity();
   //coloca o radar no canto superior direito
   gl.glTranslatef(mundoXMax-Radar.RAIO_RADAR-5, mundoYMax-Radar.RAIO_RADAR-5, 0);
   //renderiza o radar, passando a lista de locais com
   //a diferença de ângulo com base na bussola do dispositivo
   radar.draw(gl, locaisRadar);
   gl.glPopMatrix();
}
```

O tratamento de seleção dos locais para abrir os seus detalhes é realizado no método `onTouchEvent()` da `RAView`. Como cada local é representado como um retângulo na visualização da RA, para saber se o usuário selecionou um local é verificado, para cada local renderizado, se o ponto tocado está dentro da *bounding box* do retângulo. O ponto tocado é convertido para as coordenadas da tela utilizando o cálculo *Normalized Device Coordinates* (NDC). A implementação do evento `onTouchEvent()` pode ser visualizado no Quadro 11.

Quadro 11 – Métodos `onTouchEvent()` da `RAActivity`

```
@Override
public boolean onTouchEvent(MotionEvent event) {
  if(iniciandoActivityDetalhesLocal) {
    return false;
  }

  //calcula o valor do ponteiro para o eixo X utilizando o cálculo NDC
  float x = ((event.getX() - 0)
              * (deltaDestinoX/deltaOrigemX)) + mundoXMin;

  //calcula o valor do ponteiro para o eixo Y utilizando o cálculo NDC
  float y = ((event.getY() - 0)
              * (deltaDestinoY/deltaOrigemY)) + mundoYMax;

  //varre a coleção de trás para frente para que valide primeiramente os
  //locais mais a frente
  for(int i = locaisRA.size()-1; i >= 0; i--) {

    LocalRA localRA = locaisRA.get(i);

    if(localRA.posicaoDentroBoundingBox(x, y)) {
      iniciandoActivityDetalhesLocal = true;
      Intent intent = new Intent(activity,VisualizarLocalActivity.class);
      intent.putExtra(TCCActivity.PARAM_ID_LOCAL, localRA.local.id);
      activity.startActivityForResult(intent,
           ProcurarLocaisActivity.REQUEST_CODE_VISUALIZAR_LOCAL);
      break;
    }

  }

  return true;
}
```

3.3.5 Geração e consumo dos *web services* REST

A geração dos *web services* foi implementada utilizando a biblioteca RestEasy que, através de anotações em métodos e atributos, publica serviços os quais consomem e retornam dados no formato JSON. Estes serviços ficam acessíveis através de requisições HTTP. Um exemplo de definição de *web service* consumível via requisição *post* pode ser visualizado no Quadro 12.

Quadro 12 – Exemplo de geração de *web services* utilizando anotações

```
@Path("/ws/Endereco")
@Consumes({ MediaType.APPLICATION_JSON + ";charset=UTF-8" })
@Produces({ MediaType.APPLICATION_JSON + ";charset=UTF-8" })
public interface IEnderecoWS {

  @POST
  @Path("/getEndereco")
  public Endereco getEndereco(PosicaoGeografica posicaoGeografica);

  ...

}

public class PosicaoGeografica implements Serializable  {

  @FormParam("posicaoLatitude")
  public String latitude;

  @FormParam("posicaoLongitude")
  public String longitude;

  ...

}
```

O consumo destes *web services* pela aplicação Android foi abstraída a partir do uso da biblioteca RestEasy Client Mobile, que disponibiliza a classe `ProxyFactory` para obter-se instâncias dos serviços. Esta classe foi utilizada pela `WebServiceUtil` (Quadro 13) que centraliza a obtenção das instâncias devido à quantidade de parâmetros passados ao método `ProxyFactory.create()`.

Quadro 13 – Classe `WebServiceUtil`

```
public abstract class WebServiceUtil {

  private static ResteasyProviderFactory pf;
  private static BasicHttpParams params;

  static {
    pf = ResteasyProviderFactory.getInstance();
    pf.addClientErrorInterceptor(new WebServiceExceptionInterceptor());
    params = new BasicHttpParams();
    HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
    HttpProtocolParams.setContentCharset(params,
                                  HTTP.DEFAULT_CONTENT_CHARSET);
    HttpProtocolParams.setUseExpectContinue(params, false);
  }

  public static <T> T createWebServiceInstance(Class<T> classe) {
    return ProxyFactory.create(
        classe
        ,ProxyFactory.createUri(ProjetoConstants.URL_PROJETO_WEB)
        ,new ApacheHttpClient4Executor(params)
        ,pf
      );
  }

}
```

A execução dos *web services* na aplicação Android é sempre realizada através de processos concorrentes que usualmente abrem um diálogo ao usuário informando a execução de tarefas que podem demorar. No Quadro 14 pode-se visualizar a chamada ao *web service* `getEndereco()` dentro de uma `Activity` da aplicação.

Quadro 14 – Exemplo de chamada paralela a um *web service*

```
AsyncTaskExecutor.execute(this, this, REQUEST_CODE_AT_LOCAL_NASCIMENTO,
  new AsyncTask<String, Void, AsyncTaskResult>() {

    @Override
    protected AsyncTaskResult<Endereco> doInBackground(String... params)
    {
      try {

        IEnderecoWS enderecoWS =
           WebServiceUtil.createWebServiceInstance(IEnderecoWS.class);

        Endereco endereco = enderecoWS.getEndereco(posicaoGeografica);

        return new AsyncTaskResult<Endereco>(endereco);

      } catch(Exception ex) {
        return new AsyncTaskResult<Endereco>(ex);
      }
    }
  }
);
```

3.3.6 Operacionalidade da implementação

A operacionalidade da aplicação é exposta através das funcionalidades da aplicação Android, onde são apresentadas imagens da execução do mesmo no dispositivo Sony Xperia S LT26i, utilizado para os testes durante o desenvolvimento do sistema. O fluxo da apresentação leva em consideração que é a primeira vez que o usuário está utilizando a aplicação e, portanto, não possui cadastro no sistema.

Ao acessar a aplicação, é apresentada a tela de login (Figura 37) ao usuário com a opção `Quero me cadastrar`.

Figura 37 – Tela de login

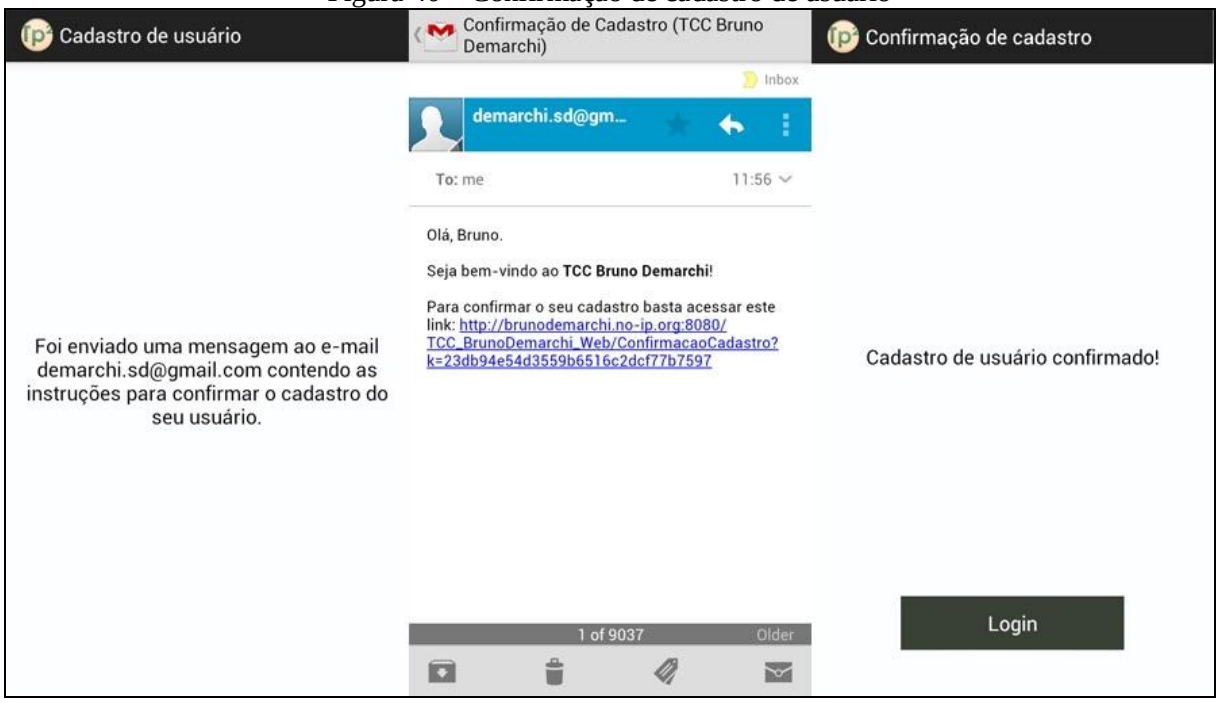
O usuário acessa esta opção e então á apresentada a tela com os formulários para o cadastro de um novo usuário, visíveis na Figura 38. Na aba `Local de Nascimento`, ao pressionar o botão `Preencher conforme GPS`, o sistema preenche os dados do local de nascimento de acordo com a localização do usuário. Já na aba `Endereço Residencial`, ao pressionar-se o botão `Selecionar no Google Maps`, o sistema abre o Google Maps para que o usuário selecione uma posição no mapa. Desta forma o sistema preenche os dados do endereço residencial conforme a posição selecionada, o que pode ser visualizado na Figura 39.

Figura 38 – Telas de cadastro de usuário

Figura 39 – Seleção de endereço residencial na tela de cadastro de usuário

Preenchidos os dados obrigatórios, o usuário pressiona o botão `Cadastrar usuário` que irá realizar o cadastro no sistema e enviar uma mensagem de confirmação ao e-mail informado no cadastro. O usuário acessa o seu e-mail e clica no *link* contido na mensagem, finalizando então o cadastro do usuário. A imagem destes passos pode ser visualizado na Figura 40.

Figura 40 – Confirmação de cadastro de usuário



O usuário então pode realizar a autenticação na aplicação informando seu usuário e senha, sendo redirecionado para o menu da aplicação (Figura 41).

Figura 41 – Menu

Durante a execução de tarefas que podem ser longas, como por exemplo as chamadas ao módulo de serviços, é apresentada ao usuário uma mensagem de progresso de execução da tarefa. Essa mensagem pode ser visualizada na Figura 42.

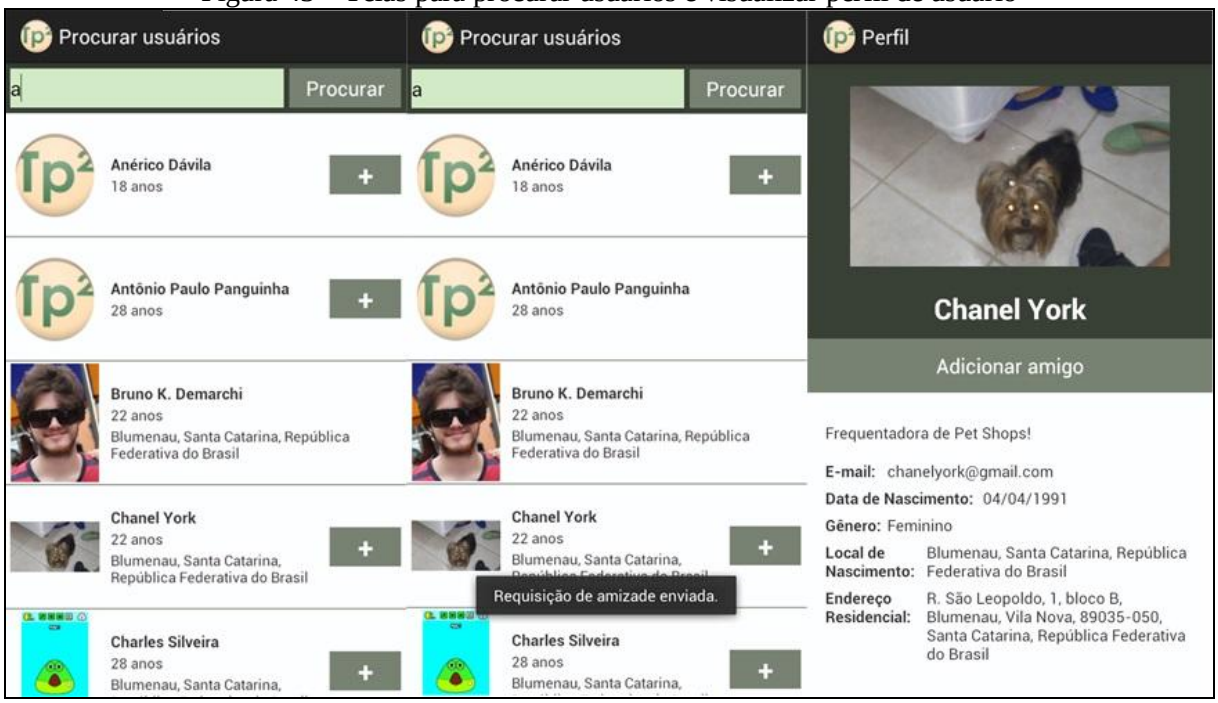
Figura 42 – Mensagem de progresso

A seguir são apresentadas as funcionalidades `Procurar usuários`, `Cadastrar local` e `Procurar locais`.

3.3.6.1 Procurar usuários

Ao acessar esta funcionalidade, é apresentado ao usuário um campo para informar o nome a ser pesquisado e um botão para executar a ação. A pesquisa é realizada por paginação, ou seja, os usuários são procurados de cinco em cinco conforme o usuário for avançando ao final da lista. As fotos dos usuários são carregadas de forma assíncrona, não interrompendo a navegação. Os perfis encontrados que ainda não fazem parte da lista de amigos do usuário apresentam um botão + posicionado à direita que ao ser pressionado envia uma requisição de amizade. Ao selecionar um usuário na lista, o sistema é redirecionado para o perfil do mesmo. Estas telas podem ser visualizadas na Figura 43.

Figura 43 – Telas para procurar usuários e visualizar perfil de usuário



3.3.6.2 Cadastrar local

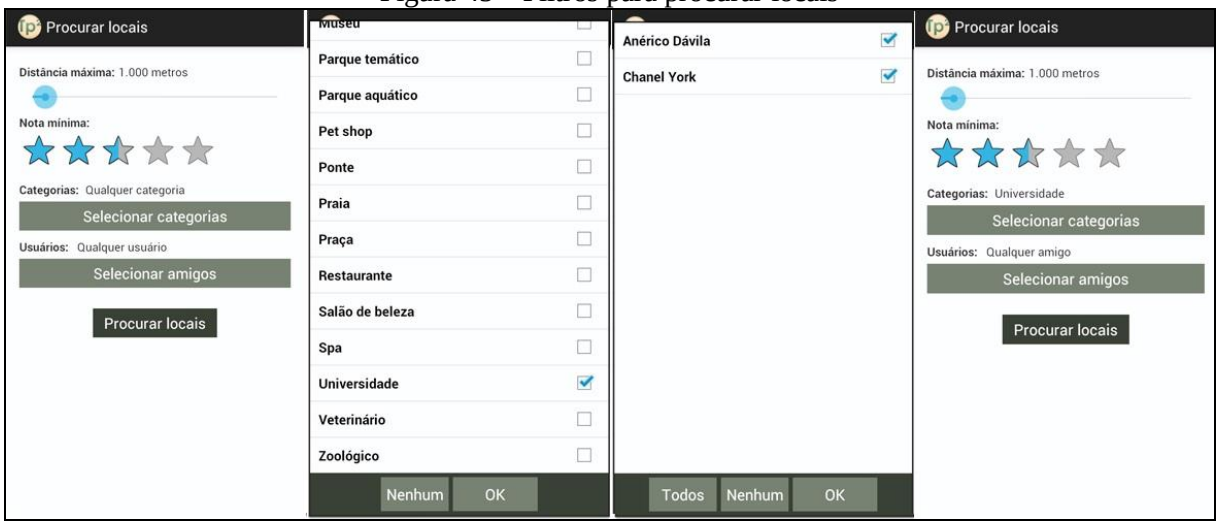
Ao acessar esta funcionalidade, é apresentado ao usuário o mapa do Google Maps com um círculo azul no raio de 1.000 metros cujo centro é a posição atual do dispositivo. O usuário deve selecionar uma posição no mapa dentro deste círculo. Então o sistema apresentará os campos para preenchimento de informações sobre o local e também a recomendação do usuário sobre aquele local. Ao pressionar o botão Cadastrar Local, o sistema realiza as validações das informações. Caso todas as informações estejam corretas, o cadastro do local é finalizado e a aplicação volta ao menu principal apresentando uma mensagem de sucesso. Estes cinco passos podem ser visualizados na Figura 44.

Figura 44 – Cadastro de local em cinco passos

3.3.6.3 Procurar locais

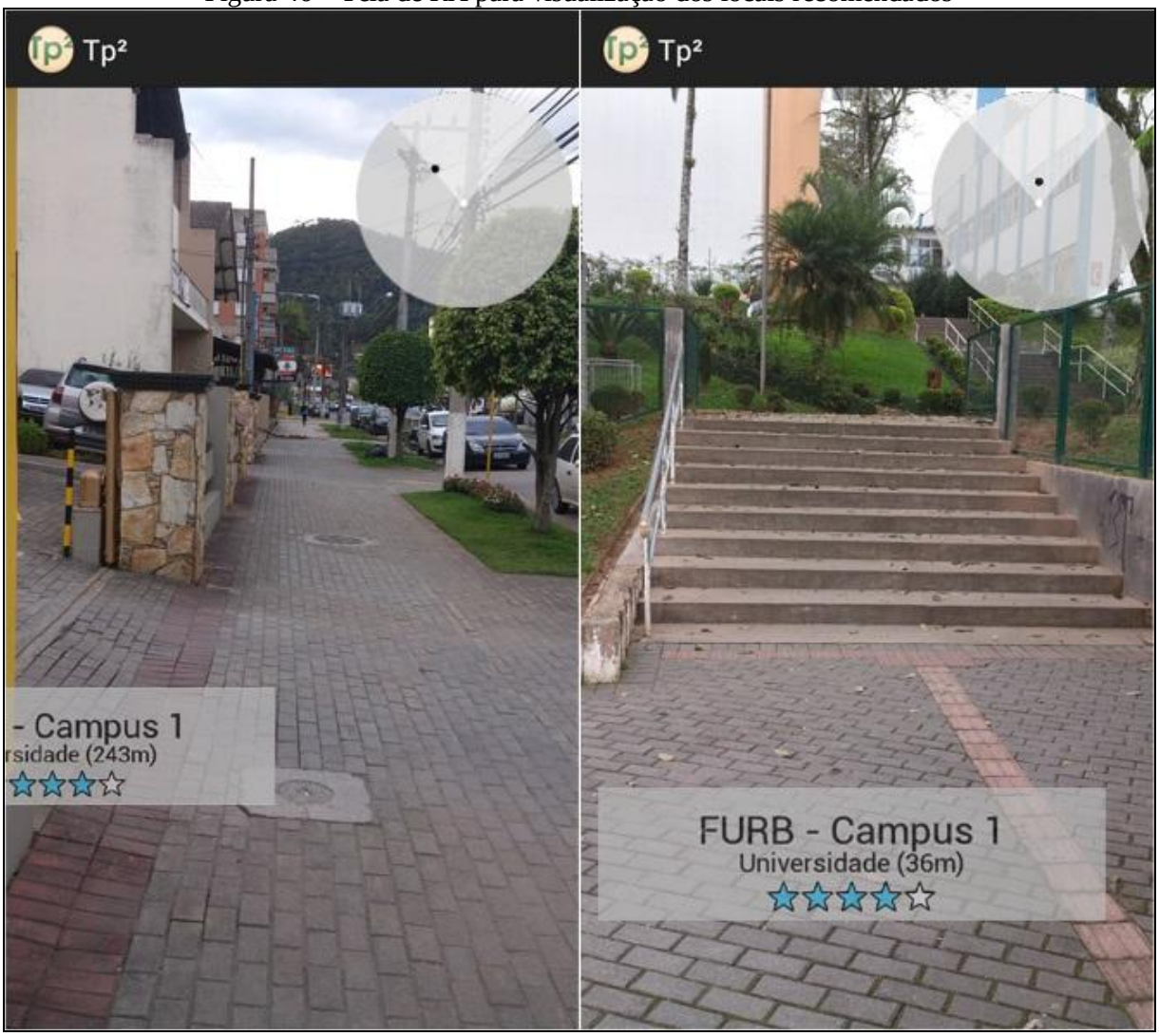
Esta funcionalidade, ao ser acessada, apresenta um formulário com os filtros para que o usuário possa procurar os locais que deseja: distância máxima, média de nota mínima das recomendações dos outros usuários, categorias e usuários. As telas de filtro podem ser visualizadas na Figura 45.

Figura 45 – Filtros para procurar locais



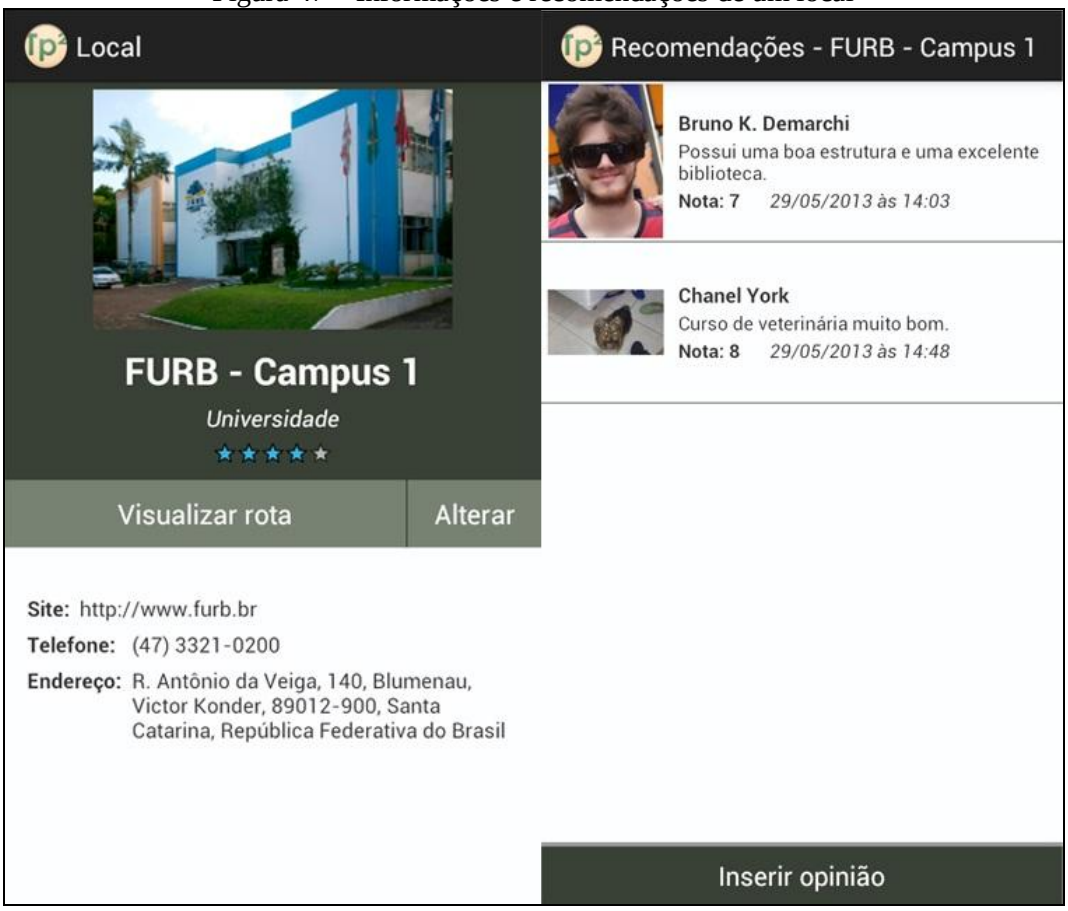
Ao pressionar o botão `Procurar locais` na tela dos filtros, é realizada a busca dos locais seguindo os filtros informados. Os locais encontrados são apresentados na tela de RA, que pode ser visualizada na Figura 46.

Figura 46 – Tela de RA para visualização dos locais recomendados



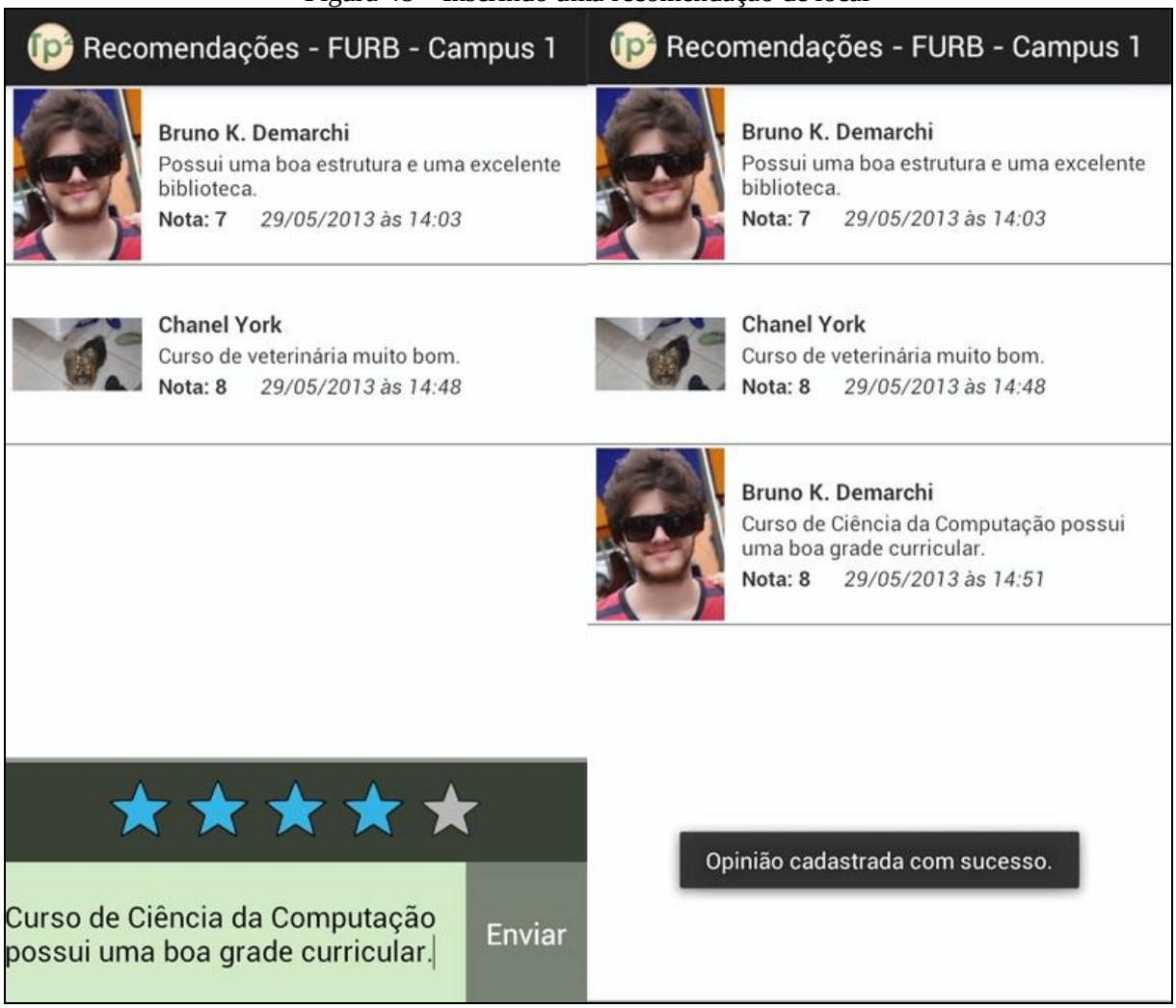
Ao selecionar um local na tela de RA, o sistema abre a tela com informações do local e também as recomendações realizadas sobre ele. A imagem do local é carregada assincronamente. A recomendação do criador do local é apresentada no topo da lista, seguida pelas recomendações dos amigos e posteriormente as recomendações dos demais usuários. As recomendações são carregadas utilizando paginação de cinco em cinco. As imagens dos usuários na lista de recomendações são carregadas assincronamente. A Figura 47 apresenta a tela com as informações do local e suas recomendações.

Figura 47 – Informações e recomendações de um local



Ao pressionar o botão `Inserir opinião` na tela anterior, é verificado se a posição atual do dispositivo se encontra a no máximo 1.000 metros da posição do local em questão. Caso sim, o usuário pode adicionar uma recomendação sobre este local, informando uma opinião e uma nota. A tela para recomendação do local pode ser visualizada na Figura 48.

Figura 48 – Inserindo uma recomendação de local



3.4 RESULTADOS E DISCUSSÃO

O presente trabalho apresentou o desenvolvimento de uma rede social para recomendações de locais em Android utilizando RA. A aplicação desenvolvida permite que seus usuários cadastrem seus perfis, mantenham uma lista de amigos, cadastrem locais baseados em sua posição geográfica e realizem recomendações sobre os locais cadastrados. Além disso, a aplicação permite realizar a busca de locais de acordo com os filtros informados, onde os locais encontrados são apresentados através de RA.

Todos os objetivos e requisitos definidos na proposta inicial foram alcançados e seguidos, exceto um RNF que foi alterado. Na proposta inicial deste trabalho foi definido um RNF onde a aplicação deveria ser compatível com o sistema operacional Android na versão `2.2`. Este RNF foi alterado para que a versão miníma compatível fosse a `4.0.4` devido a disponibilidade de dispositivos para testes.

Para o desenvolvimento deste trabalho foram explorados alguns trabalhos correlatos que possuem características semelhantes ao sistema criado. O Quadro 15 apresenta uma

comparação de funcionalidades e características envolvendo a aplicação desenvolvida e os trabalhos correlatos.

Quadro 15 – Comparação com trabalhos correlatos

Características / Funcionalidades	Aplicação desenvolvida	Foursquare	What is Up App	TripAdvisor Augmented Reality	TCC Rampelotti
desenvolvimento finalizado	sim	sim	não	sim	sim
recomendação de locais	sim	sim	sim	sim	não
disponível para Android	sim	sim	não	não	sim
realidade aumentada	sim	não	sim	sim	sim

Para análise dos trabalhos correlatos foram utilizadas somente as características presentes no sistema desenvolvido. Dentre os sistemas citados, somente o What is Up App ainda não teve sua implementação concluída. Além disso, todas são ferramentas possuem como objetivo a recomendação de locais - exceto a aplicação de Rampelotti (2011), cujo objetivo é auxiliar o cotidiano de alunos e professores num ambiente universitário. Percebe-se também que a solução desenvolvida é a única com o objetivo de recomendação de locais que possui implementação de RA e está disponível para Android. As ferramentas Foursquare e TripAdvisor Augmented Reality possuem outros grandes diferenciais com um nível de detalhamento de informações maior por se tratarem de sistemas consolidados e que estão disponíveis há anos no mercado.

Este trabalho teve como desafio o desenvolvimento para Android e a implementação de RA utilizando OpenGL ES, além do uso da bibliotecas RestEasy e RestEasy Client Mobile para geração e consumo de serviços REST. Todas estas tecnologias tiveram de ser estudadas do início, visto que o autor do trabalho não possuia experiência com as mesmas.

Um problema surgiu logo no início do desenvolvimento do sistema, onde ocorria erro no momento da captura de uma exceção gerada por um serviço REST por parte da aplicação do dispositivo. Foi constatado que a biblioteca RestEasy Mobile Client não possuía as classes `ByteArrayProvider` e `ReadFromStream` por se tratar de uma versão reduzida especialmente feita para ser utilizada em dispositivos móveis. Além disso, a biblioteca possuía um *bug* na classe `BodyEntityExtractor` que havia sido corrigido para a versão normal do RestEasy. Para resolver estes problemas foi necessário adicionar manualmente as duas primeiras classes e substituir a terceira pela versão corrigida no arquivo `resteasy-mobile-1.0.0.jar`.

Outra dificuldade encontrada foi na utilização do OpenGL ES. Inicialmente a versão escolhida para implementação da RA foi a `2.0`, porém constatou-se que a produtividade com a mesma estava muito baixa devido ao seu uso ser mais complexo do que a versão `1.0`. Como o nível de detalhamento dos gráficos renderizados para a RA são baixos, optou-se por utilizar a versão `1.0`, o que trouxe mais produtividade no desenvolvimento da aplicação.

4 CONCLUSÕES

Este trabalho apresentou um aplicativo móvel para a plataforma Android que busca juntar a interatividade e utilidade de uma rede social específica para recomendações de locais com a naturalidade e facilidade de uso de um *software* de RA, utilizando diversos recursos dos dispositivos móveis, tais como câmera, bússola e GPS. A aplicação fornece uma forma de conhecer novos lugares baseado na opinião de outras pessoas que potencialmente fazem parte da rede social do usuário. Esta rede social é formada através de uma lista de amigos que pode ser mantida pelo usuário. Também é possível cadastrar novos lugares e recomendar positiva ou negativamente locais que já fazem parte do banco de dados do sistema.

O presente trabalho demonstra o desenvolvimento na plataforma Android utilizando diversos recursos disponíveis, como a API para acesso aos sensores dos dispositivos móveis e a biblioteca OpenGL ES para renderização de gráficos. No caso da biblioteca OpenGL ES, optou-se pela versão `1.0` pelo fato de a produtividade com a versão `2.0` não estar sendo alta e de que não há um grande nível de detalhamento dos gráficos renderizados na RA.

Esta aplicação foi desenvolvida buscando a melhor interação do usuário com a aplicação. Para tal foram implementadas integrações com os serviços do Google Maps para busca de endereços baseados em posições geográficas. Também houve o cuidado de apresentar uma mensagem de progresso durante e execução de tarefas que podem ser mais demoradas. Além disso, foram adotadas estratégias como paginação de itens em listas e carregamento assíncrono das imagens do sistema. Por último, foi implementada a tela de RA para visualização dos locais recomendados.

Neste trabalho adotou-se o uso de uma arquitetura cliente-servidor, sendo que no servidor foram disponibilizados serviços REST retornando dados em JSON. Este serviços são acessados através de requisições HTTP. Tal arquitetura mostrou-se bastante adequada ao sistema proposto visto que a quantidade de dados trafegados em serviços REST é bastante reduzida já que possui poucos dados de controle. Além disso, os serviços centralizam as regras de negócio do sistema e poderiam ser reaproveitados em aplicações para outras plataformas como iOS e Windows Phone, ou até mesmo em um *site* para acesso em computadores convencionais.

Por fim, a utilização das bibliotecas RestEasy para geração e consumo de serviços REST se mostrou bastante produtiva, permitindo o foco no desenvolvimento dos serviços e da aplicação Android.

4.1 EXTENSÕES

Como sugestões de extensões para a continuidade do presente trabalho, tem-se:

a) criar uma forma de denúncia de *spam*;
b) desenvolver um processo para confirmação dos donos dos locais: por exemplo, através do envio de uma chave de confirmação através de carta para o endereço do local;
c) criar um sistema para reportar defeitos em casos de erros, onde a aplicação registra um incidente automaticamente com informações como: modelo do dispositivo, versão do sistema operacional, tela em que ocorreu o erro, dados inseridos, etc;
d) criar uma página web para administração do sistema: para, por exemplo, analisar as denúncias de *spam*, administrar as confirmações de donos dos locais e verificar as reportagens de defeitos;
e) criar uma lista de locais favoritos do usuário;
f) criar uma rotina que executa todos os dias para limpar as confirmações de cadastro que não foram ativadas, por exemplo, em até 7 dias;
g) na tela onde é possível visualizar as recomendações de um local, caso tenha muitas recomendações, mostrar um sumário que pode ser detalhado;
h) utilizar serviços de *push notification* para a aplicação receber avisos mesmo quando a mesma não está ativa, com o intuito de economizar energia;
i) alterar a aplicação para utilizar o OpenGL ES na versão `2.0` ao invés da `1.0`.

REFERÊNCIAS BIBLIOGRÁFICAS

AGÊNCIA DDA. **Realidade Aumentada (RA)**. [São Paulo], [2013]. Disponível em: <http://www.agenciadda.com.br/realidade-aumentada-ra>. Acesso em: 20 maio 2013.

AGUIAR, Sonia. Redes sociais na internet: desafio à pesquisa. In: CONGRESSO BRASILEIRO DE CIÊNCIAS DA COMUNICAÇÃO, 30., 2007, Santos. **Anais eletrônicos...** Santos: Intercom, 2007, p. 2. Disponível em: <http://www.sitedaescola.com/downloads/portal_aluno/Maio/Redes%20sociais%20na%20internet-%20desafios%20%E0%20pesquisa.pdf>. Acesso em: 20 maio 2013.

ANDROID. **Activities.** [S.l.], 2013a. Disponível em: <http://developer.android.com/guide/components/activities.html>. Acesso em: 19 maio 2013.

_____. **AsyncTask.** [S.l.], 2013b. Disponível em: <http://developer.android.com/reference/android/os/AsyncTask.html>. Acesso em: 21 maio 2013.

_____. **Camera.** [S.l.], 2013c. Disponível em: <http://developer.android.com/guide/topics/media/camera.html>. Acesso em: 22 maio 2013.

_____. **Location and maps.** [S.l.], 2013d. Disponível em: <http://developer.android.com/guide/topics/location/index.html>. Acesso em: 21 maio 2013.

_____. **Location strategies.** [S.l.], 2013e. Disponível em: <http://developer.android.com/guide/topics/location/strategies.html>. Acesso em: 21 maio 2013.

_____. **OpenGL.** [S.l.], 2013f. Disponível em: <http://developer.android.com/guide/topics/graphics/opengl.html>. Acesso em: 23 maio 2013.

_____. **Sensors overview.** [S.l.], 2013g. Disponível em: <http://developer.android.com/guide/topics/sensors/sensors_overview.html>. Acesso em: 21 maio 2013.

_____. **The AndroidMainfest.xml file.** [S.l.], 2013h. Disponível em: <http://developer.android.com/guide/topics/manifest/manifest-intro.html>. Acesso em: 22 maio 2013.

AZUMA, Ronald T. A survey of augmented reality. **Presence**: Teleoperators And Virtual Environments, [S.l.], v. 6, n. 4, p. 355-385. Aug. 1997. Disponível em: <http://www.cs.unc.edu/~azuma/ARpresence.pdf>. 20 maio 2013.

BIMBER, Oliver; RASKAR, Ramesh. **Spatial augmented reality merging real and virtual worlds.** [S.l.]: A K Peters LTD, 2005. Disponível em: <http://www.uni-weimar.de/medien/ar/SpatialAR/download.php>. Acesso em: 20 maio 2013.

BLACKBERRY OS. **TripAdvisor -** Wikitude powered augmented reality App. [S.l.], 2013. Disponível em: <http://www.blackberryos.com/content/tripadvisor-wikitude-powered-augmented-reality-app-3882/>. Acesso em: 20 maio 2013.

BOYD, Danah M.; ELLISON, Nicole B. Social network sites: definition, history,and scholarship. **Journal of Computer-Mediated Communication**, [S.l.], 2008. Disponível em: <http://www.home-electronics.ch/HomeElectronic/Files/Web/Dossier/VL%203%20SNS%20History.pdf>. Acesso em 20 maio 2013.

COMSCORE. **A comScore lança o relatório 2013 *Brazil digital future in focus***. São Paulo, 2013. Disponível em: <http://www.comscore.com/por/Insights/Press_Releases/2013/3/comScore_Releases_2013_Brazil_Digital_Future_in_Focus_Report>. Acesso em: 20 maio 2013.

DALVIKVM. **Dalvik virtual machine**. [S.l.], 2008. Disponível em: <http://www.dalvikvm.com/>. Acesso em: 20 maio 2013.

DIGITAL BUZZ. **Top 10 augmented reality examples**. [S.l.], 2010. Disponível em: <http://www.digitalbuzzblog.com/top-10-augmented-reality-examples/>. Acesso em: 22 maio 2013.

EDGELIB. **OpenGL (ES) implementations**. [S.l.], [2013]. Disponível em: <http://www.edgelib.com/index.php?node=1093>. Acesso em: 23 maio 2013.

ELINUX. **Android architecture**. [S.l.], 2011. Disponível em: <http://elinux.org/Android_Architecture>. Acesso em: 19 maio 2013.

FACEBOOK. **Sobre**. [S.l.], 2013. Disponível em: <https://www.facebook.com/facebook/info>. Acesso em: 03 jun. 2013.

FOURSQUARE. **About**. [S.l.], 2013. Disponível em: <https://pt.foursquare.com/about>. Acesso em: 20 maio 2013.

FURB. **O ambiente**. [S.l.], 2013. Disponível em: <http://ava.furb.br/ava/ambiente/index.php>. Acesso em: 03 jun. 2013.

GOOGLE. **Google play**. [S.l.], 2013. Disponível em: <https://play.google.com>. Acesso em: 20 maio 2013.

KHRONOS. **OpenGL ES -** the standard for embedded accelerated 3D graphics. Beaverton, 2013. Disponível em: <http://www.khronos.org/opengles/>. Acesso em: 23 maio 2013.

LINKEDIN. **LinkedIn**. [S.l.], 2013. Disponível em: <http://www.linkedin.com/static?key=what_is_linkedin&trk=hb_what>. Acesso em: 03 jun. 2013.

OPEN HANDSET ALLIANCE. **Alliance.** [S.l.], [2013]. Disponível em: <http://www.openhandsetalliance.com/android_overview.html>. Acesso em: 20 maio 2013.

RAMPELOTTI, Ronaldo. **Ferramenta Android baseada em realidade aumentada e serviços baseados em localização usando notificações.** 2011. 70 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

REATEGUI, Eliseo B.; CAZELLA, Silvio C. Sistemas de recomendação. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 25., 2005, São Leopoldo. **Anais eletrônicos...** São Leopoldo: Unisinos, 2005. p. 2-12. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/enia/2005/0100.pdf >. Acesso em: 20 maio 2013.

POLITICABOOK. **PoliticaBook**. [S.l.], 2013. Disponível em: <http://www.politicabook.com/>. Acesso em: 03 jun. 2013.

SONY. **Xperia™ S.** [S.l.], 2013. Disponível em: <http://www.sonymobile.com/gb/products/phones/xperia-s/specifications/#black>. Acesso em: 27 maio 2013

VASSELAI, Gabriela T. **Um estudo sobre realidade aumentada para plataforma Android.** 2010. 103 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

WHAT IS UP APP. **What is up:** augmented reality iPhone App. [S.l.], [2013]. Disponível em: <http://whatisupapp.com/>. Acesso em: 20 maio 2013.

APÊNDICE A – Detalhamento dos casos de uso do sistema

A seguir podem ser visualizados os detalhes dos casos de uso do sistema.

O caso de uso `Login` é obrigatório, pois para utilização de qualquer funcionalidade faz-se necessário estar autenticado na aplicação. Esse caso de uso apresenta uma janela com campos para realizar a autenticação do usuário. Mais detalhes estão descritos no Quadro 16.

Quadro 16 – Caso de uso Login

UC01 – Login	
Descrição	Permite o usuário entrar na aplicação informando as suas credenciais.
Requisito atendido	RF01
Pré-condições	Nenhuma
Cenário principal	1. O sistema apresenta os campos e-mail e senha, além de dois botões: um para realizar a autenticação e outro para cadastrar um novo usuário; 2. Usuário informa e-mail e senha; 3. Ao pressionar o botão `Login` o sistema deve validar o e-mail e a senha informados; 4. O sistema verifica que o e-mail e a senha são válidos e apresenta o menu principal da aplicação.
Fluxo alternativo 01	No passo 2 o usuário pressiona o botão `Cadastrar` e o sistema é redirecionado para o `UC02 – Cadastrar usuário`.
Exceção 01	No passo 4, se ocorrer um erro ao validar o e-mail e a senha. O sistema deve mostrar uma mensagem de problema de autenticação, permanecendo na mesma tela.
Pós-condições	Acesso a tela de menu principal da aplicação.

Já no caso de uso `Cadastrar usuário` o usuário que não possui uma conta na apliação pode criá-la. Mais detalhes estão descritos no Quadro 17.

Quadro 17 – Caso de uso `Cadastrar usuário`

UC02 – Cadastrar usuário	
Descrição	Permite o usuário se cadastrar no sistema criando um perfil com seus dados pessoais.
Requisitos atendidos	RF01 e RF09
Pré-condições	Na tela de login o usuário deve ter pressionado o botão `Cadastrar`.
Cenário principal	1. O sistema apresenta os campos para inclusão de um novo usuário e um botão para cadastrar o usuário; 2. O usuário informa obrigatoriamente os campos nome, data de nascimento, senha e e-mail; 3. O usuário informa opcionalmente os demais campos; 4. O usuário pressiona o botão `Cadastrar Usuário`; 5. O sistema verifica que a idade do usuário é maior ou igual a 16 anos; 6. O sistema verifica que o e-mail informado ainda não é utilizado por outro usuário; 7. O sistema envia um e-mail para o endereço de e-mail cadastrado contendo um link para a confirmação do usuário; 8. O sistema apresenta uma mensagem indicando os passos para confirmar o cadastro do usuário; 9. O usuário acessa o link contido no e-mail; 10. É informada ao usuário a confirmação da criação do perfil.
Fluxo alternativo 01	No passo 2, o usuário pressiona o botão `Nova Foto` e o sistema é redirecionado para o `UC13 – Capturar imagem`. No retorno, a imagem capturada estará visível no campo foto.
Fluxo alternativo 02	No passo 2, o usuário pressiona o botão `Procurar Foto` e o sistema abre a galeria de fotos do dispositivo. O usuário seleciona uma imagem e a mesma estará visível no campo foto.
Fluxo alternativo 03	No passo 3, o usuário pressiona o botão `Selecionar Endereço` e o sistema abre o Google Maps na posição atual do GPS do dispositivo. O usuário então seleciona uma posição no mapa e o sistema preenche os campos do endereço residencial.
Fluxo alternativo 04	No passo 3, o usuário pressiona o botão `Preencher Endereço` e o preenche os campos do local de nascimento de acordo com a posição atual do GPS do dispositivo.
Exceção 01	No passo 5, o cliente informou uma data de nascimento que resulta em uma idade inferior a 16 anos. O sistema apresenta uma mensagem informando que a pessoa deve ter mais que 16 anos para criar um perfil na aplicação.
Exceção 02	No passo 6, o sistema verifica que o e-mail informado já foi utilizado por outro usuário. O sistema apresenta uma mensagem solicitando outro e-mail.
Pós-condições	O usuário cadastrado está disponível para ser utilizado.

No caso de uso `Editar perfil do usuário` o usuário pode alterar informações do seu perfil no sistema. Mais detalhes estão descritos no Quadro 18.

Quadro 18 – Caso de uso `Editar perfil do usuário`

UC03 – Editar perfil do usuário	
Descrição	Permite alterar os dados do usuário e as configurações de privacidade.
Requisitos atendidos	RF01 e RF09
Pré-condições	O usuário deve estar autenticado na aplicação e deve ter acessado a opção `Editar perfil` no menu principal.
Cenário principal	1. O sistema apresenta os campos do perfil do usuário para alteração, carregando os dados atuais; 2. O campo e-mail é apresentado somente para leitura do usuário, não sendo possível alterar o seu valor; 3. O sistema apresenta um botão para confirmar as alterações; 4. O usuário informa obrigatoriamente os campos nome e data de nascimento; 5. O usuário pressiona o botão `Confirmar alterações`; 6. O sistema verifica que a idade do usuário é maior ou igual a 16 anos; 7. O sistema apresenta uma mensagem confirmando as alterações do usuário.
Fluxo alternativo 01	No passo 4, o usuário pressiona o botão `Bater Foto` e o sistema é redirecionado para o `UC13 - Capturar imagem`. No retorno, a imagem capturada estará visível no campo foto.
Fluxo alternativo 02	No passo 4, o usuário pressiona o botão `Procurar Foto` e o sistema abre a galeria de fotos do dispositivo. O usuário seleciona uma imagem e a mesma estará visível no campo foto.
Exceção 01	No passo 4, o cliente não informou todos os campos obrigatórios. O sistema apresenta uma mensagem solicitando os dados obrigatórios e permanece na mesma tela.
Exceção 02	No passo 5, o cliente informou uma data de nascimento que resulta em uma idade inferior a 16 anos. O sistema apresenta uma mensagem informando que a pessoa deve ter mais que 16 anos para criar um perfil na aplicação.
Pós-condições	Os dados do usuário devem ter sido atualizados.

No caso de uso `Procurar usuários` o usuário pode pesquisar por outros usuários cadastrados no sistema. Mais detalhes estão descritos no Quadro 19.

Quadro 19 – Caso de uso `Procurar usuário`

UC04 - Procurar usuários	
Descrição	Permite pesquisar outros usuários cadastrados no sistema.
Requisitos atendidos	RF01 e RF02
Pré-condições	O usuário deve estar autenticado na aplicação e deve ter acessado a opção `Procurar usuários` no menu principal.
Cenário principal	1. O sistema apresenta o campo nome para que o usuário possa filtrar a pesquisa; 2. O sistema apresenta um botão para o usuário realizar a pesquisa; 3. O usuário informa obrigatoriamente o campo nome; 4. O usuário pressiona o botão `Procurar`; 5. O sistema realiza a busca dos usuários apresentando uma lista ordenada pelo nome dos mesmos com as informações: foto, nome, idade e endereço residencial. Além disso, para cada usuário encontrado, o sistema apresenta um botão para enviar uma requisição de amizade; 6. O sistema carrega as fotos dos usuários encontrados assincronamente; 7. O sistema apresenta somente os 5 primeiros usuários encontrados; 8. O usuário desliza a lista até o último usuário da mesma estar visível; 9. O sistema carrega, se houver, os próximos 5 usuários encontrados; 10. O fluxo retorna ao passo 8.
Fluxo alternativo 01	Após o passo 5, o usuário seleciona um usuário na lista. O sistema é redirecionado para o `UC05 – Visualizar perfil do usuário`.
Fluxo alternativo 02	Após o passo 5, o usuário pressiona o botão `Adicionar amigo` de um usuário na lista. O sistema cadastra uma requisição de amizade para aquele usuário.
Exceção 01	No passo 5, o sistema não encontra nenhum usuário para o filtro informado. O sistema deve apresentar uma mensagem informando que não foram encontrados usuários e deve permanece na mesma tela.
Exceção 02	No passo 5, um usuário encontrado está configurado para não exibir os detalhes do perfil e ainda não faz parte da lista de amigos. Para este usuário devem ser apresentadas somente as informações: foto e o nome do usuário.
Exceção 03	No passo 5, um usuário encontrado já está adicionado à lista de amigos ou já existe uma requisição de amizade entre eles. Para este usuário o botão para enviar uma requisição de amizade deve ser ocultado.
Pós-condições	Deve ser apresentada a lista com os usuários encontrados.

No caso de uso `Visualizar perfil do usuário` o usuário pode visualizar detalhadamente os dados de um usuário da aplicação. Mais detalhes estão descritos no Quadro 20.

Quadro 20 – Caso de uso `Visualizar perfil do usuário`

UC05 - Visualizar perfil do usuário	
Descrição	Permite a visualização dos dados de um usuário.
Requisito atendido	RF01
Pré-condições	O usuário deve estar autenticado na aplicação e deve ter selecionado um `Usuário 2` para visualizar o perfil no `UC04 - Procurar usuários` ou no `UC06 - Gerenciar amigos`.
Cenário principal	1. O sistema realiza a busca das informações do `Usuário 2` e as apresenta na aplicação; 2. O sistema carrega a foto do usuário assincronamente; 3. Se o `Usuário 2` faz parte da lista de amigos, o sistema apresenta um botão para enviar uma requisição de amizade. Caso contrário, o sistema apresenta um botão para desfazer a amizade.
Fluxo alternativo 01	Após o passo 3, o usuário pressiona o botão `Adicionar amigo`. O sistema cadastra uma requisição de amizade para aquele usuário.
Fluxo alternativo 02	Após o passo 3, o usuário pressiona o botão `Desfazer amizade`. O sistema exclui o `Usuário 2` da lista de amigos do usuário e vice-versa.
Exceção 01	No passo 1, o `Usuário 2` está configurado para não exibir os detalhes do perfil e ainda não faz parte da lista de amigos. Devem ser apresentadas somente as informações: foto e o nome do usuário.
Pós-condições	Devem ser apresentadas as informações do `Usuário 2`.

No caso de uso `Gerenciar amigos` o usuário pode aceitar e rejeitar requisições de amizade, além de visualizar a lista de amigos. Mais detalhes estão descritos no Quadro 21.

Quadro 21 – Caso de uso `Gerenciar amigos`

UC06 – Gerenciar amigos	
Descrição	Permite o usuário visualizar a lista de amigos e aceitar/rejeitar requisições de amizade.
Requisito atendido	RF02
Pré-condições	O usuário deve estar autenticado na aplicação e deve ter acessado a opção `Gerenciar amigos` no menu principal.
Cenário principal	1. O sistema realiza a busca dos amigos do usuário apresentando uma lista ordenada pelo nome dos mesmos com as informações: foto, nome, idade e endereço residencial. Além disso, o sistema apresenta um botão para desfazer a amizade; 2. O sistema realiza a busca das requisições de amizade onde o usuário é o destinatário, apresentando uma lista ordenada pela data de criação das mesmos com as informações do usuário remetente: foto, nome, idade e endereço residencial. Além disso, o sistema apresenta um botão aceitar e outro para rejeitar a amizade; 3. O sistema carrega as fotos dos usuários de ambas as listas assincronamente; 4. O usuário pressiona o botão `Aceitar`; 5. O sistema adiciona o usuário à lista de amigos do outro e vice-versa.
Fluxo alternativo 01	Após o passo 2, o usuário seleciona um usuário em uma das listas. O sistema é redirecionado para o `UC05 - Visualizar perfil do usuário`.
Fluxo alternativo 02	No passo 4, o usuário pressiona o botão `Rejeitar`. O sistema exclui a requisição de amizade sem alterar a lista de amigos dos usuários.
Fluxo alternativo 03	Após o passo 2, o usuário pressiona o botão `Excluir` na lista de amigos do usuário. O sistema exclui um usuário da lista do outro e vice-versa.
Exceção 01	No passo 2, o usuário apresentado na lista está configurado para não exibir os detalhes do perfil. Devem ser apresentadas somente as informações: foto e o nome do usuário.
Pós-condições	Deve ser apresentada a lista de amigos e a lista de requisições de amizade.

No caso de uso `Cadastrar local` o usuário pode cadastrar um novo local no sistema. Mais detalhes estão descritos no Quadro 22.

Quadro 22 – Caso de uso `Cadastrar local`

UC07 - Cadastrar local	
Descrição	Permite o cadastro de novos locais.
Requisitos atendidos	RF03 e RF04
Pré-condições	O usuário deve estar autenticado na aplicação e deve ter acessado a opção `Cadastrar Local` no menu principal.
Cenário principal	1. O sistema abre o Google Maps com o centro da tela na localização do GPS do dispositivo, indicando com um circulo um raio de 1.000 metros onde o usuário pode escolher a posição do novo local; 2. O usuário seleciona no mapa uma posição dentro do círculo; 3. O sistema fecha a janela do Google Maps e abre o formulário para o cadastro do local, disponibilizando um botão para escolher um novo ponto geográfico e outro botão para confirmar o cadastro do local; 4. O sistema preenche as informações do endereço do local (país, estado, município, bairro, logradouro, CEP e número) através do serviço do Google Maps enviando como parâmetro o ponto geográfico selecionado pelo usuário no passo 2. Estes campos preenchidos automaticamente não poderão ser editados, exceto o número; 5. O usuário preenche obrigatoriamente os campos: foto, nome e categoria; 6. O usuário preenche opcionalmente os campos: complemento do endereço, telefone e site; 7. O usuário preenche obrigatoriamente os campos relativos à recomendação do local: nota e opinião; 8. O usuário pressiona o botão `Cadastrar Local`; 9. O sistema realiza o cadastro do local; 10. O sistema realiza o cadastro da recomendação através do `UC10 - Realizar recomendação de local`, a partir do passo 4.
Fluxo alternativo 01	Após o passo 4, o usuário pressiona o botão `Alterar endereço`. O fluxo retorna ao passo 1.
Fluxo alternativo 02	No passo 5, o usuário pressiona o botão `Bater Foto` e o sistema é redirecionado para o `UC13 - Capturar imagem`. No retorno, a imagem capturada estará visível no campo foto.
Fluxo alternativo 03	No passo 5, o usuário pressiona o botão `Procurar Foto` e o sistema abre a galeria de fotos do dispositivo. O usuário seleciona uma imagem e a mesma estará visível no campo foto.
Exceção 01	No passo 2, o usuário seleciona uma posição fora do circulo. O sistema apresenta uma mensagem informando que deve ser escolhida uma posição dentro do circulo definido.
Pós-condições	O local e a recomendação devem estar disponíveis para visualização dos usuários da aplicação.

No caso de uso `Visualizar locais recomendados` o usuário pode procurar por locais recomendados por outros usuários através de diversos filtros. O resultado é visualizado através da

câmera, apresentando as informações dos locais encontrados em RA. Mais detalhes estão descritos no Quadro 23.

Quadro 23 – Caso de uso `Visualizar locais recomendados`

UC08 - Visualizar locais recomendados	
Descrição	Permite, através de realidade aumentada, visualizar os locais recomendados por outros usuários.
Requisitos atendidos	RF05 e RF06
Pré-condições	O usuário deve estar autenticado na aplicação e deve ter acessado a opção `Procurar locais` no menu principal.
Cenário principal	1. O sistema apresenta as opções de filtro ao usuário e um botão para realizar a busca dos locais; 2. O usuário informa obrigatoriamente os campos distância máxima e nota mínima. O usuário informa também ao menos um dos campos: categorias e amigos; 3. O usuário pressiona o botão `Procurar locais`; 4. O sistema realiza a busca dos locais de acordo com os filtros informados; 5. O sistema apresenta a imagem da câmera do dispositivo; 6. O sistema apresenta em sobreposição à imagem da câmera os locais encontrados e que estão no máximo à 45º a esquerda ou 45º a direita em relação à direção da bússola do dispositivo; 7. O usuário gira o dispositivo no eixo X; 8. O sistema executa o passo 6 novamente.
Pós-condições	O usuário deve visualizar, através de realidade aumentada, os locais encontrados.

No caso de uso `Visualizar detalhes do local` o usuário pode visualizar detalhadamente os dados de um local. Mais detalhes estão descritos no Quadro 24.

Quadro 24 – Caso de uso `Visualizar detalhes do local`

UC09 - Visualizar detalhes do local	
Descrição	Permite visualizar os detalhes de um local, bem como as opiniões dos outros usuários em relação ao mesmo.
Requisitos atendidos	RF03 e RF07
Pré-condições	O usuário deve estar autenticado na aplicação e deve ter selecionado um local no `UC08 - Visualizar locais recomendados`.
Cenário principal	1. O sistema apresenta as informações do local, além de botões para visualizar a rota e editar local; 2. O sistema apresenta, alem das informações do local, a recomendação do criador do local seguida pelas recomendações dos amigos e depois as demais recomendações; 3. O sistema apresenta para cada recomendação: foto e nome do recomendante, opinião, nota e a data em que foi realizada a recomendação; 4. O sistema carrega a foto do local e as fotos dos usuários recomendantes assincronamente; 5. O sistema apresenta somente as 5 primeiras recomendações encontradas; 6. O usuário desliza a lista até a última recomendação da mesma estar visível; 7. O sistema carrega, se houver, as próximas 5 recomendações encontradas; 8. O fluxo retorna ao passo 6.
Fluxo alternativo 01	O usuário pressiona o botão `Editar local`. O sistema é redirecionado para o `UC11 - Editar local`.
Fluxo alternativo 02	O usuário pressiona o botão `Visualizar rota`. O sistema é redirecionado para o `UC12 - Visualizar rota para local`.
Pós-condições	As informações do local devem estar visíveis ao usuário.

No caso de uso `Realizar recomendação de local` o usuário pode adicionar uma recomendação sobre um local. Mais detalhes estão descritos no Quadro 25.

Quadro 25 – Caso de uso `Realizar recomendação de local`

UC10 - Realizar recomendação de local	
Descrição	Permite recomendar positiva ou negativamente um local.
Requisito atendido	RF04
Pré-condições	O usuário deve estar autenticado na aplicação e deve estar visualizando um local no `UC09 - Visualizar detalhes do local` ou estar inserindo um local no `UC07 - Cadastrar local`.
Cenário principal	1. O sistema apresenta o formulário com os campos para inserir uma nova recomendação e outro botão para cadastrar a recomendação; 2. O usuário preenche obrigatoriamente os campos: nota e opinião; 3. O usuário pressiona o botão `Cadastrar recomendação`; 4. O sistema realiza o cadastro da recomendação.
Exceção 01	No passo 1, o ponto geográfico do local a ser recomendado não está dentro de um raio de 1.000 metros da localização do GPS do dispositivo. A realização da recomendação é cancelada e o sistema apresenta uma mensagem informando o cancelamento ao usuário.
Pós-condições	A recomendação deve estar visível para os outros usuários..

No caso de uso `Editar local` o usuário pode editar algumas informações ou até mesmo excluir um local. Mais detalhes estão descritos no Quadro 26.

Quadro 26 – Caso de uso `Editar local`

UC11 - Editar local	
Descrição	Permite editar informações sobre um local já cadastrado ou até mesmo excluí-lo do sistema.
Requisito atendido	RF03
Pré-condições	O usuário deve estar autenticado na aplicação e deve ter pressionado o botão `Editar local` no `UC09 - Visualizar detalhes do local`.
Cenário principal	1. O sistema apresenta os campos do local para alteração, carregando os dados atuais; 2. Os campos de endereço (exceto o complemento), foto e nome devem ser apresentados somente para leitura; 3. O sistema apresenta um botão para confirmar as alterações e outro botão para excluir o local; 4. O usuário preenche opcionalmente os campos: complemento do endereço, telefone e site; 5. O usuário pressiona o botão `Confirmar`; 6. O sistema atualiza os dados do local.
Fluxo alternativo 01	O usuário pressiona o botão `Excluir`. O sistema exclui o local assim como as suas recomendações.
Fluxo alternativo 02	No passo 2, o usuário autenticado não é o criador do local sendo editado. O sistema deve ocultar o botão `Excluir`.
Pós-condições	Os dados do local devem ser atualizados.

No caso de uso `Visualizar rota para local` o usuário pode visualizar a rota a pé ou de carro da sua posiçao atual até um local selecionado através do Google Maps. Mais detalhes estão descritos no Quadro 27.

Quadro 27 – Caso de uso `Visualizar rota para local`

UC12 - Visualizar rota para local	
Descrição	Permite a visualização das rotas para chegar de carro ou a pé em um determinado local utilizando os serviços e a API do Google Maps.
Requisito atendido	RF08
Pré-condições	O usuário deve estar autenticado na aplicação e deve ter pressionado o botão `Visualizar rota` no `UC09 - Visualizar detalhes do local`.
Cenário principal	1. O sistema apresenta uma janela com duas opções: uma para visualizar a rota a pé e outra para visualizar a rota de carro; 2. O usuário seleciona uma das opções; 3. O sistema abre o Google Maps indicando a rota da localização atual do GPS até o local em questão.
Pós-condições	A rota para o local deve estar visível ao usuário.

Por fim, no caso de uso `Capturar imagem` o usuário utiliza a câmera do dispositivo para capturar uma imagem. Mais detalhes estão descritos no Quadro 28.

Quadro 28 – Caso de uso `Capturar imagem`

UC13 - Capturar imagem	
Descrição	Permite capturar uma imagem da câmera e guardá-la em disco para uso posterior.
Requisito atendido	RF01 e RF03
Pré-condições	O usuário deve estar autenticado na aplicação.
Cenário principal	1. O sistema apresenta a imagem da câmera do dispositivo e apresenta um botão para capturar a imagem; 2. O usuário pressiona o botão `Capturar imagem`; 3. O sistema guarda a imagem capturada em disco e volta à tela de onde o caso de uso foi chamado.
Pós-condições	A foto capturada deve estar gravada no disco do dispositivo.