Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

RockDroid - Uma Arquitetura para Coleta de Dados Geológicos

Bernardo Augusto Pereira de Macêdo
Renata Cristina Machado Nunes

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.ª Dr.ª Maristela Terto de Holanda

Coorientadora
Prof.ª Dr.ª Tati de Almeida

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Homero Luiz Piccolo

Banca examinadora composta por:

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora) — CIC/UnB
Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo — CIC/UnB
Prof. MSc. Henrique Pereira de Freitas Filho — IFG

CIP — Catalogação Internacional na Publicação

de Macêdo, Bernardo Augusto Pereira.

RockDroid - Uma Arquitetura para Coleta de Dados Geológicos / Bernardo Augusto Pereira de Macêdo, Renata Cristina Machado Nunes. Brasília : UnB, 2016.
199 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2016.

1. RockDroid, 2. Android, 3. aplicativo, 4. coleta de dados, 5. banco de dados, 6. geologia, 7. pesquisa de campo, 8. sincronização

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

RockDroid - Uma Arquitetura para Coleta de Dados Geológicos

Bernardo Augusto Pereira de Macêdo
Renata Cristina Machado Nunes

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora)
CIC/UnB

Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo
CIC/UnB

Prof. MSc. Henrique Pereira de Freitas Filho
IFG

Prof. Dr. Homero Luiz Piccolo
Coordenador do Bacharelado em Ciência da Computação

Brasília, 28 de Janeiro de 2016

Dedicatória

À nossa orientadora Prof.ª Dr.ª Maristela Terto de Holanda, que nos ofereceu a oportunidade de iniciar nossos estudos na área de desenvolvimento de aplicativos para o sistema operacional *Android* há anos atrás e que, recentemente, nos deu a chance de dar prosseguimento ao nosso trabalho.

A todos os outros professores que nos ajudaram a construir o conhecimento necessário para desenvolver este trabalho.

Agradecimentos

Aos nossos pais, que sempre nos deram apoio em tudo o que nos propusemos a fazer, que nos acompanharam em nossa jornada e que nos fizeram ser quem somos hoje.

À Prof.ª Dr.ª Maristela Terto de Holanda, que nos incentivou e nos ofereceu a oportunidade de realizar este trabalho.

À Prof.ª Dr.ª Tati de Almeida e ao Prof. Dr. Henrique Llacer Roig, que nos forneceram o conhecimento necessário para desenvolver este aplicativo.

Ao Prof. MSc. Henrique Freitas, que esteve sempre disposto a nos ajudar e que nos acompanhou nessa jornada de aprendizado.

Ao Fabrício Chaves, que esteve presente em várias reuniões e que nos ajudou em uma parte importantíssima deste trabalho.

A todos os outros professores e colegas que de alguma forma contribuíram para a nossa formação.

Resumo

Este documento apresenta o RockDroid, aplicativo desenvolvido para o sistema operacional *Android* que visa auxiliar geólogos durante suas pesquisas de campo, fornecendo ferramentas e formulários para facilitar seu trabalho de coleta de dados. A capacidade de trabalhar em modo *offline* e a sincronização com um banco de dados central são duas das principais características do RockDroid. O aplicativo encontra-se descrito neste trabalho e sua validação foi realizada por alunos e especialistas da área da geologia e pesquisadores de áreas relacionadas, como a geofísica.

Palavras-chave: RockDroid, Android, aplicativo, coleta de dados, banco de dados, geologia, pesquisa de campo, sincronização

Abstract

This document presents RockDroid, an application developed for the operating system *Android* that aims to assist geologists during their field researches by providing tools and forms to ease their work of data collection. The ability of working in offline mode and the synchronization with a central database are two of the main features of RockDroid. The application is described in this paper and its validation was performed by students, geology experts and researchers in related fields such as geophysics.

Keywords: RockDroid, Android, application, gathering data, database, geology, field research, synchronization

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

Os recursos presentes nos dispositivos móveis atuais os tornam úteis em diversos contextos. O desenvolvimento de tecnologias para ambientes de computação móvel e o crescente número de aparelhos de uso pessoal ou institucional, como *smartphones* e *tablets*, permitem que o usuário tenha acesso a uma infraestrutura compartilhada independente de sua localização física. Desta evolução, surge a necessidade de acesso aos dados armazenados de forma distribuída nos diversos aparelhos móveis [14].

Dentre os recursos presentes nos dispositivos móveis disponíveis atualmente no mercado, existe a capacidade de reconhecimento de localização através de seu *Global Positioning System* (GPS), rede de telefonia celular, ou de uma conexão com a *Internet*. Desta forma, *smartphones* e *tablets* tornam-se opções viáveis para a coleta de dados geológicos [10].

O ambiente de computação móvel permite metodologias de coleta de dados nas quais os pesquisadores podem gravar as informações em seus dispositivos e submetê-las a outras máquinas na rede, permitindo que outras equipes trabalhem paralelamente sobre tais informações. A solução descrita neste trabalho propõe-se a atender às necessidades de geólogos durante suas pesquisas de campo, facilitando a coleta, o armazenamento e a sincronização dos dados, usando, para isto, *smartphones* e *tablets*, recursos de computação móvel de fácil acesso aos membros das equipes de coleta.

Contudo, um dos desafios da implementação da arquitetura de coleta de dados citada é garantir a integridade e a consistência dos bancos de dados envolvidos. Além disso, é importante que haja uma forma de reunir todos os dados coletados individualmente pelos pesquisadores em um banco de dados central, para possibilitar acesso a todos os dados coletados pelas equipes e sua posterior análise. Percebe-se então que é imprescindível que haja um mecanismo de sincronização entre os bancos de dados móveis e o banco de dados central.

O aplicativo criado para suprir as necessidades mencionadas acima é chamado RockDroid e foi desenvolvido para plataforma *Android*, levando em consideração as várias limitações impostas pelo contexto de coleta de dados em campo, como por exemplo a falta de acesso à *Internet*, o espaço de armazenamento limitado dos dispositivos móveis, dentre outras. Além disso, houve a preocupação em utilizar apenas bibliotecas de código aberto para facilitar sua manutenção e disponibilização.

1.2 Objetivo

O objetivo geral deste trabalho é desenvolver um sistema de coleta de dados - RockDroid - voltado para pesquisas de campo geológicas em um ambiente computacional heterogêneo que permita ao pesquisador em campo armazenar dados mesmo sem possuir acesso à *Internet*, bem como a sincronização dessas informações com um banco de dados central quando o dispositivo estiver conectado.

1.2.1 Objetivos Específicos

Para o desenvolvimento do objetivo geral, os seguintes objetivos específicos devem ser realizados:

- Desenvolver um aplicativo de formulário digital para o sistema operacional *Android* que realize a validação dos dados cadastrados e que os estruture de forma organizada;
- Criar um banco de dados central para os dados coletados;
- Criar um mecanismo de sincronização que permita a transmissão dos dados cadastrados para o repositório central.

1.3 Estrutura do Trabalho

Este documento é composto pelos seguintes capítulos:

- **Capítulo 2 - Fundamentação Teórica:** apresentação dos conceitos necessários para o entendimento da pesquisa. Neste capítulo são abordados os seguintes temas: sistemas de coleta de dados móveis, computação móvel, bancos de dados móveis, serviços *web* RESTful e sistema operacional *Android*.
- **Capítulo 3 - O RockDroid:** apresentação de sistemas similares que serviram de inspiração para a criação do RockDroid, explicação do processo de levantamento de requisitos e descrição do sistema proposto, incluindo sua arquitetura e seu processo de desenvolvimento.
- **Capítulo 4 - Prova de Conceito:** descrição dos testes realizados na aplicação e de seus resultados, apontando possíveis problemas e pontos a melhorar.
- **Capítulo 5 - Conclusão:** apresentação das conclusões e dos trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo visa apresentar os principais conceitos necessários para o entendimento completo deste trabalho. Isto inclui uma breve discussão sobre sistemas de coleta de dados, computação móvel, bancos de dados móveis, serviços *web* RESTful e sobre o sistema operacional *Android*.

2.1 Sistema de Coleta de Dados Móveis

Cadernetas de papel são comumente usadas por geólogos para coletar dados em pesquisas de campo, mas atualmente há uma tendência a modernizar esse procedimento através da utilização de meios eletrônicos para coleta e processamento de dados.

Um dos maiores problemas da coleta por meio de formulários de papel é o intervalo de tempo entre a coleta dos dados e sua validação, que geralmente ocorre durante o processo de digitalização dos mesmos. Eventuais erros detectados nesta etapa só poderiam ser corrigidos caso uma nova coleta fosse realizada, o que pode não ser possível em algumas situações. Além disso, há todo um retrabalho envolvido no processo de digitalizar os dados coletados em papel para posterior análise computacional [21]. Outras desvantagens no uso de formulários de papel é a inviabilidade de se realizar a coleta e o processamento dos dados de forma simultânea e os dados são mais vulneráveis no que se refere à segurança da informação [7].

Quando se compara os dois métodos de coleta, em papel e o digital, em grande parte dos casos há uma melhora na precisão e na completude dos dados quando se utiliza algum meio eletrônico. A razão para isto é a validação dos dados durante sua coleta, que não permite que campos importantes fiquem em branco ou possuam valores incorretos. O tempo de coleta também é reduzido, resultando em mais dados coletados no mesmo período de tempo [21].

O custo de se utilizar meios eletrônicos para a coleta pode parecer superior ao custo de se utilizar formulários de papel, mas geralmente não é, já que no segundo método são incluídos custos de equipamentos de impressão, de materiais e de pessoal para realizar as coletas (levando em consideração que o tempo de coleta é maior e que, para suplantar isto, talvez seja necessário ter equipes maiores) e digitalizar os dados posteriormente, entre outros [21].

No contexto deste trabalho, o maior desafio para a modernização dos meios de coleta é criar um *software* que seja adequado para a grande maioria dos pesquisadores de geologia,

visto que até mesmo dentro do Instituto de Geociências da Universidade de Brasília (IG/UnB) não existe um processo de coleta amplamente definido e cada pesquisador possui suas prioridades e preferências.

Outro grande desafio é transferir os dados coletados do dispositivo do pesquisador para um banco de dados central, para que sejam agregados e analisados, porque eles raramente possuem acesso à *Internet* quando estão realizando suas coletas em lugares isolados.

O aplicativo desenvolvido neste trabalho, RockDroid, é um sistema de coleta de dados móveis que permite a coleta dos dados em localidades remotas e a transferência desses dados para um banco de dados central através da *Internet* quando esta estiver disponível. Os dados serão coletados por meio de um formulário digital no dispositivo do pesquisador. Este formulário será responsável por validar alguns dos dados inseridos, diminuindo as chances de haver erros ou dados incompletos.

O pesquisador terá a opção de exportar estes dados para um arquivo em seu próprio celular ou sincronizar os dados com o banco de dados central, localizado na Universidade de Brasília (UnB), agilizando assim o processo de análise. Um esquema desta arquitetura pode ser visto na Figura 2.1. Como pode ser observado, cada pesquisador deve ter seu próprio dispositivo móvel, que terá um banco de dados local para armazenar as informações coletadas. Após a coleta, os dados poderão ser sincronizados para o banco de dados central, desde que haja conexão com a *Internet*.

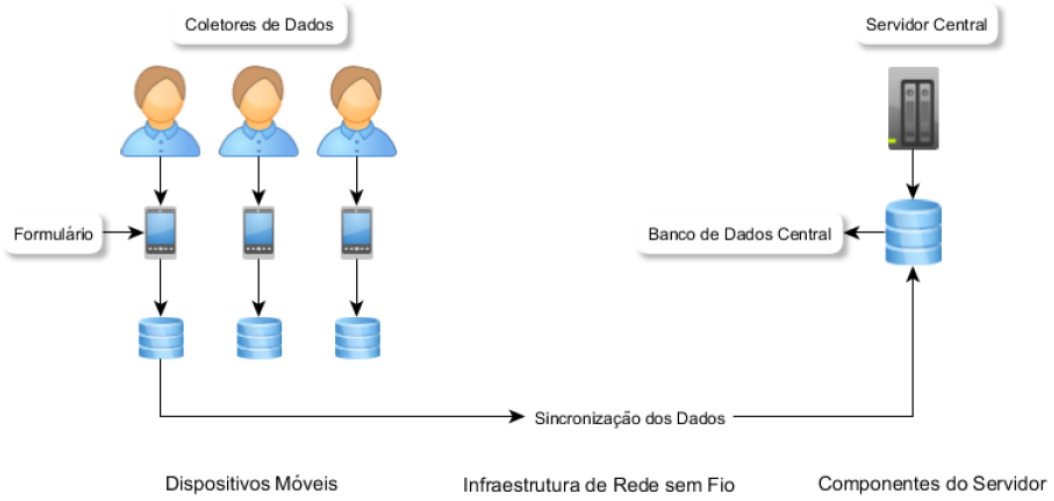

Figura 2.1: Arquitetura do Sistema de Coleta de Dados Móveis [7].

2.2 Computação Móvel

A crescente evolução dos computadores portáteis juntamente com o avanço nas tecnologias de comunicação sem fio têm possibilitado aos usuários de dispositivos móveis acesso a serviços e informações com pouca ou nenhuma restrição de tempo e lugar. A esse paradigma computacional é dado o nome de **computação móvel** ou **computação nômade** [22]. A palavra-chave desse novo paradigma é mobilidade. Neste ambiente, não é necessário que o dispositivo computacional tenha uma posição fixa na rede, o acesso a serviços independe de onde o dispositivo esteja localizado e, mais importante, de mu-

danças de localização [12]. Este cenário é responsável por mudanças na forma como as pessoas trabalham, estudam e usam seu tempo, bem como na forma como as empresas desenvolvem e divulgam seus serviços e interagem com seus clientes.

A computação móvel amplia o conceito de computação distribuída no sentido de que não é necessário que o usuário esteja conectado a uma infraestrutura fixa e estática, graças à comunicação sem fio; apenas uma parte segue o formato tradicional, formada por uma infraestrutura de comunicação fixa com computadores estáticos. No entanto, a mobilidade trazida pela computação móvel introduz restrições que não são muito frequentes na computação distribuída tradicional [12], tais como: desconexão, banda limitada, alta latência, baixa capacidade de armazenamento e consumo de energia. Assim sendo, as mesmas soluções utilizadas para sistemas distribuídos não podem ser simplesmente usadas em sistemas de computação móvel, devendo ser analisadas e modificadas com o objetivo de fornecer aos usuários um conjunto de serviços comparáveis aos existentes na computação distribuída convencional [6].

2.2.1 Arquitetura

A mobilidade permite que os usuários movam-se livremente enquanto mantêm uma conexão com uma infra-estrutura de comunicação, tendo acesso aos serviços, recursos e dados distribuídos. Essa é a característica que difere a computação móvel da computação distribuída tradicional.

A topologia do ambiente de computação móvel, apresentada na Figura 2.2, consiste de computadores móveis e uma rede de alta velocidade que interconecta computadores fixos e estações-base [10].

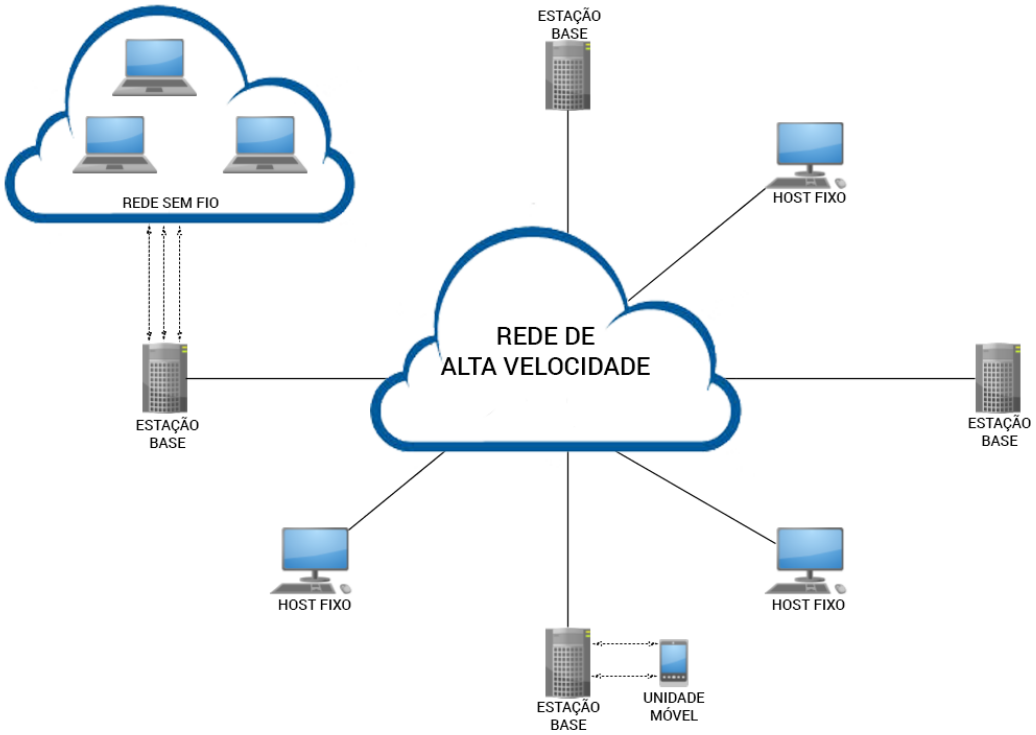

Figura 2.2: Arquitetura da Computação Móvel, adaptada de [20].

As estações-base, também chamadas de estações de suporte móveis, agem como intermediárias na comunicação entre a rede fixa e os dispositivos móveis, recebendo as mensagens destinadas a eles e encaminhando-as para o endereço correto. Isto implica que as estações-base têm conhecimento da localização dos dispositivos móveis que estão dentro de seu alcance. Cada estação-base é responsável por gerenciar uma região geográfica chamada de célula, e é essa região que determina o seu alcance. Além de conhecer a localização dos dispositivos que estão dentro de suas células, as estações-base são capazes também de detectar dispositivos que entram e saem de seu domínio [13].

Os computadores fixos são computadores de propósito geral que podem disponibilizar serviços para os dispositivos móveis, mas que não se comunicam diretamente com eles. Já os computadores móveis, também chamados de usuários ou clientes, podem requisitar serviços disponibilizados pelos computadores fixos (servidores) e se comunicam com as estações-base através de um canal de comunicação sem fio com dois *links*: *uplink*, que transmite dados dos clientes para as estações, e *downlink*, que transmite dados das estações para os clientes [13]. Caso os computadores móveis estejam próximos, a comunicação pode ser feita diretamente entre eles, sem a intervenção da estação-base [10].

2.2.2 Características

A computação móvel se diferencia da computação distribuída convencional em alguns aspectos que são descritos mais detalhadamente nas subseções a seguir.

Conexão a redes sem fio

A comunicação sem fio é mais difícil de ser estabelecida do que a conexão física convencional. Além disso, está sujeita a interferências do meio no sinal, por meio de ruídos e ecos. Conexões sem fio possuem qualidade mais baixa, largura de banda estreita, alta taxa de erro e desconexões frequentes. Tais fatores podem provocar mais retransmissões, que contribuem para o aumento da latência.

As limitações introduzidas pelas redes sem fio impõem severas restrições no volume de dados que pode ser transmitido e no número de usuários que podem se conectar simultaneamente. A própria estrutura da rede também cria vulnerabilidades na segurança e privacidade dos dados, exigindo protocolos de segurança e autenticação para evitar que estes sejam usados de forma maliciosa [6].

Mobilidade

É um dos principais objetivos da computação móvel e permite que a localização de um dispositivo móvel e, consequentemente, seu ponto de acesso à rede mudem constantemente à medida que o usuário se move. Por esse motivo, a topologia torna-se bastante dinâmica e acaba apresentando desafios infrequentes na computação tradicional. Dentre os problemas que a mobilidade apresenta estão as variações de tráfego, interferências, velocidade do canal, gerenciamento de localização da unidade móvel e outros [10].

Devido à mudança constante de localização das unidades dentro da rede, um dispositivo móvel deve primeiramente ter sua localização pesquisada antes de receber mensagens. Este é um dos motivos pelos quais algoritmos de sistemas distribuídos não podem ser simplesmente mapeados para sistemas de computação móvel. O projeto de protocolos e

algoritmos deve levar em consideração questões como privacidade, segurança, gerência de localização e variações na carga do sistema [6].

Para solucionar a questão de redirecionamento de mensagens, faz-se com que os computadores móveis informem suas localizações às suas estações-base toda vez que mudarem seu ponto de conexão com a rede. Quando um computador quiser enviar uma mensagem a um dispositivo móvel, ele enviará essa mensagem para a estação-base deste dispositivo e esta redirecionará o pacote para o dispositivo em si [12].

Desconexão

Desconexões são frequentes em ambientes de computação móvel e podem ser classificadas em voluntárias ou involuntárias. Desconexões voluntárias ocorrem quando o usuário ou o dispositivo decidem cortar o acesso à rede por motivos diversos, como diminuir os custos das tarifas de comunicação ou o consumo de energia. Já as desconexões involuntárias acontecem quando o dispositivo entra em uma região sem cobertura. Pelo fato de desconexões serem muito comuns neste contexto, protocolos, sistemas operacionais e aplicativos devem estar preparados para lidar com elas. Desconexão não é necessariamente uma falha; ela pode ser tratada como uma falha planejada, que pode ser antecipada. Alguns exemplos de desconexões previsíveis são [12]:

- Desconexão voluntária;
- Variações na taxa do sinal;
- Pouca energia disponível na bateria;
- Conhecimento da distribuição da largura de banda num determinado momento.

Handoff

Handoff é o processo de movimentação de computadores móveis entre duas estações-base adjacentes durante a execução de uma aplicação. A transação em execução deve ser mantida, bem como a conectividade da unidade móvel com a rede, de forma a manter esse processo invisível para o usuário [10].

Heterogeneidade

Devido à mobilidade, uma unidade móvel pode encontrar diferentes ambientes e serviços de rede, necessitando de protocolos de acesso específicos. Os sistemas devem dar suporte a redes com diferentes características para evitar problemas de interoperabilidade. Porém, nem sempre isto é possível e a conectividade entre os elementos da rede não é sempre garantida.

Outro problema relacionado à heterogeneidade é o custo de acesso. Diferentes redes impõem diferentes custos. Portanto, o custo de uma consulta a um banco de dados central, por exemplo, depende da localização do usuário. Essas informações devem ser levadas em conta quando se deseja desenvolver métodos para otimização de consultas [6].

2.2.3 Problemas e Desafios

Alguns problemas e desafios da computação móvel são [14]:

- **Características do ambiente:** como já mencionado, as próprias características inerentes à mobilidade criam dificuldades no desenvolvimento de soluções. Dentre elas estão largura de banda limitada, alta taxa de erros, desconexões frequentes e maior taxa de latência.
- **Portabilidade:** o projeto de soluções para aplicações distribuídas deve levar em consideração vários fatores relacionados aos dispositivos portáteis, principalmente seu tamanho, que influencia diretamente no desenvolvimento da interface com o usuário. Dispositivos menores podem contribuir para a mobilidade e portabilidade, porém acabam possuindo um tamanho de tela limitado e dificultando o projeto da interface [6].
- **Adaptação:** para que as unidades móveis funcionem corretamente em meio às constantes mudanças de contexto causadas pela mobilidade, elas devem ser adaptativas. Adaptação é a capacidade de um sistema de se ajustar e produzir resultados apesar de condições adversas. A responsabilidade de adaptação pode ser inteiramente dos aplicativos (nesse caso, falta um gerenciador central capaz de resolver incompatibilidades de demanda de recursos), inteiramente do sistema (a infraestrutura computacional é responsável por gerenciar recursos), ou dividida entre o sistema e as aplicações (as aplicações determinam o quanto devem se adaptar, enquanto o sistema monitora os recursos e gerencia sua alocação) [7].
- **Energia:** é indispensável que dispositivos móveis possuam sua própria fonte de energia. O problema é que atualmente as baterias disponíveis só conseguem armazenar energia para algumas horas de uso. Este problema é visto como o maior empecilho no uso de computadores móveis e deve ser tratado tanto pelo hardware quanto pelo *software*. Por ser um recurso limitado, seu consumo deve ser minimizado [13].
- **Capacidade dos dispositivos móveis:** devido ao seu tamanho limitado e à sua restrição de energia, tais dispositivos não comportam muitos *chips* de memória ou processadores muito potentes e, consequentemente, não possuem grande capacidade de memória e de processamento [6].
- **Segurança:** os dados transmitidos através de redes sem fio podem ser mais facilmente interceptados e usados para fins maliciosos caso não haja nenhum protocolo de autenticação e criptografia implementado [12].
- **Privacidade:** uma unidade móvel que está se comunicando com a parte fixa da rede pode ser rastreada (como é feito constantemente pelas estações-base) e isso nem sempre é desejável pelos usuários [6].
- ***Seamless communication:*** a mudança entre infra-estruturas de comunicação com o objetivo de evitar desconexões deveria ser imperceptível para o usuário, mas atualmente isso não é possível [14].

2.2.4 Dispositivos para Computação Móvel

Tendo em vista as características da computação móvel citadas, um dispositivo para atender esta demanda precisa possuir capacidade de processamento, trocar informações via rede e ser portátil. Seu tamanho deve ser reduzido e ele não deve ser limitado por cabos de conexão com a rede ou fonte de energia elétrica - consequentemente, o dispositivo deve possuir uma bateria e capacidade de conectar-se a redes sem fio. Os dispositivos mais usados para este fim são *laptops*, *smartphones* e *tablets* [14].

Laptops possuem capacidade e características comparáveis às de um computador comum, porém ainda não são a melhor opção para uso em movimento devido ao seu tamanho, que exige um lugar propício para usá-los, e a sua bateria, que não possui grande autonomia e deve ser recarregada constantemente.

Smartphones e *tablets* possuem um tamanho reduzido e isto, apesar de contribuir para sua mobilidade, acaba limitando sua capacidade de processamento, armazenamento e energia. Porém eles vêm evoluindo bastante e atualmente possuem uma vasta gama de recursos. Por este motivo, são os mais indicados para suprir as necessidades da computação móvel e são o foco deste trabalho.

2.3 Bancos de Dados Móveis

Bancos de dados também são influenciados pela presença de usuários móveis. Devido às limitações impostas pelos computadores portáteis, existe a necessidade de novas formas de acesso aos dados que foquem na redução do número de acesso aos servidores de arquivos, diminuindo o tráfego de informações na rede. As características das redes sem fio também exigem soluções capazes de lidar com problemas como a desconexão frequente, a largura de banda limitada e outros [10].

Tendo isto em vista, novos paradigmas de transação devem ser desenvolvidos para tratar usuários que se movimentam e se desconectam constantemente, além de mecanismos para manutenção da consistência de dados e para o tratamento de consultas quando a unidade móvel não tiver acesso à *Internet* [12].

2.3.1 Arquitetura

Uma arquitetura básica é formada por computadores móveis, um servidor com um banco de dados central e a comunicação realizada entre eles, como mostrado na Figura 2.3. Os computadores possuem parte do banco e um sistema gerenciador de bancos de dados móveis, através do qual os usuários interagem e acessam informações armazenadas no banco de dados central. O servidor que armazena o banco de dados principal contém dados dinâmicos que podem ser acessados por usuários móveis. A posição deste servidor na rede depende do tipo de informações que estão sendo processadas [10].

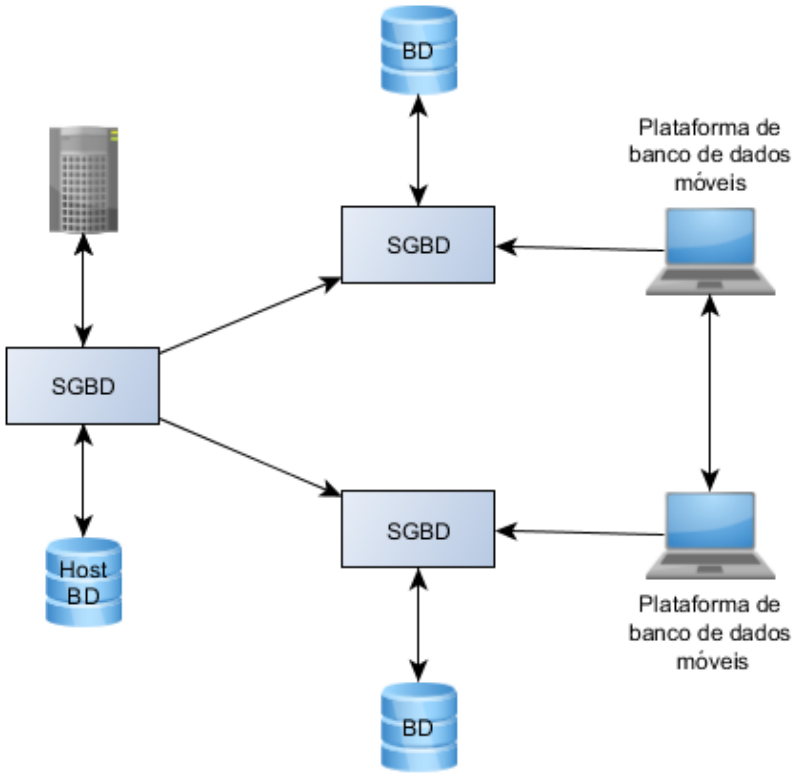

Figura 2.3: Arquitetura de Bancos de Dados Móveis, adaptada de [10].

Os *softwares* para ambientes distribuídos devem lidar com a topologia dinâmica e com os problemas da computação móvel, principalmente a desconexão frequente dos usuários. O banco de dados móvel troca informações com o banco de dados central para se manter atualizado, realizar consultas, etc. A comunicação entre os dois ocorre mesmo que eles não estejam conectados na mesma rede, mas em intervalos irregulares, por curtos períodos de tempo e com interrupções.

2.3.2 Características

As desconexões frequentes causadas pelas limitações das baterias e pela própria mobilidade dos dispositivos portáteis fazem com que a comunicação entre os clientes móveis e o servidor fixo seja instável. Além disso, o custo de se usar comunicação de rede sem fio pode ser elevado, uma vez que a velocidade de transmissão nesses enlaces é baixa e as taxas podem ser cobradas por tempo de uso. Todos esses fatores incentivam os computadores móveis a manterem-se desconectados por longos períodos de tempo. Dessa forma, é necessário prover suporte à desconexão, projetando soluções para possibilitar aos usuários executarem suas operações apesar das adversidades do meio [13].

Nas subseções a seguir, encontram-se algumas características dos bancos de dados móveis que lidam com os desafios mencionados.

Caching e difusão de dados

O mecanismo de *caching* tem como objetivo o acesso mais eficiente aos dados, aumentando a probabilidade de o dado requerido estar armazenado localmente na unidade móvel. Dados que serão possivelmente reutilizados são mantidos no banco de dados local, o que reduz o tempo de resposta às consultas e melhora o desempenho do sistema [10].

Quando os dados necessários não se encontram na *cache*, o cliente faz uma solicitação ao servidor, que pode ser total, quando o cliente solicita todos os dados do servidor, ou parcial, quando apenas alguns dados são necessários para responder à consulta [7].

A difusão de dados, também chamada de *broadcasting*, é um sistema que envia as mensagens de uma estação-base para todas as unidades móveis que estejam conectados a ela. Dentre as vantagens de se utilizar a difusão de dados estão a economia de energia para o dispositivo móvel, que não precisa transmitir uma solicitação, e a facilidade de se transmitir dados para vários clientes simultaneamente, quando estes possuem as mesmas necessidades de atualizações [19].

Replicação de dados

Replicação é o processo no qual as transações são propagadas assincronicamente para um ou mais bancos de dados de forma serial, ou seja, elas são replicadas na mesma ordem em que foram solicitadas, a fim de manter a consistência entre a unidade móvel e o servidor [10]. Propagar de forma assíncrona significa armazenar as operações que serão replicadas do servidor em um banco de dados fonte até que haja conexão para que elas sejam enviadas para as unidades móveis.

As unidades móveis utilizam replicação de dados para armazenar aqueles utilizados mais frequentemente durante as desconexões. Assim, a leitura é realizada localmente, independente de comunicação com o servidor. A replicação completa é melhor que o método de *caching*, porque o computador portátil pode se desconectar e trabalhar em seu banco de dados local. Quando o dispositivo for reconectado, todos os arquivos atualizados serão reintegrados.

Gerenciamento de dados móveis

Na área de gerenciamento de dados, os maiores desafios estão relacionados à otimização de consultas a bancos de dados, organização, alocação e replicação de dados entre um computador móvel e um computador fixo [12]. São necessárias mudanças no gerenciamento e nos mecanismos de garantia da consistência dos dados.

Muitas das questões de gerenciamento de dados distribuídos também se aplicam aos bancos de dados móveis, desde que se leve em conta os seguintes pontos [8]:

- **Distribuição de dados e replicação:** os dados podem estar desigualmente e irregularmente distribuídos entre unidades móveis e servidores. Os dados em *cache* devem ser gerenciados de forma a manter a consistência dos bancos.
- **Modelos de transação:** para lidar com questões de tolerância a falhas e corretude, as transações e a forma como elas são executadas devem ser ser adaptadas. No ambiente de computação móvel, as transações são executadas de forma sequencial através de diferentes estações-base e possivelmente em múltiplos conjuntos de

dados, de acordo com a movimentação da unidade móvel. As propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) podem precisar de modificação.

- **Processamento de consultas:** é necessário possuir informações sobre a localização dos dados, sendo este um dos fatores mais importantes no desempenho de uma consulta. O processamento deve ser realizado de forma completa e correta mesmo que as unidades móveis estejam em trânsito, mudando de rede constantemente.
- **Recuperação e tolerância a falhas:** as falhas nos bancos de dados móveis podem ser categorizadas em falhas locais, falhas de mídia, falhas de transação e falhas de comunicação. As falhas de transação são as mais frequentes, principalmente durante o processo de *handoff*.
- **Serviços baseados na localização:** as informações no cache da unidade móvel podem se tornar inconsistentes à medida em que o cliente se move pela rede. Desta forma, as consultas dependentes de localização devem ser atualizadas frequentemente, de modo a manter o cache sempre atualizado.
- **Divisão do trabalho:** o ambiente móvel possui características que acabam provocando uma mudança na divisão do trabalho no processamento de consultas. Em alguns casos, o cliente precisa funcionar independentemente do servidor.
- **Segurança:** os dados móveis são mais vulneráveis a perdas e inconsistência. As soluções para este ambiente devem gerenciar o acesso a dados críticos e ser capazes de compensar sua perda.

Algumas destas características são explicadas com mais detalhes nos próximos tópicos.

Gerenciamento de transações

Transações são unidades lógicas de processamento de banco de dados que incluem uma ou mais operações de acesso à base (inserção, exclusão, modificação e consulta). Pelo fato de várias transações poderem ser executadas concorrentemente no mesmo banco de dados, este passa a estar sujeito a erros como perda de consistência dos dados [8].

Em sistemas de bancos de dados tradicionais, as transações são atômicas, consistentes, isoladas e duráveis (propriedades ACID). O sistema gerenciador do banco de dados deve controlar a execução concorrente de transações para assegurar o estado consistente do banco. Ainda assim, é possível que haja conflito entre transações concorrentes; neste caso, são aplicadas técnicas de serialização e agendamento de transações [7].

Em aplicações móveis, as propriedades citadas acima se mostram restritivas e pouco adequadas, por não oferecerem suporte à desconexão e a operações parciais. Neste contexto, as transações não são completamente gerenciadas pelo sistema, é o deslocamento de uma unidade móvel que controla a execução da transação. Elas podem ser executadas a partir de diferentes servidores e podem acessar dados que estão em constante mudança de lugar, sendo assim dependente da localização deles [10].

A mobilidade resulta em transações que acessam sistemas heterogêneos de informações. O ambiente de comunicação sem fio, por estar sujeito à sobrecarga de dados e desconexões frequentes, faz com que as transações demorem mais para serem executadas, sendo mais

propensas a erros. Por esse motivo, vários refinamentos de transações são necessários, como o uso de sessões e o tratamento de transações como um conjunto de transações parciais, para evitar transações demasiadamente demoradas [10]. Se uma transação for interrompida por uma desconexão, na reconexão, seus efeitos devem ser incorporados ao banco de dados, garantindo seu sucesso.

Processamento de consultas

Consultas realizadas em bancos de dados móveis são mais complexas que as consultas feitas em bancos de dados centralizados. No ambiente móvel, inúmeros fatores podem afetar o desempenho dessas consultas, como por exemplo, a largura de banda, o consumo de energia, a mobilidade, entre outros. Para que a consulta seja realizada com sucesso nesse contexto, é necessário que a localização da unidade móvel seja conhecida [10].

Recuperação de falhas

O processo de recuperação é responsável por preservar a consistência do banco de dados após falhas do sistema, de transações ou dos meios de comunicação. Esse processo detecta falhas e restaura o banco de dados para um estado consistente antes da ocorrência da falha [7].

Em sistemas distribuídos, são utilizados pontos de recuperação, ou *checkpoints*, como base para a recuperação de falhas. Em caso de erro, a aplicação usa o último ponto de restauração salvo para reiniciar sua execução. A unidade móvel é a responsável por criar *checkpoints* em caso de mudança de célula de cobertura ou desconexão, portanto ela deve estar sempre ciente de sua localização e de quando será a próxima desconexão.

2.3.3 Tecnologias de Sincronização

Sincronização no contexto da computação móvel é o estabelecimento de equivalência entre dois conjuntos de dados, entre cliente e servidor, após alguma modificação nos registros armazenados. Esta sincronização é feita através da replicação de dados. Existem dois tipos de sincronização [7]:

- **Unidirecional**: os dados são transmitidos em um único sentido, podendo partir tanto do servidor para o cliente móvel quanto do cliente móvel para o servidor. Quando parte do cliente móvel, todas as alterações são submetidas ao servidor, que as gerencia e resolve os eventuais conflitos. Quando parte do servidor, o cliente móvel recebe os dados atualizados do banco de dados central.
- **Bidirecional**: a transmissão de dados ocorre nos dois sentidos, partindo tanto do cliente quanto do servidor. Neste caso, o cliente envia os dados modificados para o servidor, que os identifica e trata os conflitos existentes e retorna os dados corretos ao cliente. No final, ambos os bancos possuem os mesmos dados.

O fluxo de trabalho durante o processo de sincronização é definido por um protocolo que deve dar suporte à identificação de registros e resolução de conflitos. Definir este protocolo é uma tarefa difícil, pois existem cada vez mais tecnologias de sincronização não-interoperáveis. A maioria das soluções desenvolvidas para esta situação são proprietárias

múltiplas e não satisfazem todo o espectro de funcionalidades e o suporte requerido aos dispositivos [7].

Um protocolo precisa ser interoperável, sincronizando dados de qualquer rede com qualquer dispositivo móvel. A complexidade e o custo de tal protocolo seriam elevados, e esta é a razão pela qual muitas organizações não se dispõem a criar uma solução assim.

Pontos-chave no projeto de protocolos de sincronização são a capacidade de identificação das condições do ambiente, adaptabilidade do modo de apresentação das informações e continuidade da prestação do serviço ao longo das mudanças entre diferentes redes sem fio. As características intrínsecas à computação móvel também dificultam a implementação de um protocolo, que deve levar em consideração a alta latência da rede, a largura de banda limitada, o alto custo de pacotes e a baixa confiabilidade de dados e conectividade.

2.4 Serviços *Web RESTful*

A evolução tecnológica e a industrialização do desenvolvimento de *softwares* levou a uma heterogeneidade de tecnologias e à necessidade de interação entre elas. Para que as diferentes tecnologias se comuniquem é necessário definir uma interface comum de comunicação, especialmente para sistemas distribuídos.

A arquitetura orientada a serviços tem entre seus propósitos o objetivo de padronizar a disponibilidade de operações computacionais entre clientes e servidores de sistemas distribuídos. Esta padronização se dá pela publicação dos componentes computacionais em recursos denominados serviços [11].

O uso de uma interface definida para comunicação entre os diferentes componentes de um sistema distribuído permite que exista uma separação de conceitos, ou seja, uma modularização do sistema, ideal para o desenvolvimento e evolução independente de cada componente.

Uma das tecnologias usadas para se implementar um estilo de arquitetura orientada a serviços é denominada *web service* [4]. Esta tecnologia consiste de uma gama de padrões que gerenciam os serviços através da *web*, entre eles o padrão *Representational State Transfer* (REST).

REST compreende um estilo de arquitetura híbrido derivado de vários outros estilos baseados em rede e combinado com restrições adicionais que definem uma interface uniforme [9]. Por convenção, são denominados *RESTful* os componentes que seguem este estilo e são projetados com todas as restrições bem definidas.

As principais restrições para que um sistema siga este modelo são apresentadas em seguida.

2.4.1 Distribuição Cliente-Servidor

Uma das restrições deste estilo de arquitetura é a distribuição dos componentes entre clientes e servidores. Seguindo o princípio da separação de preocupações, a responsabilidade dos clientes (consumidores do serviço) deve estar separada das responsabilidades dos servidores. Por exemplo, a preocupação de um cliente pode se dar sobre a interface do usuário, enquanto a responsabilidade do servidor recai sobre o armazenamento dos dados. Assim, a interface provida pelo *web service* não está relacionada ao formato que os

dados serão apresentados para o usuário, mas à representação do conteúdo da informação armazenada.

Isto permite que os componentes sejam mais facilmente mantidos e melhora as chances de escalabilidade do sistema para um maior número de usuários. Além de facilitar o reúso dos componentes em ambientes heterogêneos, onde é importante suportar diversos tipos de consumidores distintos.

2.4.2 *Stateless*

Outra restrição do REST é o fato de que o servidor não deve guardar estados na interação com os clientes, ou seja, cada interação com a interface provida pelo serviço deve possuir todos os dados necessários para que a requisição seja entendida [9]. Como consequência, qualquer informação do estado da sessão deve ser armazenada e gerenciada nos clientes.

Esta restrição é uma restrição arquitetural que permite que os serviços disponibilizados tenham uma maior escalabilidade, confiabilidade e visibilidade [9]. O fator escalabilidade é aumentado já que não é necessário armazenar informações sobre cada cliente no servidor. A melhora da confiabilidade se dá pela facilidade que se provê para que o sistema se recupere de falhas. Finalmente, a visibilidade também cresce já que é necessário apenas a observação de uma interação com o serviço para que a requisição seja determinada.

2.4.3 *Cache*

Os serviços *RESTful* devem ser projetados para produzir metadados de controle de *cache* para que o cliente saiba se o resultado da requisição feita pode ser armazenado localmente de forma temporária. A implementação deste armazenamento é responsabilidade de cada cliente, assim como a decisão de utilizá-lo ou não.

Esta restrição tem o objetivo de tornar a interação com o serviço mais eficiente, evitando o envio e processamento de requisições e respostas repetidas. No entanto, se os dados não forem checados e atualizados regularmente, a informação armazenada pode se tornar obsoleta. Portanto, a confiabilidade é reduzida caso o sistema não seja construído levando em consideração as características intrínsecas às informações representadas.

2.4.4 Interface Uniforme

Os consumidores e os servidores de um serviço devem concordar em um contrato para comunicação. Para que sejam considerados componentes *RESTful*, os serviços devem ser projetados para que sejam acessados através de métodos, tipos de dados e uma sintaxe de identificação de recurso padronizados para vários serviços e consumidores.

Várias outras restrições arquiteturais estão relacionadas a esta. Para que se obtenha uma interface uniforme é necessário definir um formato para a identificação de recursos, a representação destes recursos, a criação de mensagens auto-explicativas e o uso do conceito de *Hypermedia as the Engine of Application State* (HATEOAS) [2].

Identificação de recursos

Um recurso é a principal abstração de uma informação no REST. Um recurso pode ser um documento, imagem, um conjunto de outros recursos, entre outros dados. Esta abstração pode ser reconhecida como qualquer outra informação que pode ser nomeada [9], uma definição semelhante à abstração de objetos no paradigma de orientação à objetos.

O identificador de um recurso é um endereço único que representa o recurso real que o serviço expõe. A padronização exigida pelo estilo não define quais identificadores podem ser utilizados, mas sim a sintaxe de utilização destes identificadores. A sintaxe mais comum para este propósito é a própria sintaxe de identificação de recursos utilizada na *web*, a *Uniform Resource Identifier* (URI) [3].

Representação de recursos

Os serviços REST manipulam os recursos e, através da interface de comunicação, transferem representações destes aos clientes. Este mecanismo permite que os consumidores dos serviços sejam informados dos estados dos respectivos recursos.

Uma representação consiste de dados e metadados que descrevem o recurso seguindo uma sintaxe de chave-valor [9]. Por exemplo, duas sintaxes de representação de recursos bem difundidas em serviços RESTful são a do formato *JavaScript Object Notation* (JSON) e a do *eXtensible Markup Language* (XML).

Mensagens auto-explicativas

As respostas que o serviço provê ao consumidor deve conter todos os metadados que permitam que este consiga interpretar corretamente seu conteúdo. Os metadados de controle podem também definir o propósito da mensagem entre os componentes, como a ação requisitada ou o significado da resposta enviada para o consumidor.

Um terceiro uso para os metadados inclui o de alterar alguns comportamentos padronizados da interação entre os componentes. Por exemplo, pode-se usar uma informação adicional que exerça controle sobre o estado do *caching* do recurso.

HATEOAS

Um serviço REST deve retornar para os consumidores todas as informações necessárias para que este seja capaz de navegar através de todos os outros recursos relacionados. Este é um dos diferenciais de um serviço baseado no estilo de arquitetura REST para um serviço baseado em outros padrões [2].

A navegação entre os recursos pode ser feita através de *links* para os recursos relacionados incluídos como metadados na resposta do serviço.

2.4.5 Arquitetura em Camadas

Além das restrições já mencionadas, para que se considere que um sistema segue o modelo REST, ele deve ser disposto com uma organização arquitetada em componentes que interagem através de módulos que podem ser facilmente alterados, adicionados, removidos ou reordenados sem que os outros módulos precisem ser modificados para isto.

Cada componente modular representa uma camada da arquitetura do sistema e, para que a facilidade de substituição dos módulos seja possível, cada camada deve se comunicar apenas com as camadas vizinhas utilizando uma interface bem definida.

Esta restrição permite que se adicionem camadas intermediárias entre os serviços e os consumidores finais, melhorando a escalabilidade do sistema sem comprometer o nível de complexidade de manutenção e de forma transparente ao cliente final.

Em sistemas *RESTful* que operam na *web*, é muito comum utilizar-se das facilidades providas por um modelo arquitetado em camadas para tratar o balanceamento de carga sem que o cliente saiba ou que o serviço provedor aumente sua complexidade. A Figura 2.4 é um exemplo deste caso. Nela, o consumidor não possui conhecimento de que existe um balanceador de carga interceptando o tráfego de requisições e respostas. Assim, o fato de que existem vários servidores distintos que podem responder às transações é irrelevante do ponto de vista deste consumidor.

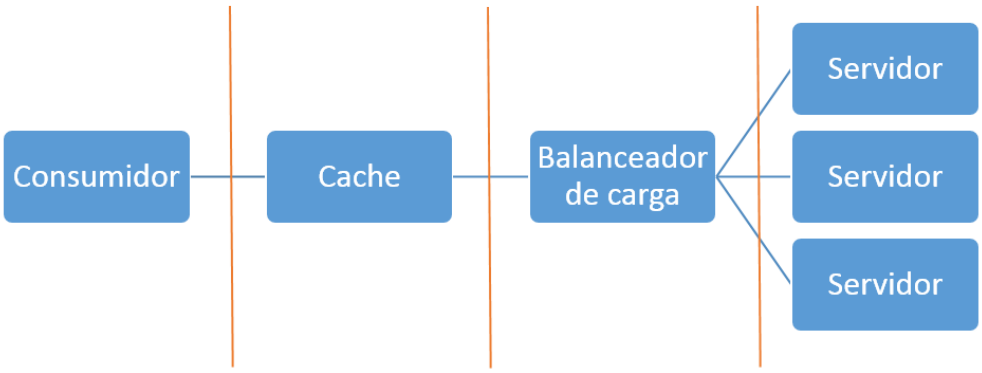

Figura 2.4: Demonstração de uma Arquitetura em Camadas de um Serviço *RESTful*.

2.5 Sistema Operacional *Android*

O *Android* é um pacote de ferramentas de *software* de código aberto associado a um sistema operacional baseado em Unix projetado para dispositivos móveis [5]. O projeto, que foi disponibilizado pela Google, é mantido por um aglomerado de grandes companhias chamado *Open Handset Alliance* (OHA), do qual a própria Google participa.

O sistema iniciou competindo com vários outros concorrentes que já estavam disponíveis no mercado, como a plataforma Symbian, iPhone, Windows Mobile, BlackBerry, Java Mobile Edition, entre outros. No entanto, obteve grande sucesso desde o seu lançamento e ocupa uma fatia do mercado de 82.8% no segundo semestre de 2015 [1].

2.5.1 Plataforma

A plataforma de desenvolvimento *Android* é voltada para a criação de aplicativos, ou *apps*, projetados para serem executados em dispositivos móveis com tela sensível ao toque como *smartphones* e *tablets*, mas as recentes atualizações desta plataforma tornaram

possível sua utilização em uma variada gama de aparelhos incluindo televisões, carros e relógios de pulso.

O *Android* oferece vários recursos para o desenvolvimento de aplicativos utilizando a linguagem Java, mas os componentes centrais do sistema foram criados usando a linguagem C e C++ e é viável criar módulos também nestas linguagens para que sejam executados de forma mais otimizada, quando a preocupação maior é com o desempenho.

O código Java das aplicações é compilado para uma máquina virtual específica, projetada para dispositivos móveis. Até a versão 4.3 do *Android*, a máquina virtual disponível no sistema operacional tinha o nome de Dalvik e era otimizada para usar pouca memória. A partir da versão 4.4 do sistema, uma segunda máquina virtual, denominada *Android Runtime* (ART), está disponível nos aparelhos. Esta nova máquina virtual destina-se a substituir a Dalvik e possui melhorias sobre sua antecessora, como o recurso de compilação antecipada, um coletor de lixo mais otimizado e um melhor suporte à depuração do código.

Durante a compilação, o código é empacotado em um arquivo *Android package* (APK), que é um arquivo compactado com extensão ".apk". Este arquivo contém todo o conteúdo do aplicativo, incluindo o código compilado e os recursos necessários como imagens e arquivos de configuração. Quando um aplicativo é publicado na loja de aplicativos do Google, chamada *Google Play*, este é o arquivo baixado pelo usuário para efetuar a instalação.

Para assegurar a segurança do sistema, o *Android* implementa o princípio de menor privilégio, ou seja, cada aplicativo, por padrão, só possui acesso aos componentes necessários para que seu trabalho seja possível e nada mais [18]. Por isso, para que um aplicativo faça uso de certos recursos do sistema, o *Android* disponibiliza uma estrutura de permissões. Por exemplo, para que o aplicativo interaja com os contatos do usuário, com mensagens SMS, com o cartão de memória, com a câmera, ou diversos outros recursos, o usuário do aplicativo deve explicitamente acionar uma permissão.

A evolução dos dispositivos que operam com o sistema operacional *Android* levou os aparelhos a se tornarem bastante versáteis, principalmente pela adição de hardware específico que é especialmente útil para a computação móvel. O sistema operacional oferece suporte a uma gama de recursos que, além dos já citados, inclui também: telefonia GSM, conexão *bluetooth*, conexão com redes 3G, 4G e WiFi, GPS, bússola e acelerômetros, entre outros.

2.5.2 Arquitetura

O sistema *Android* engloba uma série de componentes que inclui não apenas o núcleo do sistema operacional, mas também outros elementos que, em conjunto, formam uma arquitetura em camadas, conforme apresenta a Figura 2.5. As camadas são dispostas como uma pilha em que a camada superior utiliza os recursos da camada logo abaixo.

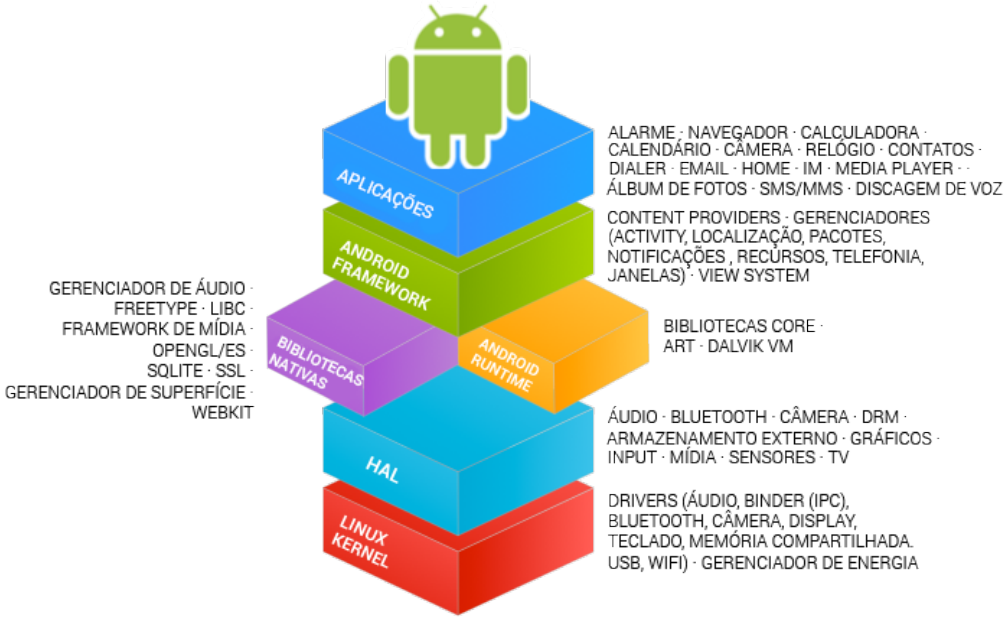

Figura 2.5: Detalhes da arquitetura do *Android* [1].

Cada uma das camadas provê um nível de abstração mais alto com o intuito de encapsular os detalhes desnecessários da implementação das camadas abaixo. Além do encapsulamento, a arquitetura em camadas oferece maior segurança para o sistema como um todo, já que os recursos do sistema só podem ser utilizados através de uma API específica e oferece uma redução da complexidade das camadas superiores, sem comprometer as funcionalidades que estas podem oferecer.

A camada mais abaixo compreende os componentes do *kernel* do Linux utilizado no *Android*. A responsabilidade desta camada é gerenciar os serviços de mais baixo nível do sistema: os processos, a memória, permissões de leitura e/ou escrita nos arquivos, etc. Este *kernel* é a base do sistema multiusuário no qual cada aplicativo é um usuário diferente. Por padrão, cada *app* é executado em seu próprio processo Linux.

A camada logo acima é a *Hardware Abstraction Layer* (HAL), que oferece uma interface para comunicação com os componentes que são específicos do hardware de cada dispositivo. Isto inclui os componentes responsáveis por interagir com áudio, *bluetooth*, câmera, cartão de memória, sensores diversos, entre outros recursos.

Duas camadas fazem a comunicação com a HAL: a camada de bibliotecas e a camada de *runtime*. A primeira contém os componentes das APIs desenvolvidas na linguagem de programação C ou C++ responsáveis por oferecer suporte de mais alto nível aos recursos de hardware do dispositivo, isto inclui a API de persistência de dados do SQLite e a API de renderização de imagens do OpenGL. Já a camada de *runtime* é responsável por oferecer a API Java para o desenvolvimento das aplicações, bem como a máquina virtual que executa o código desenvolvido.

A próxima camada contém os componentes do *framework* de aplicações, que apresentam o conjunto de serviços que formam o ambiente no qual os aplicativos são executados

[1] Fonte: http://source.android.com/source/index.html

e gerenciados. Alguns dos serviços incluem: o gerenciador de *Activities*, que controla o ciclo de vida do *app*, os *Content Providers*, que permitem que diferentes *apps* ou diferentes componentes de um mesmo *app* se comuniquem entre si, o sistema de *Views*, que provê um conjunto de componentes para a criação de interfaces de usuário, etc.

Finalmente, na camada mais acima estão os aplicativos instalados no aparelho. Isto inclui os *apps* nativos - como a Calculadora e o Alarme - e os instalados pelo usuário - como o próprio RockDroid. A não distinção entre os aplicativos nativos e os aplicativos de terceiros permite que qualquer serviço provido nativamente possa ser substituído facilmente para que seja provido por um *app* instalado pelo usuário.

2.5.3 Kit de Desenvolvimento

O Kit de desenvolvimento de *software* do *Android*, *Android* SDK, provê as diversas ferramentas necessárias para se desenvolver um aplicativo *Android*. Dentre essas ferramentas, estão um depurador, as diferentes versões da plataforma *Android*, um emulador (para testes sem necessidade de um dispositivo físico), códigos fonte de exemplo, entre outros programas essenciais para um desenvolvedor.

O Google disponibiliza ainda um *Integrated development environment* (IDE) oficial para programar aplicativos *Android*. Denominado *Android Studio*, esta IDE serve para integrar as ferramentas do SDK, facilitar a navegalibidade pelo código fonte e, consequentemente, aumentar a rapidez de desenvolvimento dos aplicativos.

Além de possuir integração com as ferramentas do SDK, o *Android Studio* possui integração com outras ferramentas para facilitar o desenvolvimento. Algumas ferramentas incluem: o *Gradle*, um gerenciador de dependências, o *Lint*, um verificador estático de código, o *ProGuard*, um verificador de código, entre outras. Esta IDE também possui um editor de *layout*, com suporte à pré-visualização dinâmica, para criar interfaces de usuário.

2.5.4 Componentes Principais

Um componente de um aplicativo é considerado como qualquer elemento do aplicativo que pode servir como de ponto de partida do *app*. Nem todos os componentes são pontos de partida e alguns possuem dependências com outros, mas cada um é uma entidade com um papel específico na aplicação.

Pode-se categorizar os componentes dos aplicativos *Android* em dois grupos: componentes principais e componentes adicionais. O grande diferencial entre estas categorias é que os componentes adicionais, diferentemente dos principais, não foram projetados para servir como ponto de início do aplicativo. A responsabilidade atribuída aos componentes principais envolve funcionalidades de maior escopo e complexidade enquanto os demais componentes servem de suporte para realizá-las.

A Figura 2.6 apresenta a hierarquia completa de todos os componentes. Ao todo, existem quatro tipos distintos de componentes principais: a *Activity*, componente de interface com o usuário, o *Service*, utilizado para tarefas em segundo plano, o *Content Provider*, que gerencia os dados, e o *Broadcast Receiver*, um transmissor de mensagens assíncronas entre os demais componentes. Mais detalhes sobre estes componentes são dados em seguida.

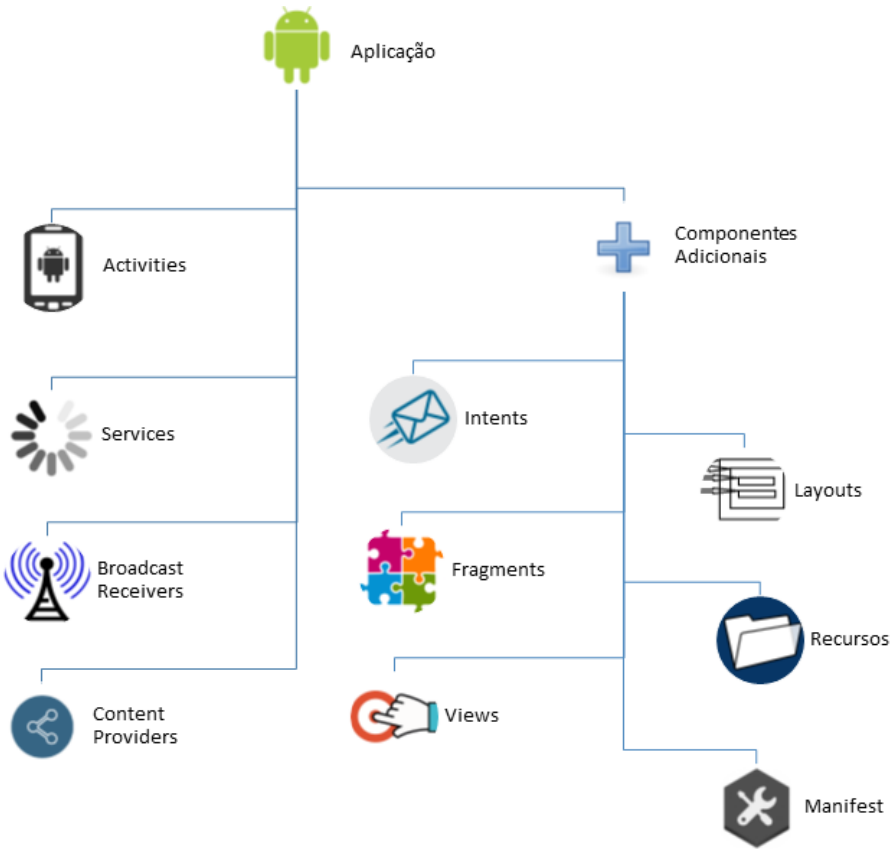

Figura 2.6: Hierarquia dos Componentes de uma Aplicação *Android*.

Activities

Uma *Activity* representa uma tela única com uma interface de usuário. Este é o componente responsável por tratar a interação do usuário com a tela do dispositivo. Se um aplicativo possui mais de uma *Activity*, então uma delas deve ser configurada como aquela que será apresentada quando o aplicativo for aberto.

Por exemplo, uma *Activity* em um aplicativo de coleta de dados geológicos como o Rockdroid pode apresentar a lista de rochas coletadas, outra *Activity* pode apresentar um formulário para entrada dos dados de uma nova rocha, etc.

Services

Um *Service* é um componente que não possui interface de usuário, isto é, ele é executado em segundo plano. Seu objetivo é executar operações que exigem um tempo longo para serem completadas.

É importante salientar que não necessariamente um *Service* é executado em uma *thread* separada, por padrão este componente, quando iniciado, executa na mesma *thread* de quem o chamou. Ou seja, se iniciado na *thread* principal, um *Service* pode paralisar a interação do usuário com o componente que esteja em primeiro plano.

A utilidade de um *Service* está no fato de que seu ciclo de vida não está atrelado ao ciclo da tela apresentada para o usuário, isto implica que este componente pode continuar em execução mesmo que o usuário não esteja mais interagindo com o aplicativo específico.

Um exemplo de uso de um *Service* pode ser o de buscar dados na rede sem impedir a navegação do usuário no aplicativo, ou executar uma música enquanto o usuário está com outro aplicativo aberto em primeiro plano.

Content providers

Um *Content Provider* é um componente responsável por gerenciar dados de um aplicativo que podem ser compartilhados com outros. Este componente define uma interface padronizada para que outros *apps* possam lidar com os dados do aplicativo em questão. Isto inclui o tratamento de acesso e de persistência desses dados.

Os próprios aplicativos nativos do *Android* disponibilizam alguns *Content Providers* que podem ser utilizados por quaisquer outros *apps*, desde que estes possuam as permissões necessárias. Por exemplo, o aplicativo de Contatos permite que outros aplicativos recuperem e atualizem informações sobre um contato específico da agenda de contatos do dispositivo.

Também é possível utilizar este componente para compartilhar dados internamente ao próprio aplicativo, apesar deste uso ser menos comum, já que sua implementação pode se tornar uma complexidade desnecessária para a maioria dos casos.

Broadcast receivers

Um *Broadcast Receiver* é o componente ativado quando uma mensagem é transmitida pelo sistema ou por outros aplicativos a todos aqueles que estiverem ouvindo. Ao interceptar uma mensagem deste tipo, este componente é capaz de iniciar o tratamento apropriado para a mensagem recebida. *Broadcast Receivers* podem executar um aplicativo mesmo que este não estivesse sendo executado anteriormente. Estes componentes não possuem interface de usuário, no entanto, eles são capazes de criar uma notificação na barra de *status* do dispositivo caso seja necessário alertar o usuário sobre um determinado evento. Um exemplo de uso deste componente seria um aplicativo diminuir o uso da rede quando recebe a mensagem do sistema de que a bateria do aparelho está em um nível baixo.

2.5.5 Componentes Adicionais

Um aspecto único do sistema *Android* é que ele foi projetado com o foco em estimular o reúso dos componentes entre diferentes aplicativos. Assim, uma parte importante de sua implementação foi permitir que qualquer aplicativo pudesse iniciar um componente de um outro aplicativo.

Por exemplo, se para o seu aplicativo é necessário capturar uma imagem com a câmera do dispositivo, como provavelmente já existe outro aplicativo instalado que faz este serviço, ao invés de codificar uma *Activity* que implemente este processo, basta pedir para o sistema *Android* que inicie o *app* responsável por isto.

Para possibilitar este tipo de característica dos aplicativos, facilitar a construção dos componentes, simplificar a lógica por trás dos recursos que o aplicativo oferece e possibili-

tar a conexão entre os componentes principais, existem os componentes adicionais. Estes também estão listados na Figura 2.6. Abaixo está uma descrição detalhada de cada um deles:

- ***Intents*:** uma *Intent* representa uma mensagem assíncrona enviada para ativar *Activities*, *Services* ou *Broadcast Receivers*. É usando este componente que se inicia uma *Activity*, seja para navegação dentro do seu próprio aplicativo ou para utilizar um serviço de um outro *app*. É possível passar dados nesta mensagem para que o componente chamado saiba tratar a requisição apropriadamente.
- ***Fragments*:** um *Fragment* representa uma parte de uma *Activity*. Este componente foi introduzido na versão 3.0 do *Android*, a versão que adicionou suporte a *tablets*, mas é suportado em versões anteriores também. Seu objetivo principal foi possibilitar que o conteúdo de uma *Activity* pudesse ser alterado de acordo com o tamanho da tela do dispositivo. Multiplos *Fragments* poderiam ser dispostos em uma mesma *Activity* caso a tela fosse grande o suficiente, ou poderiam ser dispostos um a um, caso a tela fosse menor. Este modo de desenvolver também possibilitou a criação de aplicativos cujas *Activities* fossem implementadas de maneira mais modular.
- ***Views*:** uma *View* representa o elemento mais básico de uma interface de usuário. Este elemento é responsável pelo desenho e pelo tratamento de eventos da estrutura desta interface. Estes componentes podem ser adicionados em um arquivo XML para a criação da interface do usuário de forma declarativa, mas também podem ser gerenciados dinamicamente através de comandos da API Java do *Android*. Exemplos de *Views* incluem botões, campos de texto, etc.
- ***Layouts*:** um *Layout* é um *container* para *Views* que gerencia a sua disposição na tela. Estes componentes são particularmente úteis para o ajuste da interface do usuário criada declarativamente a partir de um arquivo XML. O *Android* disponibiliza uma série de diferentes *Layouts*, cada um com suas propriedades para dispor os elementos usando uma configuração distinta. Por exemplo, um componente *LinearLayout* dispõe os elementos em sequência, seja verticalmente ou horizontalmente. Já um componente *RelativeLayout* dispõe os elementos usando um elemento principal como referência de posicionamento.
- **Recursos:** um aplicativo *Android* possui, além do código fonte que implementa a lógica de negócio, uma série de recursos que o código utiliza. Estes recursos incluem imagens estáticas, definição de cores, temas, estilos, animações e *layouts* das interfaces de usuário, arquivos de *Strings* utilizados para internacionalização do aplicativo, entre outros arquivos de configuração. É possível ainda armazenar arquivos quaisquer que seu aplicativo pode necessitar.
- ***Manifest*:** o *AndroidManifest.xml* é o principal arquivo de configuração do aplicativo. Neste arquivo são declaradas as permissões necessárias para a execução das tarefas, bem como os componentes - *Activities*, *Services*, *Broadcast Receivers* e *Content Providers* - que o aplicativo possui. Além disso, neste arquivo são declaradas informações gerais sobre o *app*, como a versão mínima do *Android* compatível, a versão atual do aplicativo, entre outras.

2.5.6 Persistência

O sistema *Android* provê várias opções para a persistência de informações dos aplicativos. A solução ideal para cada caso depende de fatores como: se os dados armazenados devem ser exclusivos de uma aplicação ou deverão ser compartilhados e quanto espaço é necessário para armazenar os dados. As opções de armazenamento providas pelo *Android* estão listadas na Tabela 2.1

Tabela 2.1: Descrição das Diferentes Formas de Armazenamento Suportadas pelo *Android*.

Local	**Descrição**
Shared Preferences	Armazena variáveis primitivas de forma privada ou pública em pares de chave-valor.
Armazenamento interno	Armazena dados de forma privada na memória do aparelho.
Armazenamento externo	Armazena dados de forma pública no cartão de memória do aparelho.
SGBD SQLite	Armazena dados estruturados de forma privada em um banco de dados.
Nuvem	Armazena dados em um computador remoto na rede.

As *Shared Preferences* consistem em uma API disponibilizada pelo *Android* para persistência de dados simples, considerados como variáveis primitivas da linguagem Java - *booleans*, *floats*, *ints*, *longs* e *Strings*. Estes dados continuarão armazenados mesmo que o aplicativo não esteja em execução. Esta é uma API de fácil uso e é uma boa opção para uma pequena quantidade de dados. É comum seu uso para armazenar variáveis que devem manter seu valor entre sessões do usuário.

A memória interna do aparelho também pode ser usada para persistir informações. Os dados armazenados serão mantidos enquanto a aplicação estiver instalada no aparelho e, por padrão, serão exclusivos do *app*, ou seja, outras aplicações não poderão recuperar estes dados.

Todo aparelho *Android* possui suporte a um armazenamento externo que também pode ser usado para persistir dados. Este armazenamento pode ser um cartão de memória removível ou uma memória de armazenamento interna. É comum nos aparelhos atuais que esta memória tenha capacidade medida em *Gigabytes*.

Os dados persistidos neste local são públicos e poderão ser lidos, alterados ou removidos por outros aplicativos e pelo próprio usuário ao conectar o aparelho via USB em outro dispositivo, portanto sua disponibilidade deve ser sempre verificada antes do uso. Vale salientar ainda que os dados escritos neste local não são automaticamente removidos quando o usuário remove o aplicativo.

Este espaço de armazenamento deve ser utilizado, portanto, para guardar dados que sejam compartilhados ou que pertençam ao usuário, como músicas ou fotos. No entanto, também é possível utilizá-lo para dados do aplicativo criando um diretório específico para isto.

O *Android* também oferece suporte completo aos bancos de dados SQLite [16]. Esta é a opção ideal para o armazenamento de dados que podem ser estruturados em tabelas

típicas de um banco relacional. As informações persistidas serão privadas ao aplicativo, ou seja, o banco criado por um aplicativo será acessível apenas ao aplicativo que o criou.

Por fim, quando há uma rede disponível, pode-se utilizar os serviços de comunicação em nuvem, como os *web services*, para possibilitar o armazenamento e a recuperação de dados em um servidor remoto. Esta opção é interessante por não se limitar à capacidade de armazenamento local do aparelho, mas ela depende da disponibilidade e da capacidade da conexão.

Capítulo 3

RockDroid

O sistema proposto neste documento é uma aplicação voltada para a coleta de dados geológicos em um ambiente com conectividade limitada e no qual o usuário está em constante movimento. Seu objetivo é auxiliar os pesquisadores do Instituto de Geociências da Universidade de Brasília (IG/UnB) com as anotações em suas pesquisas de campo.

Inicialmente, foi realizada uma pesquisa sobre outros sistemas criados para a coleta de dados geológicos e foi feita uma comparação de suas funcionalidades em relação ao desejado para o contexto específico dos pesquisadores interessados, para verificar a necessidade e a motivação da construção de um novo sistema.

Após esta comparação, foram levantados, em detalhes, os requisitos específicos para o cenário da UnB, através de entrevistas com pesquisadores responsáveis pelos trabalhos em campo. De posse dos requisitos e levando em consideração as críticas obtidas das ferramentas semelhantes, a proposta do sistema foi descrita em termos das funcionalidades necessárias para cumpri-los.

Este capítulo descreve cada um destes passos, na ordem em que foram tomados e apresenta o processo de desenvolvimento escolhido.

3.1 Ferramentas Semelhantes

Para auxiliar na definição do projeto do RockDroid, foram identificados, testados e revisados outros softwares inseridos no mesmo contexto de coleta de dados geológicos, com funcionalidades semelhantes às desejadas para este aplicativo. Os aplicativos escolhidos foram o **Geotools**, o **Strike and Dip**[1] da *Major Forms*, o **FieldMove Clino**[2] da *Midland Valley Exploration Ltd.*, o **Rocklogger**[3] da *RockGecko* e o **Geopaparazzi**[4] da *HydroloGIS S.r.l.*. Eles podem ser baixados gratuitamente pela *Google Play Store*, plataforma do Google onde se concentram os aplicativos *Android*, exceto o **Geotools**, cujas informações foram extraídas a partir de um artigo publicado na revista *Computers & Geosciences* [23]. A escolha dos quatro aplicativos disponíveis na *Google Play Store* foi resultante de uma seleção que buscou, dentre os aplicativos que se encaixavam no

[1] https://play.google.com/store/apps/details?id=com.shopzeus.android.majorforms_1013
[2] https://play.google.com/store/apps/details?id=com.mve.fieldmove.clino
[3] https://play.google.com/store/apps/details?id=com.rockgecko.dips
[4] https://play.google.com/store/apps/details?id=eu.hydrologis.geopaparazzi

contexto desejado, os que tinham maior pontuação dos usuários. Na Figura 3.1, pode-se ver *screenshots* dos aplicativos mencionados.

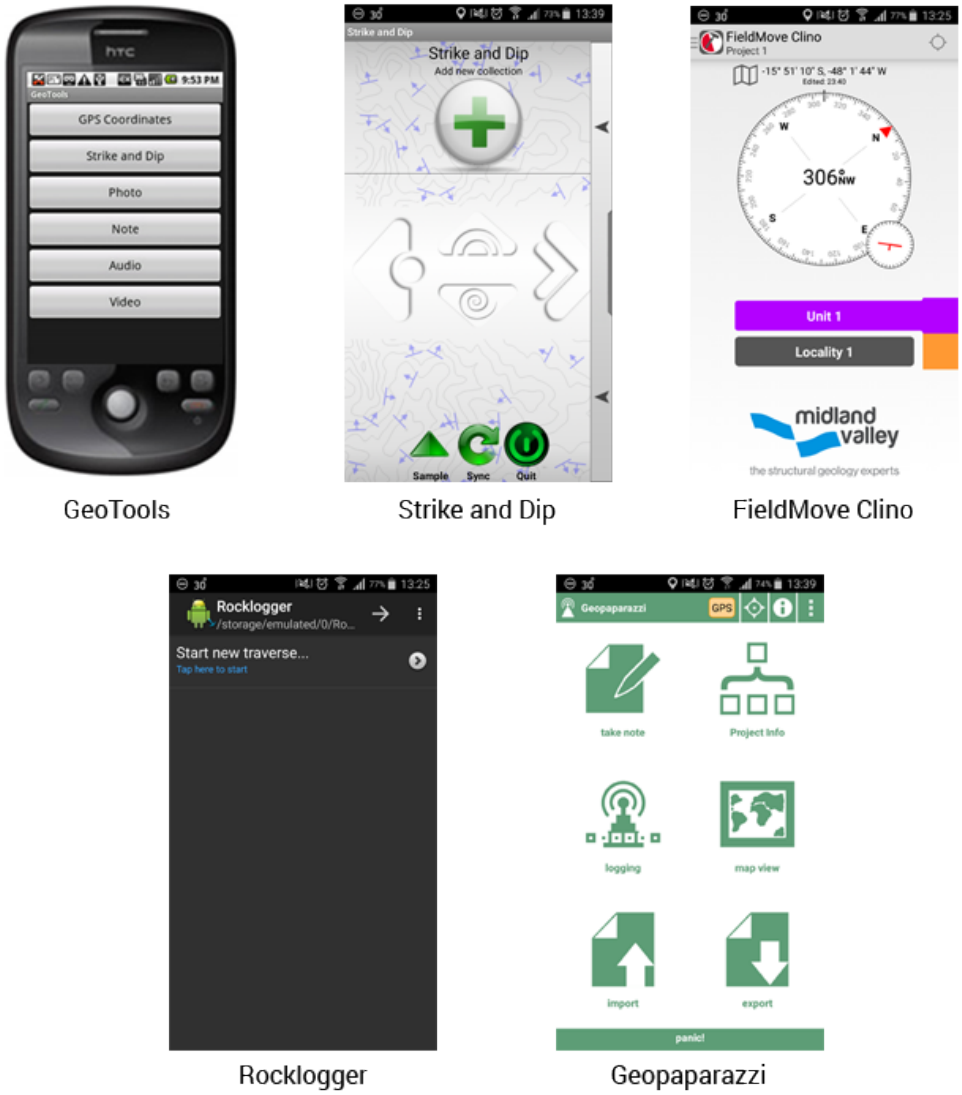

Figura 3.1: Aplicativos Pesquisados.

Os critérios utilizados para a análise dos aplicativos selecionados foram organização dos dados, compartilhamento dos dados, conexão, mapas e usabilidade, descritos a seguir:

- **Organização dos dados:** diz respeito à estrutura que o aplicativo utiliza para armazenar os dados coletados (se são divididos em projetos ou por localização, ou outros).
- **Compartilhamento dos dados:** é a capacidade do aplicativo de compartilhar os dados coletados com outros usuários, seja exportando ou acessando os dados remotamente. O objetivo é identificar a possibilidade de usar o aplicativo quando se está trabalhando em equipe.

- **Conexão:** verifica se o aplicativo é capaz de trabalhar sem conexão com a *Internet*, o que é uma situação muito comum para os geólogos em campo.
- **Mapas:** analisa a capacidade do aplicativo de exibir os dados de uma forma visual, até mesmo quando não há conexão com a *Internet*. É importante para os geólogos poderem trabalhar com mapas quando estão *offline*. Além disso, um dos dados mais importantes a serem coletados é a localização das amostras.
- **Usabilidade:** diz respeito à facilidade de uso do aplicativo e à existência de tutoriais ou opções de ajuda para usuários menos experientes. Também verifica se o usuário possui controle sobre os dados coletados pelo aplicativo (se ele consegue modificar valores que são obtidos automaticamente pelo *software* ou se ele pode colocar seus próprios valores em vez de usar métodos de obtenção automática). Além disso, é observada a capacidade de adicionar campos personalizados.

Os resultados da comparação entre as cinco aplicações citadas podem ser vistos nas tabelas abaixo. Na Tabela 3.1 tem-se a comparação das ferramentas em relação à necessidade de conexão com a *Internet* e aos dados que podem ser coletados. Na Tabela 3.2, tem-se os resultados da comparação quanto ao compartilhamento de dados e aos mapas. Na Tabela 3.3, as ferramentas são comparadas quanto à usabilidade.

Tabela 3.1: Comparação das Ferramentas em Relação à *Internet* e aos Dados Coletados

Aplicativo	**Internet**		**Dados coletados**	
	Ao iniciar	**Para usar**	**Automaticamente**	**Manualmente**
FieldMove Clino	Não	Não	Coordenadas GPS; Direção e mergulho; Bússola; Foto; Data e hora.	Descrições textuais; Cor da rocha; Lineação.
Strike and Dip	Sim	Não	Coordenadas GPS; Direção e mergulho; Foto; Data e hora.	Descrições textuais; Vídeos e gravações; Temperatura; Pressão.
Rocklogger	Não	Não	Coordenadas GPS; Direção e mergulho.	Descrições textuais; Lineação.
Geotools	Não	Não	Coordenadas GPS; Direção e mergulho; Foto; Vídeos e gravações.	Descrições textuais.
GeoPaparazzi	Não	Não	Coordenadas GPS; Foto; Desenhos.	Descrições textuais.

Tabela 3.2: Comparação das Ferramentas em Relação ao Compartilhamento de Dados e às Características dos Mapas

Aplicativo	**Compartilhamento de dados**	**Mapas**		
		Offline	**Stereonet**	**Formato**
FieldMove Clino	Exportar para CSV; Exportar para MVE.	Sim	Pago	*Google Maps* para visões de terreno e de satélite.
Strike and Dip	Sincronização dentro da mesma conta; Criação de site.	Não	Não	*Google Maps* para visões de terreno e de satélite.
Rocklogger	Exportar para CSV; Exportar para KML.	Sim	Pago	*Google Maps* para visões de terreno e de satélite.
Geotools	Exportar para XML.	Não	Não	Não suporta mapas.
GeoPaparazzi	Exportar para KMZ; Exportar para GPX.	Não	Não	*OpenStreetMaps* para visão de terreno; não suporta visão de satélite.

Tabela 3.3: Comparação das Ferramentas em Relação à Usabilidade

Aplicativo	**Usabilidade**	
	Vantagens	**Desvantagens**
FieldMove Clino	Fácil inserção de novos registros.	Interface pouco intuitiva: requer algum treinamento para usar; Não associa foto à rocha.
Strike and Dip	Possibilidade de inclusão de campos personalizados.	Interface pouco intuitiva: difícil de usar e de aprender.
Rocklogger	Possibilidade de criação de tipos de plano personalizados; *Feedback* sobre precisão de medidas.	Não permite edição de dados automaticamente adquiridos com os sensores do dispositivo.
Geotools	Simples de usar.	A falta de campos específicos faz com que o usuário use um texto grande para várias informações.
GeoPaparazzi	Fácil importação e exportação de dados.	Não utiliza os botões físicos do dispositivo.

Pode-se observar, a partir dos dados apresentados, que a capacidade de utilização sem conectividade é essencial para o contexto de coleta de dados em campo e é levada em consideração por todos os aplicativos exceto o *Strike and Dip*. No que se refere à obtenção automática dos dados, a maioria lida com coordenadas recuperadas do GPS do aparelho

e fotos obtidas através da câmera. O *Geotools*, *FieldMove Clino* e o *Strike and Dip* se sobressaem ao obter também a direção e o mergulho das estruturas.

Apenas o *Rocklogger* e o *FieldMove Clino* possuem capacidade de visualização de mapas *offline*, apesar da maioria dos aplicativos oferecer suporte a mapas de alguma forma. No entanto, nenhum dos aplicativos oferece visualização de imagens de satélite sem conexão com a Internet.

A interface com o usuário da maioria dos aplicativos é pouco intuitiva e difícil de aprender a usar inicialmente. Neste quesito, o aplicativo *FieldMove Clino* se destaca, já que sua interface segue um padrão mais recente do sistema *Android*. Mesmo assim, pelo formato de seus botões e pela estruturação da navegação entre as telas, este *app* ainda requer algum treinamento inicial.

O formato do compartilhamento de dados é bastante variado entre os aplicativos. Os *apps Rocklogger* e *GeoPaparazzi* oferecem opções de exportação para formatos que podem ser lidos pelo *software Google Earth*, facilitando o uso e análise dos dados coletados. No entanto, neste tipo de exportação, muitas informações não são repassadas, apenas dados de localização e uma descrição geral da coleta. A exportação de todos os dados coletados, incluindo todas as descrições, só é possível para arquivos no formato CSV ou XML, oferecido pelo *Geotools*, *Rocklogger* e *FieldMove Clino*.

O *Strike and Dip* é o único aplicativo que provê sincronização dos dados dentro da mesma conta, o que implica que existe um repositório central que armazena todos os dados coletados. No entanto, no âmbito da UnB, o acesso a este repositório central é inviável, então um pesquisador do IG só teria acesso aos próprios dados, assim como nos demais *apps*.

A partir destas observações, pode-se concluir que, para os pesquisadores de geologia da UnB, nenhum destes sistemas de coleta de dados atende as necessidades. A criação de um sistema próprio, que tivesse uma boa usabilidade e fornecesse posterior acesso aos dados coletados em um banco de dados central seria imprescindível para que se pudesse de fato substituir as cadernetas de papel por um sistema computadorizado nas pesquisas de campo.

3.2 Elicitação de Requisitos

Para este trabalho, o foco inicial foi o projeto da interface de usuário, que acabou incorporando técnicas que auxiliaram no desenvolvimento do sistema como um todo. Existem vários motivos para justificar a importância dada à interface de usuário [17]: Foco

- Para a maioria dos usuários, o sistema e a interface são a mesma coisa, principalmente se o público-alvo da aplicação não possuir conhecimentos básicos sobre programação, como é o caso do RockDroid, cujos usuários finais são professores, estudantes e pesquisadores da área de geologia.
- A complexidade das interfaces de usuário vem crescendo nos últimos anos, com novas tecnologias e padrões. É preciso seguir a tendência do mercado de forma a atrair a atenção e o interesse do usuário.

- Interfaces de usuário bem elaboradas aumentam a produtividade do usuário, diminuindo o tempo levado para realizar tarefas e os custos de treinamento, suporte e manutenção. Além disso, o sistema se torna mais atrativo e a satisfação dos usuários é maior.

O objetivo de se utilizar este processo de desenvolvimento é criar um software que funcione como esperado, cuja interface seja consistente e o qual seja eficiente, fácil de aprender, confiável e seguro. O primeiro passo para isto é levar em consideração vários aspectos relacionados ao usuário, como seus objetivos, suas tarefas, suas habilidades e suas limitações. Dentre vários modelos de desenvolvimento, o mais adequado foi o modelo centrado no usuário, que pode ser visto na Figura 3.2.

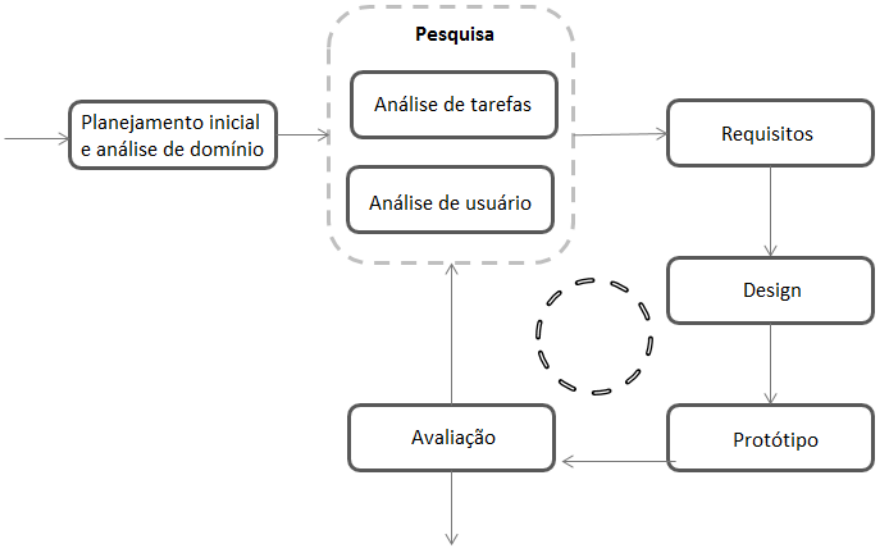

Figura 3.2: Processo de Desenvolvimento Centrado no Usuário, adaptada de [15].

Neste modelo, o primeiro passo é fazer o planejamento inicial e a análise de domínio. Após este estudo inicial, inicia-se a fase de pesquisa, que envolve a análise de tarefas e a análise de usuário. Ao final da pesquisa, espera-se ter definido os requisitos, que servirão de base para o *design* do sistema. O *design* envolve a criação de *sketches* e *wireframes*, que representam a interface das principais telas do sistema. A partir daí, é criado um protótipo e é realizada uma avaliação junto ao usuário. Caso o resultado desta avaliação não seja favorável, deve-se reiniciar o processo a partir da etapa de pesquisa, atentando para pontos que possam ter passado despercebidos e levando em consideração o *feedback* recebido na fase de avaliação. Quando o protótipo estiver bom o suficiente, o processo é encerrado e inicia-se a fase de desenvolvimento. Os conceitos aqui mencionados são explicados ao longo deste capítulo.

3.2.1 Planejamento Inicial

Após o estudo inicial de outras ferramentas inseridas no mesmo contexto do RockDroid, foi feito um cronograma que incluía reuniões semanais com a professora orientadora

deste trabalho e professores do IG/UnB com o objetivo de realizar as etapas de análise de tarefas, usuários e domínio.

3.2.2 Análise de Domínio

A análise de domínio é essencial para os desenvolvedores, principalmente se eles não possuírem conhecimento algum sobre o contexto no qual o software está inserido [17]. É o entendimento do problema, que facilita a comunicação entre o engenheiro de software e os *stakeholders* (pessoas envolvidas no projeto). Neste projeto, o estudo do domínio foi feito com o auxílio dos professores do IG.

3.2.3 Análise de Tarefas

A análise de tarefas é o processo de coletar informações e investigar os procedimentos utilizados pelos usuários para fazer o que eles desejam. Tarefa é apenas um processo executado pelo usuário para atingir um objetivo. É nesta etapa que se investiga quais tarefas o usuário realiza atualmente, como ele as realiza (seja utilizando outro software, seja manualmente) e, por vezes, onde ele as realiza [17].

Neste trabalho, o levantamento de informações sobre as tarefas do usuário foi realizado por meio de entrevistas com professores do IG/UnB. Durante as entrevistas, notou-se a existência de conflitos de interesse entre diferentes usuários. Pode-se dizer que tais conflitos surgiram devido à falta de padrão no processo de coleta de dados realizado por eles, porque da forma como é feito atualmente, os dados coletados dependem dos interesses e das prioridades de cada pesquisador.

3.2.4 Análise de Usuário

A análise de usuário envolve a observação de usuários em seu meio de trabalho, se possível, além de entrevistas e a interação com o usuário durante todo o processo de desenvolvimento. É necessário identificar características do público-alvo que podem influenciar diretamente no desenvolvimento do software, como: idade, gênero, cultura, língua, nível de educação, limitações físicas ou deficiências, experiência com computadores e com seu domínio de trabalho [17].

Para esta pesquisa, o público-alvo é composto por professores e estudantes do IG/UnB. Neste caso, questões como a diferença entre o nível de experiência entre professores e estudantes influenciam diretamente no projeto do aplicativo - um aplicativo voltado para estudantes precisa de formulários mais detalhados e validações mais rígidas, bem como tutoriais e opções de ajuda, enquanto a preferência entre os professores é deixar os formulários mais generalizados, com campos mais abertos e menos estruturados.

No caso do RockDroid, um fator limitante importante para a construção do software tem a ver com a legislação. Um requisito apresentado pelos pesquisadores durante as entrevistas era o suporte a imagens de satélite na visualização do mapa sem necessidade de conexão com a Internet. Para que isto fosse possível, as imagens de uma determinada região deveriam ser obtidas e armazenadas previamente. No entanto, os termos de uso dos provedores de imagens de satélite proíbem claramente este Caso de Uso, o que tornou este requisito inviável do ponto de vista técnico.

3.2.5 Requisitos

Uma vez finalizada a fase de pesquisa, foi iniciada a elaboração dos requisitos. Inicialmente, foi feita a modelagem do processo empregado atualmente e o planejamento do processo a ser implementado com o uso do RockDroid, assim como possíveis funcionalidades e modelos de dados.

O processo atual funciona da seguinte forma: ao chegarem no local da pesquisa, os pesquisadores se dividem em grupos de 2 a 3 pessoas e definem o caminho a ser percorrido. Cada grupo escolhe diferentes pontos de coleta e anota informações relativas às rochas, amostras e estruturas encontradas. Ao final do dia de coleta ou ao retornarem da saída de campo, os dados anotados são digitados em uma planilha do Excel e somente após essa etapa é que eles podem ser analisados ou compartilhados com outros pesquisadores. O modelo desse processo (*AS IS*) encontra-se na Figura 3.3.

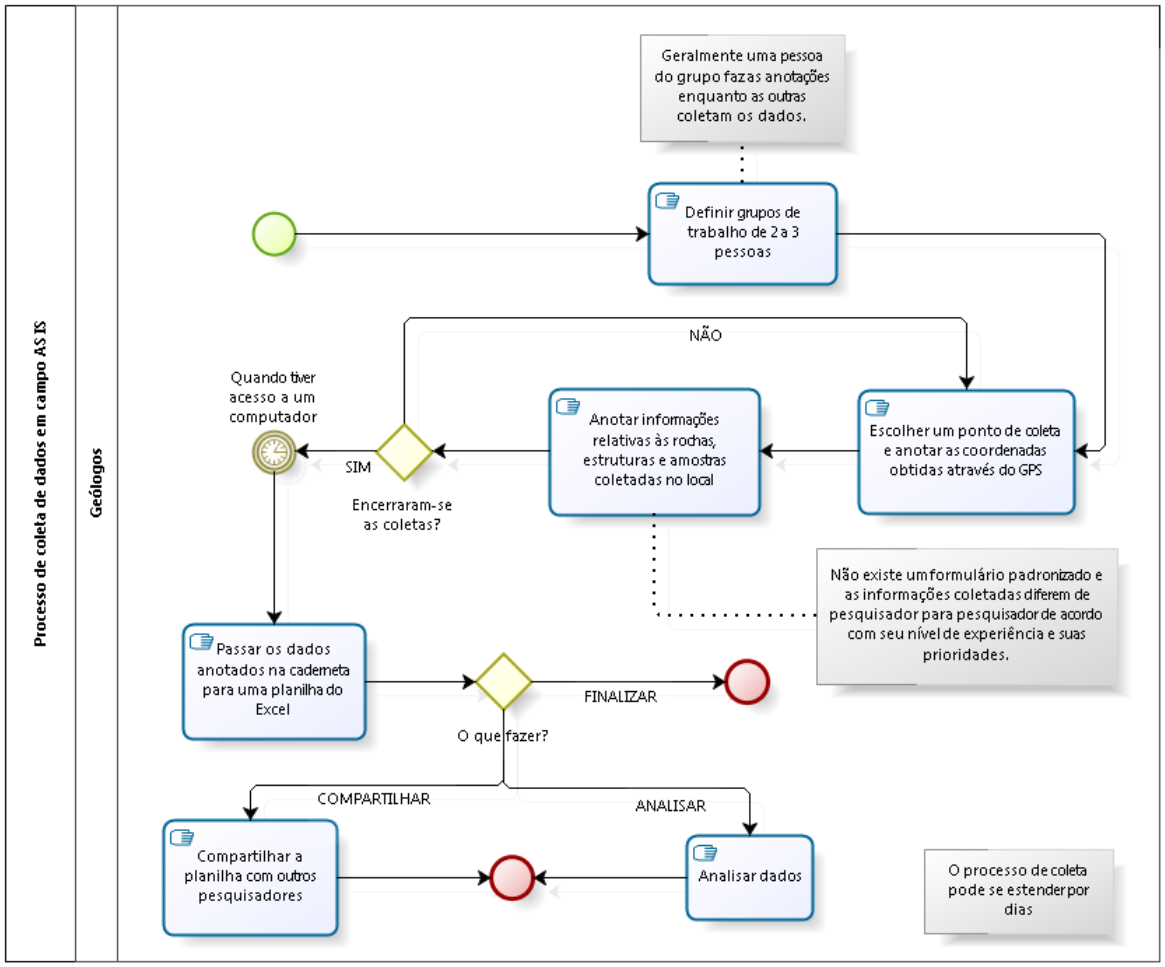

Figura 3.3: Processo Atual de Coleta.

O processo a ser implementado com o uso do RockDroid inicia-se quando o usuário abre o aplicativo. Caso ele possua *Internet*, ele faz seu cadastro no sistema e faz *login*; caso contrário, ele entra como Visitante. Uma vez logado no aplicativo, o usuário pode criar um novo projeto ou escolher um projeto já existente. Na primeira situação, o usuário

cria também uma nova etapa de campo. Já na segunda, a escolha entre criar ou usar uma etapa de campo já existente é do usuário. A partir deste ponto, inicia-se a tarefa de coleta de dados. Uma vez encerrada a coleta, o usuário pode exportar seus dados ou sincronizá-los com o servidor central, desde que tenha *Internet*. O modelo do processo TO BE encontra-se na Figura 3.4.

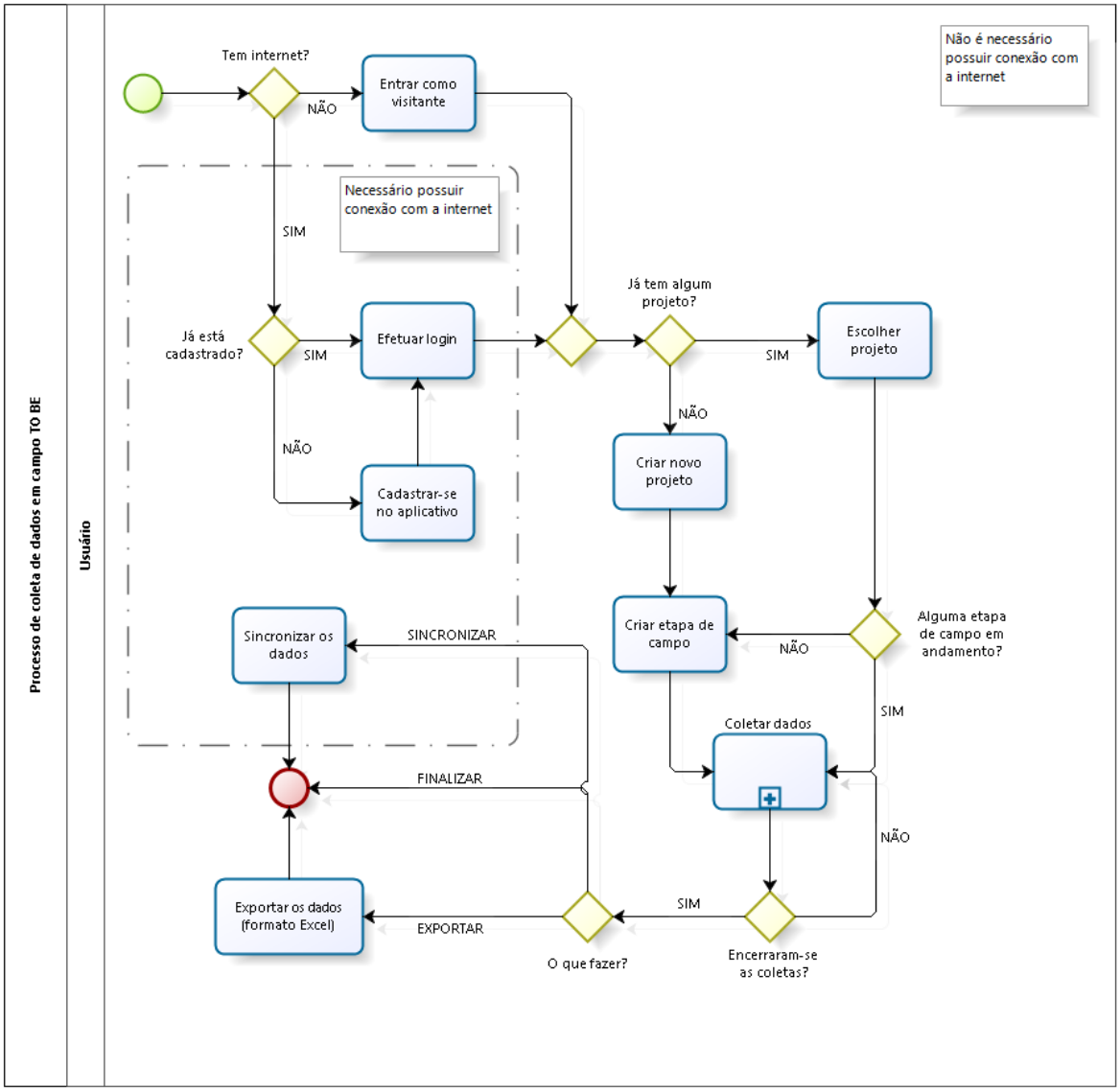

Figura 3.4: Processo de Coleta com o Uso do Aplicativo RockDroid.

A tarefa de coleta de dados, mostrada na Figura 3.4, é um sub-processo que pode ser visto com mais detalhes na Figura 3.5.

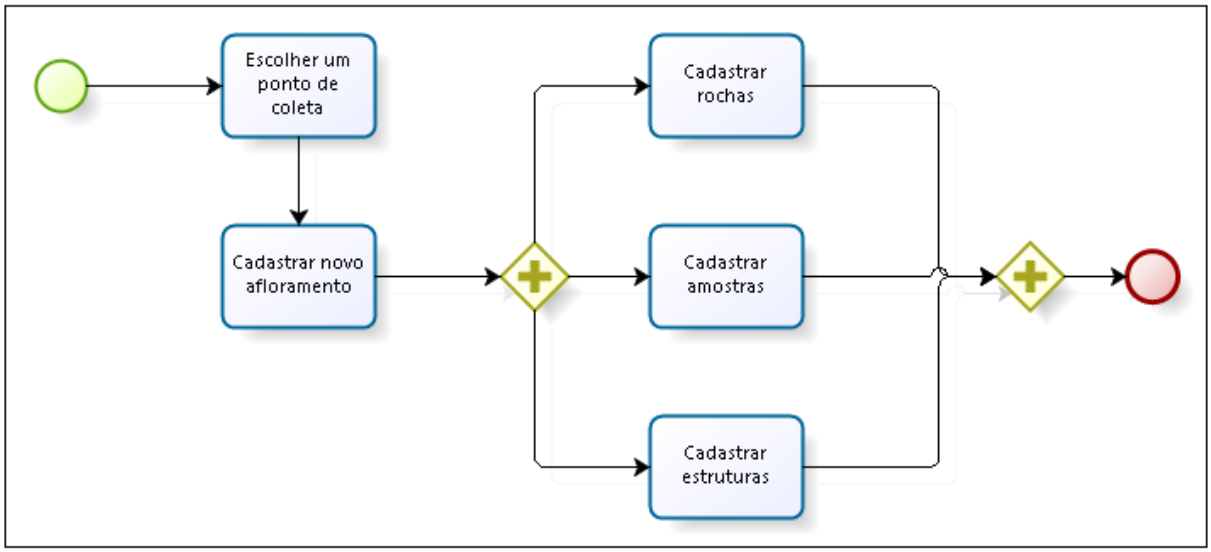

Figura 3.5: Subprocesso de Coleta de Dados Expandido.

O diagrama entidade-relacionamento dos dados encontra-se na Figura 3.6. Nele, é possível ver que foram definidas as seguintes entidades:

- **Usuário:** representa um usuário cadastrado no sistema. Seus atributos são: id, email, nome e senha;
- **Projeto:** representa um projeto criado por um usuário. Seus atributos são: id e nome;
- **Etapa de campo:** um projeto é organizado em várias etapas. Cada etapa possui uma data de início e ocorre em um dado município. Seus atributos são: id, nome, data de início, município e UF;
- **Afloramento:** representa um ponto de coleta em que se encontram rochas e estruturas. Cada afloramento está relacionado com uma determinada etapa de campo. Seus atributos são: id, nome, latitude, longitude, altitude, toponímia, descrição, data de criação e um atributo multivalorado de fotos;
- **Rocha:** representa as medições e descrições de uma rocha. Cada rocha está associada a um afloramento específico. Cada rocha coletada possui um tipo que define seus atributos, por isso, elas são especializadas em rocha sedimentar, ígnea e metamórfica. Os atributos compartilhados por todas as rochas são: id, nome, tipo, mineralogia, textura e um atributo multivalorado de fotos;
- **Rocha Sedimentar:** representa uma rocha constituída de sedimentos. Esta entidade não possui nenhum atributo adicional, apenas os que herda da entidade Rocha;
- **Rocha Ígnea:** representa uma rocha consolidada a partir do magma. Seus atributos específicos são: composição, nomenclatura, tamanho e trama;
- **Rocha Metamórfica:** representa uma rocha resultante da transformação de outras rochas. Seus atributos específicos são: composição, nomenclatura e grau;

- **Estrutura:** representa uma estrutura geológica encontrada em um afloramento. Uma estrutura é o resultado de uma deformação de uma rocha, por isso, toda estrutura está associada a pelo menos uma rocha. Uma rocha também pode conter várias estruturas distintas, por isso há um relacionamento de muitos para muitos entre estas entidades. As estruturas são classificadas em primárias e secundárias. Os atributos compartilhados por ambos os tipos são: id, tipo e um atributo multivalorado de fotos;
- **Estrutura Primária:** representa o resultado de uma deformação originado no processo de formação de uma rocha. Seu atributo específico é: descrição;
- **Estrutura Secundária:** representa o resultado de uma deformação ocorrida posteriormente à formação de uma rocha. Seus atributos específicos são: tipo de plano, fase, mergulho e direção de mergulho;
- **Amostra:** representa um fragmento geológico retirado em campo para análise laboratorial. Cada amostra está relacionada a um afloramento. Seus atributos são: id, nome e um atributo multivalorado de fotos.

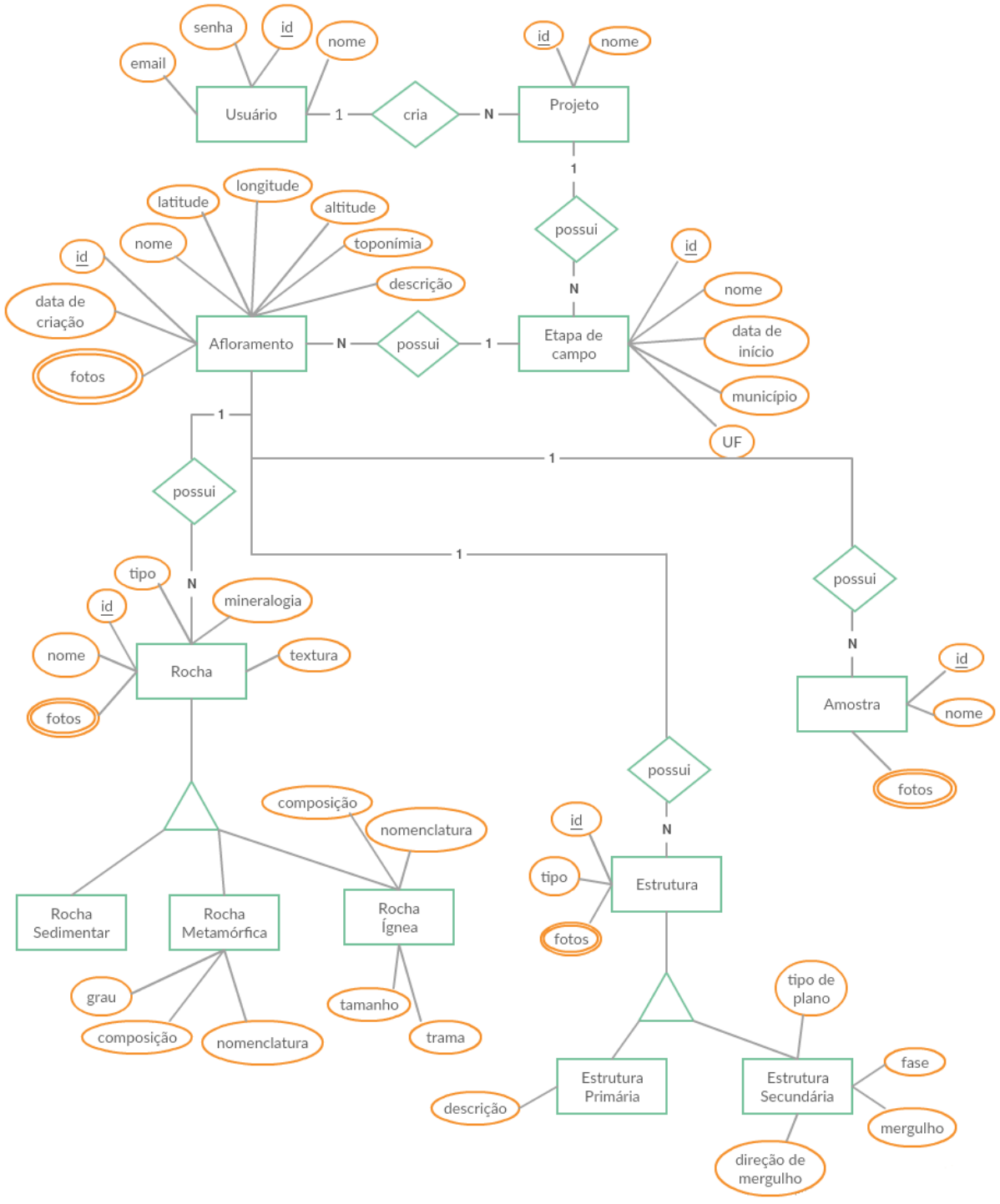

Figura 3.6: Diagrama Entidade-Relacionamento.

Os problemas identificados e as funcionalidades propostas para solucioná-los encontram-se na Seção 3.3.

3.2.6 *Design* e Protótipo

Após a definição dos requisitos, foi elaborado um modelo conceitual, que é a concepção do aplicativo que será desenvolvido. Após esse planejamento inicial, iniciou-se o processo de *design* da interface de usuário. Existem várias técnicas, mas foram utilizadas apenas duas:

- ***Sketching***: é o planejamento inicial de telas realizado através de rascunhos feitos em papel. É importante tentar gerar várias alternativas para que a melhor delas seja escolhida e, se possível, realizar essa atividade em grupo, de forma a agregar ideias e visões diferentes. Esta técnica pode ser realizada com o auxílio de softwares de desenho, porém é recomendável utilizar lápis e papel, pois desta forma não é preciso focar em aspectos menos importantes como tamanho de fontes, cor, alinhamento e etc. Além disso, por ser mais fácil de ser feito, os rascunhos também serão mais facilmente descartados, se necessário, e poderão sofrer inúmeras alterações até o fim de sua implementação [17].
- ***Wireframing***: é uma versão refinada dos rascunhos feitos na etapa anterior. Nesta técnica, sim, são utilizados softwares de prototipação e há uma maior preocupação com a aparência da interface, incluindo suas cores, tamanho dos elementos, fontes, dentre outros [17].

Para o trabalho em questão, foram feitos vários *sketches*, que tiveram diversas alterações, de acordo com o andamento das reuniões com os usuários. Alguns exemplos são mostrados na Figura 3.7.

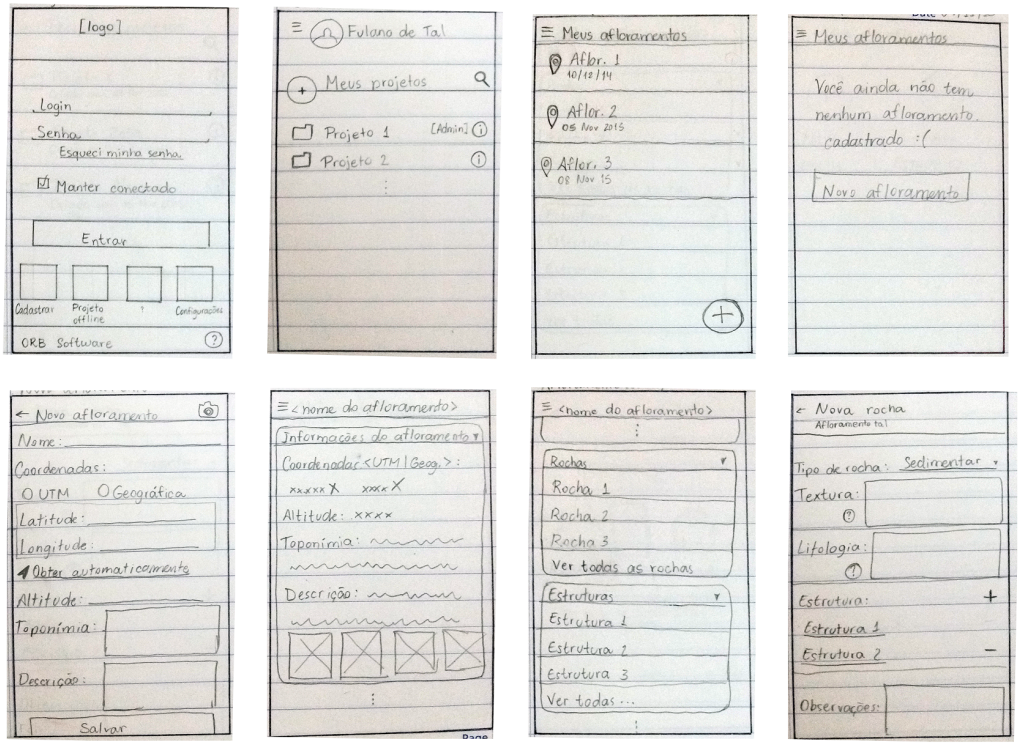

Figura 3.7: *Sketches*.

Uma vez que foi estabelecida uma estrutura satisfatória para os desenhos das telas, iniciou-se a criação dos *wireframes* com o auxílio de uma ferramenta de desenho chamada *Just In Mind* [5]. Tal ferramenta permite não somente a criação das telas com elementos específicos da plataforma *Android*, mas também a transição entre elas através do pressionamento de um botão, ou outros meios. Desta forma, foi gerada uma espécie de protótipo que foi utilizado para facilitar a transmissão de ideias aos *stakeholders* e validar as telas. Imagens do protótipo podem ser vistas na Figura 3.8.

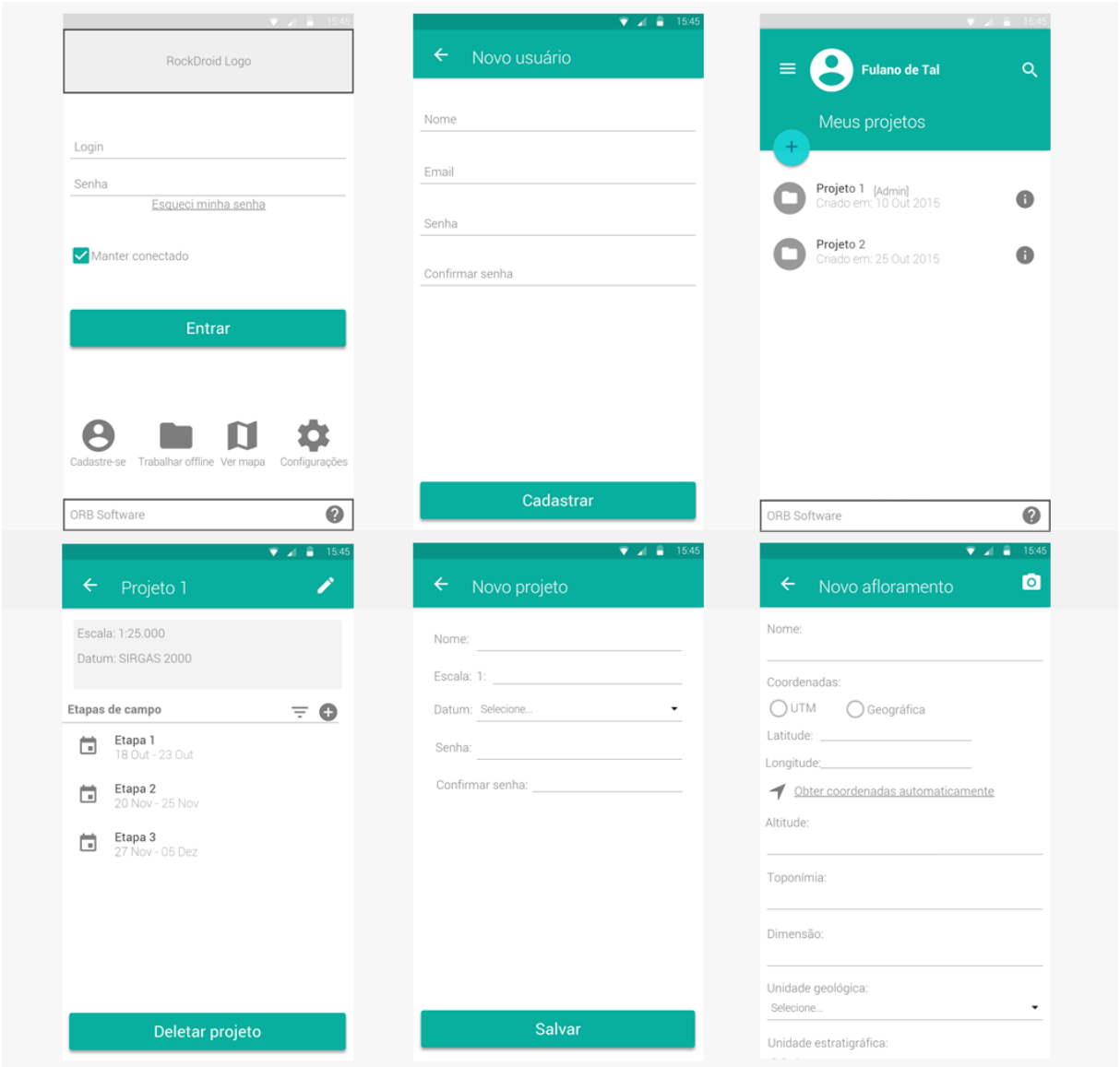

Figura 3.8: Imagens do Protótipo.

[5] http://www.justinmind.com/mobile

3.3 Descrição da Aplicação

Com base no estudo realizado sobre outras ferramentas e nos requisitos levantados através das análises de domínio, tarefas e usuário, foi possível descrever mais detalhadamente o aplicativo a ser desenvolvido para esta pesquisa. O RockDroid é uma solução de software criada para suprir uma necessidade identificada nas pesquisas de campo realizadas por profissionais e estudantes de geologia: a substituição de cadernetas de papel por formulários eletrônicos para a coleta de dados.

O objetivo da aplicação é prover ferramentas e formulários para a obtenção e inserção de dados durante pesquisas de campo, além de algumas outras funcionalidades, como: mapa, exportação de dados, opções de gerenciamento de projetos e sincronização dos dados com um servidor central.

3.3.1 Problemas Identificados

Os problemas identificados no processo atual de coleta dos dados foram:

- Dificuldade no gerenciamento de projetos: cabe aos próprios pesquisadores separarem e organizarem seu material de acordo com seus projetos e saídas de campo;
- Falta de um padrão definido para os dados: cada pesquisador anota as informações que achar mais relevante, o que pode tornar os dados da pesquisa incompletos;
- Falta de cadastro informatizado dos dados coletados em campo: a ausência de cadastro informatizado dos dados coletados pode resultar em perda de dados, dados desatualizados e perda de tempo em organizar, digitalizar e analisar os dados posteriormente;
- Necessidade de utilização de diversas ferramentas para obtenção de diferentes dados: são necessários vários equipamentos diferentes para coletar dados, como GPS, câmera fotográfica, caderno de anotações, entre outros;
- Demora na entrega da análise dos dados: os dados coletados em papel precisam ser organizados e digitalizados antes que qualquer tipo de processamento possa ser feito;
- Necessidade de visualização dos dados obtidos: os dados já coletados em um projeto precisam ser visualizados pelos geólogos durante o momento da coleta.

Para cada problema identificado, o aplicativo RockDroid apresenta uma proposta de solução como apresentado na Tabela 3.4.

Tabela 3.4: Tabela de Problemas x Funcionalidades.

Problema	**Funcionalidades**
Dificuldade no gerenciamento de projetos.	Cadastro de usuários; Autenticação; Cadastro de projetos; Cadastro de etapas de campo.
Falta de um padrão definido para os dados.	Cadastro de afloramentos; Cadastro de rochas; Cadastro de estruturas; Cadastro de amostras.
Falta de cadastro informatizado dos dados coletados em campo.	
Necessidade de utilização de diversas ferramentas para obtenção de diferentes dados.	Obtenção de coordenadas via GPS; Registro fotográfico dos recursos cadastrados.
Demora na entrega da análise dos dados.	Exportação de dados; Sincronização.
Necessidade de visualização dos dados obtidos.	Download de mapa; Visualização de afloramentos no mapa.

3.3.2 Funcionalidades

As funcionalidades que serão implementadas no RockDroid são descritas mais detalhadamente a seguir:

- **Cadastro de usuários:** manutenção do cadastro de usuários, com inclusão e exclusão. Dados de entrada: nome, email e senha;
- **Autenticação:** restringe o uso do aplicativo aos usuários cadastrados previamente. Com exceção da autenticação de visitantes, que permite o uso das funções *offline* do aplicativo sem necessidade de cadastro prévio nem conexão com a *Internet*;
- **Cadastro de projetos:** manutenção do cadastro de projetos, com inclusão, alteração e exclusão. Os projetos cadastrados por um usuário só podem ser visualizados e mantidos por este mesmo usuário, com exceção dos projetos criados pelo usuário visitante, que são públicos para qualquer outro usuário que fizer *login* nesse mesmo dispositivo. Dados de entrada: nome;
- **Cadastro de etapas de campo:** manutenção do cadastro de etapas, com inclusão, alteração e exclusão. Dados de entrada: nome, data de início, município e UF;
- **Cadastro de afloramentos:** manutenção do cadastro de afloramentos, com inclusão, alteração e exclusão. Dados de entrada: nome, latitude, longitude, altitude, toponímia, descrição e fotos;
- **Cadastro de rochas:** manutenção do cadastro de rochas, com inclusão, alteração e exclusão. Dados de entrada: nome, tipo de rocha, textura, mineralogia, cor e composição, tamanho, trama, nomenclatura, grau metamórfico, composição e fotos;

- **Cadastro de estruturas:** manutenção do cadastro de estruturas, com inclusão, alteração e exclusão. Dados de entrada: tipo de estrutura, descrição, tipo de plano, fase, mergulho, direção de mergulho e fotos;
- **Cadastro de amostras:** manutenção do cadastro de amostras, com inclusão, alteração e exclusão. Dados de entrada: nome e fotos;
- **Obtenção de coordenadas via GPS:** utilização do sistema de localização do dispositivo para obtenção das coordenadas do usuário;
- **Registro fotográfico dos recursos cadastrados:** utilização da câmera do dispositivo para captura de imagens e associação destas aos afloramentos, estruturas, amostras e rochas cadastradas;
- **Exportação de dados:** exportação de dados em formato de planilhas do Excel (XLS);
- **Sincronização:** sincronização entre o banco de dados local do dispositivo e o banco de dados do servidor central;
- **Visualização de afloramentos no mapa:** visualização dos pontos coletados no mapa;
- ***Download* de mapa:** *download* de mapa para visualização sem necessidade de conexão com a *Internet*.

As funcionalidades encontram-se representadas no diagrama de Casos de Uso da Figura 3.9.

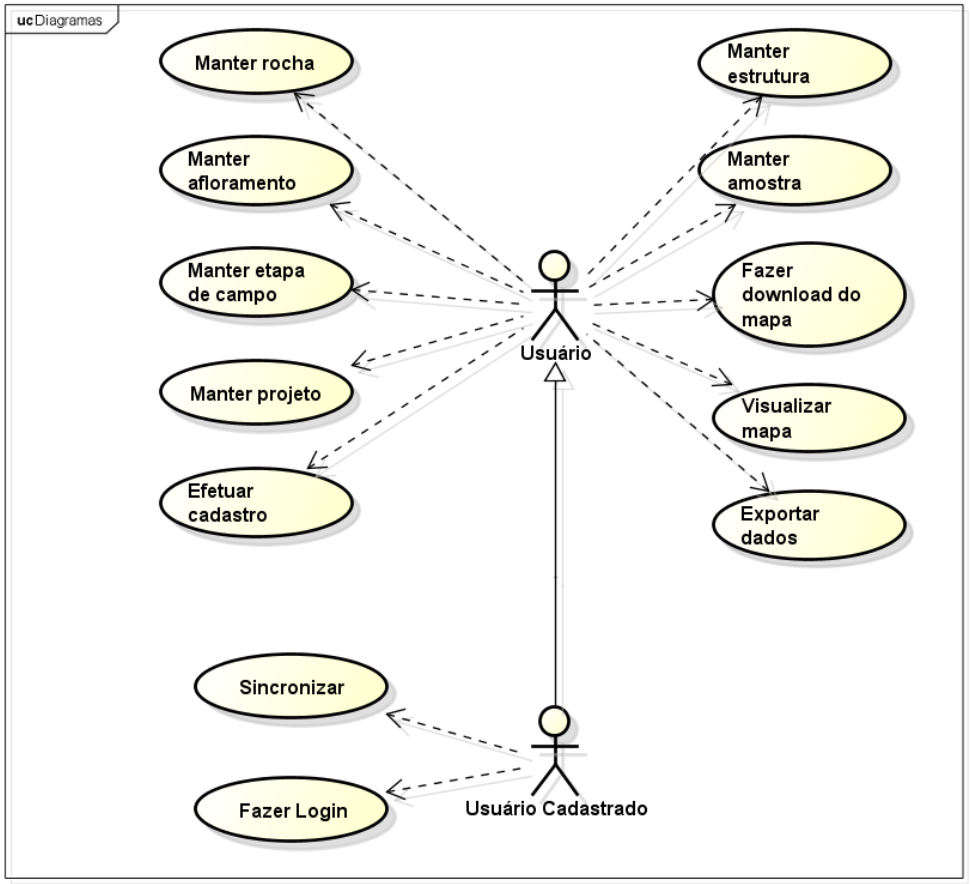

Figura 3.9: Diagrama de Casos de Uso.

3.4 O processo de Desenvolvimento

3.4.1 Projeto da Arquitetura

A arquitetura do sistema foi projetada com o objetivo de maximizar o reúso dos componentes e separar as responsabilidades. Além disso, para que o sistema seja facilmente evoluído, a arquitetura foi pensada para que seja independente de *frameworks*, independente de banco de dados e testável. Para realizar estas metas, a arquitetura utiliza padrões de arquitetura bem difundidos como a divisão em camadas, a inversão de dependências, a injeção de dependências e o *Model-View-Presenter* (MVP)[6].

Camadas

A separação das classes do sistema em camadas com responsabilidades específicas é a base de boa parte dos padrões arquiteturais porque permite uma maior organização do código e evita a multiplicação de diferentes usos para um mesmo componente. Isto é possível porque cada camada possui regras para acessar seus recursos. Assim, facilita o entendimento do código e permite que uma mudança em um determinado componente tenha uma repercussão pequena em outras partes do sistema.

[6] http://www.wildcrest.com/Potel/Portfolio/mvp.pdf

A arquitetura foi projetada para possuir quatro camadas distintas, como mostrado na Figura 3.10.

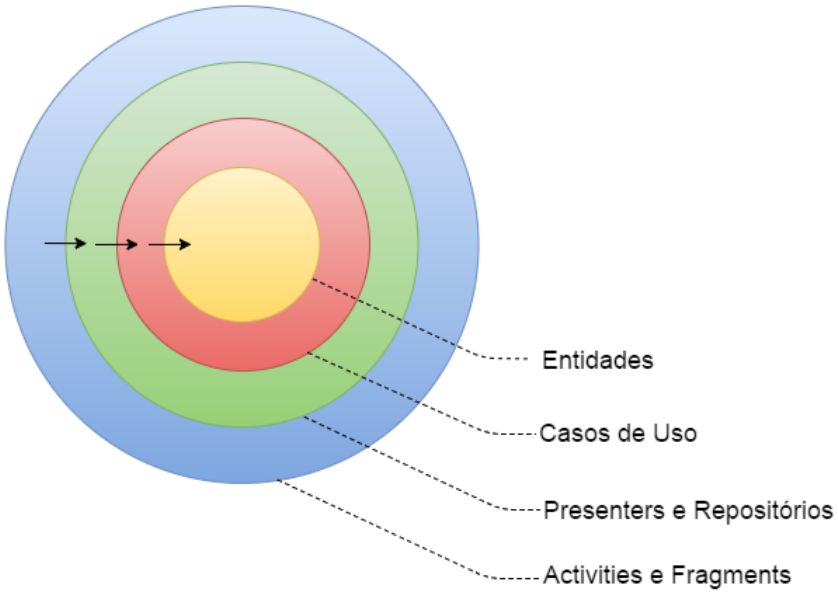

Figura 3.10: Camadas da Arquitetura[7].

As setas no diagrama representam a regra de dependência entre as camadas definida para esta arquitetura: apenas as camadas externas podem acessar os recursos das camadas internas. Assim, a **camada de entidades** desconhece a existência de todas as outras camadas. A **camada de casos de uso** conhece apenas a **camada de entidades**. A **camada de acesso** conhece as outras duas camadas mais internas. E finalmente, a **camada de apresentação** conhece todas as outras camadas. Uma descrição detalhada de cada camada é dada abaixo:

- **Camada de entidades:** entidades são classes que são modelos do domínio, ou seja, classes que fariam sentido não apenas no sistema que está sendo desenvolvido, mas em qualquer sistema que lide com o mesmo contexto. No RockDroid, são exemplos de entidades o modelo do usuários, representado pela classe *User*, e o modelo de rochas, representado pela classe *Rock*, dentre vários outros.
- **Camada de casos de uso:** casos de uso são classes que representam regras de negócio específicas da aplicação em desenvolvimento. Estas classes dependem das entidades para realizar algum processamento sobre elas. No aplicativo, são exemplos de casos de uso a classe que modela a criação de um novo afloramento, denominada *SaveOutcropInteractor*, e a classe que executa a sincronização dos dados para o servidor central, denominada *SynchronizationInteractor*.
- **Camada de acesso:** contém os componentes que dependem diretamente dos casos de uso e das entidades, mas que não fazem parte da interface com o usuário. Entram nesta categoria, por exemplo, os componentes que lidam com repositórios de dados

[7]Baseado em blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

e aqueles que fazem o intermédio entre os casos de uso e o *framework* de interface com o usuário.

- **Camada de apresentação:** esta é a camada mais externa e depende da camada de acesso. Os componentes que são utilizados aqui possuem um ciclo de vida ditado pelo *framework* utilizado. No caso do sistema operacional *Android*, os componentes desta camada são as *Activities*, os *Fragments* e, em geral, todas as classes que recebem eventos das ações do usuário no aparelho.

Módulos

A IDE *Android Studio*, em conjunto com o *Gradle* [8], que é a ferramenta oficial de automação de *builds* para *Android*, permite que um aplicativo seja formado por um projeto constituído de vários módulos separados, sendo um dos módulos considerado o principal e os demais conectados a ele da mesma forma que bibliotecas.

Esse suporte a projetos multi-módulo permite que o desenvolvedor seja forçado a aderir à separação de camadas sugerida na arquitetura de referência, já que as classes do módulo mais interno, por uma questão de controle de dependências, não têm acesso às classes do módulo mais externo. No aplicativo RockDroid, foram criados quatro módulos distintos, os quais são:

- **Módulo *app*:** é o módulo principal do projeto e utiliza o *framework* do sistema operacional para criar os componentes de interface de usuário, ou seja, é neste módulo que se encontram as *Activities*, os *Fragments*, os *Dialogs* e todos os componentes que se relacionam com a criação de elementos na tela do aparelho como os recursos XML de *layouts* e menus. Além destes componentes, o módulo *app* também contém um mecanismo de injeção de dependências que provê as referências necessárias para construir cada classe do sistema. Conforme a arquitetura de referência, este módulo contém as classes que formam a **camada de apresentação**.

- **Módulo *domain*:** possui as classes da **camada de casos de uso** e da **camada de entidades**, ou seja, engloba todos os componentes que lidam com o domínio da aplicação. Ele foi projetado para ser independente de *frameworks* e depende apenas de bibliotecas Java utilizadas para facilitar o processamento das regras de negócio.

- **Módulo *presentation*:** faz a interface entre as classes de interface com o usuário e as regras de negócio do sistema e é responsável por fazer requisições para a camada de casos de uso e mapear a resposta para a camada de apresentação. Em termos de camada arquitetural, as classes neste módulo são parte da **camada de acesso**. Este é também um módulo Java puro que tem como sua única dependência o módulo *domain*.

- **Módulo *data*:** é um módulo dependente do *framework Android* e é responsável por acessar dados providos por diferentes repositórios. Os repositórios que as classes deste módulo acessam incluem o banco de dados local, SQLite, o sistema de armazenamento de preferências do *Android*, o banco de dados remoto, acessado através do serviço *web*, e os arquivos da memória interna ou externa do dispositivo. Em

[8] http://gradle.org/

conjunto com as classes do módulo *presentation*, as classes deste módulo completam a **camada de acesso** da arquitetura.

Model-View-Presenter (MVP)

Um dos objetivos do sistema RockDroid é prover uma boa experiência para o usuário, isto é, ter como resultado um aplicativo com boa usabilidade e responsivo. O MVP é um padrão de arquitetura conhecido pela sua responsividade, graças à resposta assíncrona aos eventos lançados pelas ações do usuário. Desta forma, um dos focos do planejamento da arquitetura foi o de adequar o padrão MVP ao sistema.

Conforme pode ser observado na Figura 3.11, o *View* do MVP corresponde aos componentes de interface com o usuário do módulo *app*, assim como o *Model* corresponde aos componentes do módulo *domain*. Desta maneira, os componentes denominados *Presenters* foram desenvolvidos no módulo *presentation* para intermediar o evento recebido da *View*, chamando a classe de negócio responsável por executar a ação necessária e posteriormente devolver a resposta que deve ser repassada ao usuário.

A implementação deste padrão permite que os componentes de interface com o usuário não precisem ter conhecimento dos de negócio diretamente, o que ajuda a simplificar estas *views* ao deixá-las responsáveis apenas por tratar o comportamento dos atributos do próprio *layout*.

Na arquitetura projetada, outra responsabilidade dada às *Views* é a de lidar com a navegação entre as diferentes telas do sistema. Ou seja, os *Presenters* respondem sempre à mesma classe de apresentação que o chamou. Desta forma, se estabelece uma relação de um para muitos entre *Views* e *Presenters* de forma que todo *Presenter* é projetado para uma *View* específica, enquanto uma *View* pode fazer uso de vários *Presenters* caso seja necessário.

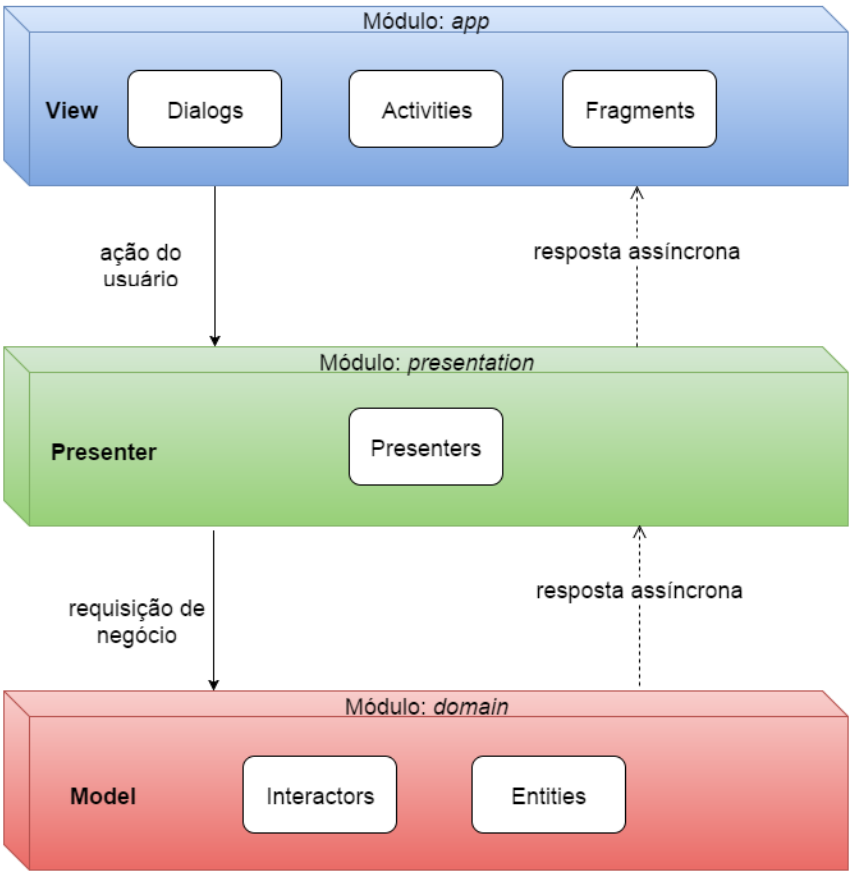

Figura 3.11: Visão do Padrão MVP Integrado aos Módulos da Arquitetura.

Inversão de dependências

O princípio da inversão de dependências também é um padrão arquitetural muito usado com o propósito de permitir que uma camada mais interna da arquitetura possa utilizar os serviços providos por uma camada mais externa sem, no entanto, ferir a arquitetura proposta.

Na linguagem Java, este princípio se baseia no uso de **interfaces** para oferecer uma abstração no módulo mais interno que pode ser implementada pelo módulo mais externo. Assim, o módulo interno não necessita de dependências externas e em tempo de compilação a arquitetura se mantém intacta, mas o fluxo de dependências se inverte em tempo de execução.

No RockDroid, a inversão de dependências é usada para que o módulo de domínio possa usar os recursos de armazenamento e recuperação de dados dos repositórios, sem que para isto precise depender do módulo *data*.

Em tempo de execução, o fluxo de chamadas dos métodos segue como demonstrado na Figura 3.12. O primeiro método do fluxo será chamado no módulo *app*, causado por uma interação do usuário ou pelo ciclo de vida do sistema operacional. Em seguida, este módulo passará uma mensagem para uma classe do módulo *presentation*, que, por sua vez, irá chamar uma determinada lógica de negócio contida no módulo *domain* e, se for

necessário obter dados para processar a requisição, o fluxo continuará para uma classe do módulo *data*.

Para que o sistema resolva a requisição de negócio, várias interações de chamadas e respostas podem acontecer neste fluxo durante o processamento, mas o resultado final da resposta seguirá o caminho inverso: do módulo *data*, para o *domain*, seguindo para o módulo *presentation* e finalizando com um *feedback* para o usuário apresentado em uma classe do módulo *app*.

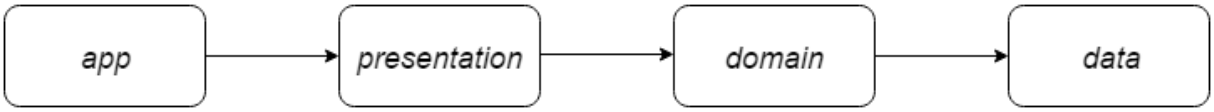

Figura 3.12: Fluxo de Dependências entre os Módulos em Tempo de Execução.

No entanto, em tempo de compilação as dependências são como apresentadas na Figura 3.13. O módulo *domain* não depende dos outros módulos, graças à aplicação do padrão de inversão de dependências, ou seja, além das classes que lidam com a lógica de negócio, este módulo contém também interfaces para os serviços que ele precisa. Estes serviços são implementados no módulo *data* que, consequentemente, depende do *domain*. Durante a execução, o mecanismo de injeção de dependências fornece a referência à classe concreta do serviço, permitindo a conexão correta entre os módulos.

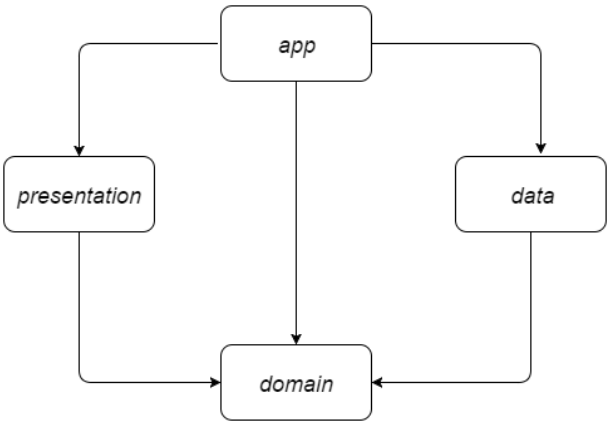

Figura 3.13: Dependências entre os Módulos em Tempo de Compilação.

Um exemplo de como a inversão de dependências toma forma em termos práticos pode ser visto na Figura 3.14. Esta imagem apresenta um diagrama de classes que demonstra que as dependências necessárias para o processamento do caso de uso estão todas armazenadas internamente ao módulo *domain*. Em tempo de execução, quando o caso de uso é instanciado, ele recebe em seu construtor a instância concreta da classe provedora do serviço, através do mecanismo de injeção de dependências.

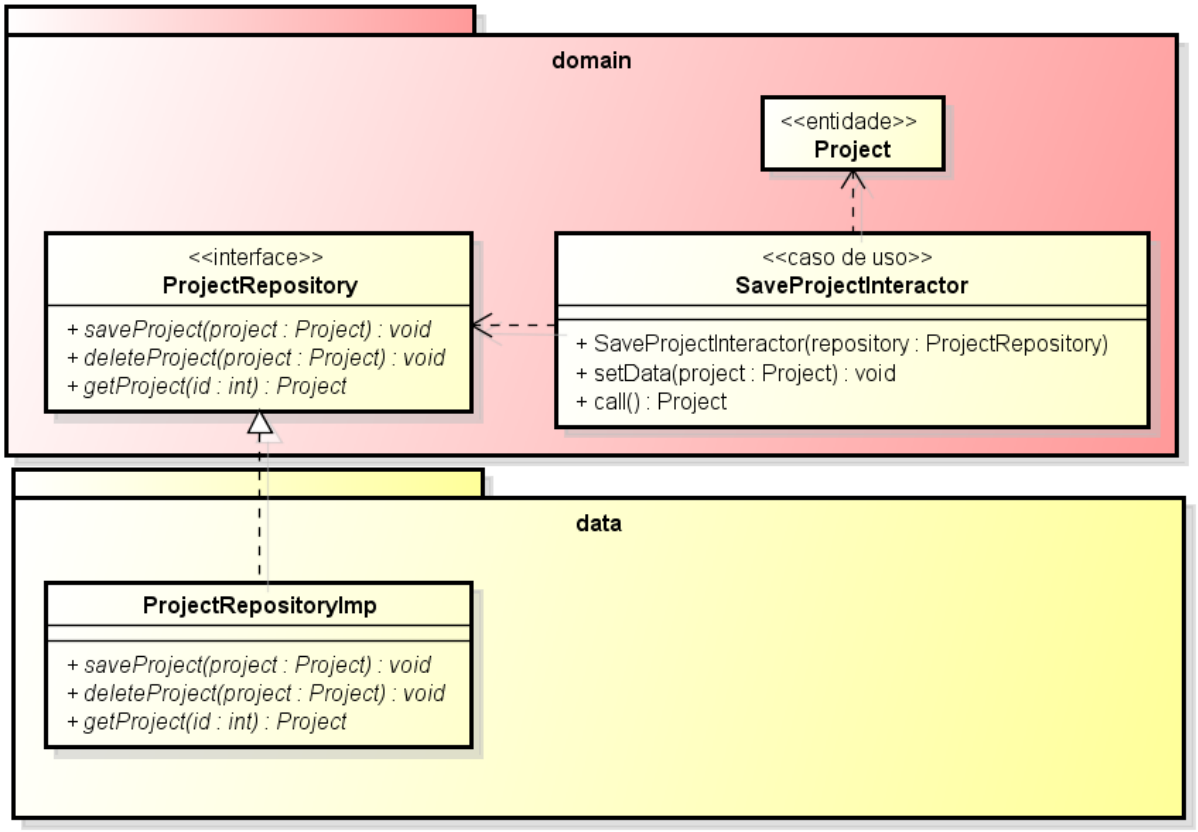

Figura 3.14: Diagrama que Exemplifica como a Inversão de Dependências é Usada.

Uma vantagem de usar este padrão de inversão de dependências é que o módulo *domain* não precisa ser um módulo *Android* e pode ser reutilizado sem quaisquer mudanças em vários outros contextos. Assim, as regras de negócio implementadas neste módulo podem ser reutilizadas, por exemplo, em uma aplicação Java para *Desktop* ou *web*.

Outra vantagem deste princípio é tornar o domínio da aplicação independente das mudanças nos detalhes da implementação das camadas externas. O domínio não tem qualquer conhecimento sobre qual o banco de dados, ou até sobre qual tipo de armazenamento está sendo utilizado, e alterações nestes detalhes não implicarão em qualquer mudança no código do módulo *domain*.

Injeção de dependências

Pode ser observado da Figura 3.13 que o módulo *app* depende do módulo *data*. No entanto, de acordo com o padrão MVP, nenhuma classe *View* do módulo principal deveria utilizar diretamente os serviços dos repositórios.

A aparente contradição de escopo das dependências na verdade não existe. Conforme mencionado, uma das responsabilidades do módulo principal *app* é a de prover o mecanismo de injeção de dependências para possibilitar a conexão entre os componentes em tempo de execução.

É a necessidade desse mecanismo que implica na associação de todos os demais módulos como dependências diretas do módulo principal. Com acesso a todas as classes do sistema, o *app* tem a responsabilidade de criar as instâncias destas e repassá-las por

parâmetro, conectando, assim, todos os componentes à implementação dos serviços que forem necessários.

A separação da construção das instâncias da lógica implementada em cada classe permite uma melhor definição de escopo do ciclo de vida de cada instância e facilita a substituição de uma determinada implementação de uma interface por outra, quando necessário.

Processamento assíncrono

A resposta assíncrona dos componentes que implementam os casos de uso é possível porque foram desenvolvidas classes de infraestrutura para a arquitetura de referência que provê um *pool* de *threads*.

Este esquema da infraestrutura permite enfileirar os Casos de Uso para que sejam executados sempre que uma *thread* estiver disponível dentro deste *pool*. O número de *threads* é fixo, mas configurável em tempo de compilação.

Dada a limitação do poder de processamento dos dispositivos móveis, a quantidade de tarefas assíncronas paralelas deve ser um número relativamente pequeno. O número de três *threads* foi escolhido inicialmente para o RockDroid.

3.4.2 Desenvolvimento

O desenvolvimento do aplicativo e do serviço *web* do servidor central foi focado na qualidade do código. A arquitetura também teve papel fundamental na estruturação do aplicativo e as funcionalidades foram implementadas sempre visando a conformidade arquitetural, para que o código resultante se mantenha legível e que novas funcionalidades possam ser implementadas sem comprometer outras partes do sistema.

Softwares utilizados

Os principais softwares utilizados foram a IDE *Android Studio*, para implementação do aplicativo, e o *Git* [9] para controle de versões, com auxílio do *Github* como hospedeiro dos repositórios do aplicativo [10] e do serviço *web* [11].

Tanto o aplicativo quanto o serviço *web* foram desenvolvidos com código aberto para permitir que outros desenvolvedores interessados possam contribuir no futuro para a evolução do aplicativo, seja com a manutenção ou adição de código.

Para assegurar a qualidade do código, o repositório do aplicativo foi associado a uma ferramenta de integração contínua chamada *Travis CI* [12]. Esta ferramenta cria automaticamente um ambiente temporário para compilar e executar todos os testes sobre o código sempre que o repositório é atualizado.

Se algum teste passa a falhar, a ferramenta notifica toda a equipe. Assim, os testes criados estão sempre sendo verificados e um mal funcionamento introduzido com uma nova mudança no repositório é identificado rapidamente.

[9] https://git-scm.com/
[10] https://github.com/Orb-software/Rockdroid
[11] https://github.com/Orb-software/RockDroidWebService
[12] https://travis-ci.org/

Desenvolvimento do aplicativo

O primeiro passo foi definir a estrutura do projeto dentro do *Android Studio* de acordo com a arquitetura especificada na seção anterior: o módulo *domain* contém toda a lógica de negócio, o módulo *data* é um módulo *Android* responsável por se comunicar com as fontes de dados da aplicação, o módulo *presentation* contém as classes *Presenter* do MVP e o módulo *app* contém as classes de interface com o usuário e une os demais módulos, provendo um mecanismo de injeção de dependências. A Figura 3.15 apresenta a distribuição dos módulos e pacotes Java no ambiente de desenvolvimento.

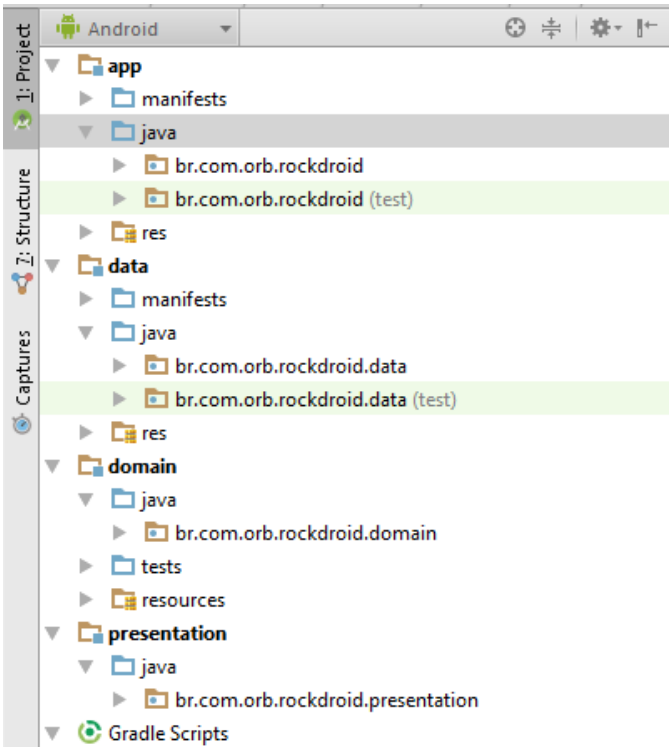

Figura 3.15: Organização do Projeto no *Android Studio*.

O próximo passo foi transformar o modelo entidade-relacionamento elaborado na elicitação de requisitos para o modelo físico do banco. O resultado desta transformação é apresentado na Figura 3.16. Pode-se observar que os atributos multivalorados de fotos receberam suas próprias tabelas. Cada foto contém um atributo identificador e um atributo que armazena o caminho para acessar a foto no aparelho do usuário. Além disso, as especializações de rochas e estruturas foram unidas em tabelas únicas que reúnem todos os atributos. Por fim, atributos adicionais foram criados para lidar com a sincronização dos registros para o banco de dados central.

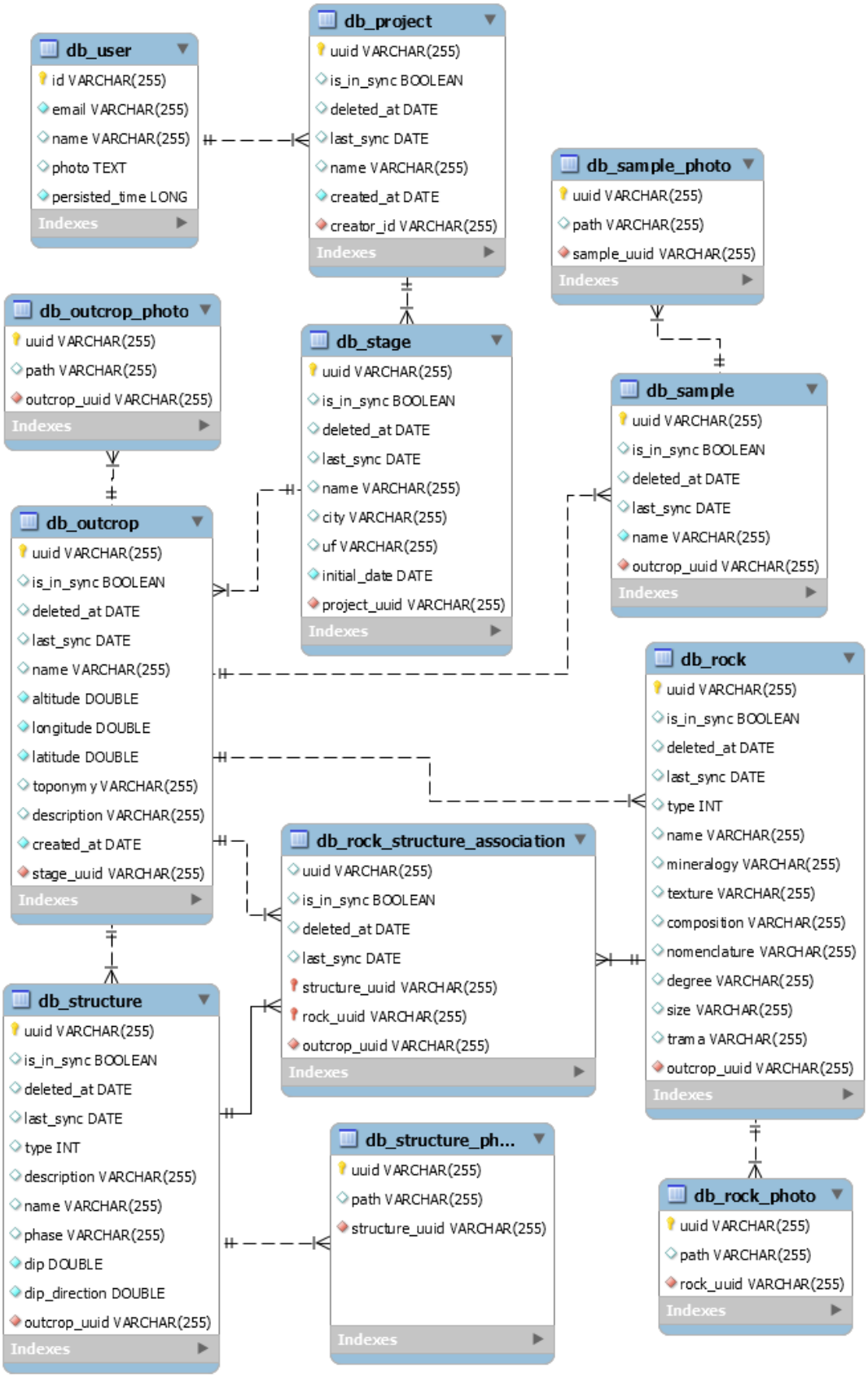

Figura 3.16: Modelo de Dados.

As funcionalidades que foram implementadas primeiro foram a autenticação e o cadastro de usuários. Foi definido que o usuário seria identificado por um email e, por isto, este atributo deveria ser único para cada usuário. Não havia necessidade de sincronizar registros de usuários, já que para cadastrar e autenticar um cliente seria necessário realizar sempre uma comunicação com o serviço *web*. No entanto, para permitir que os usuários pudessem realizar as coletas em campo sem necessidade de autenticar-se previamente, foi criado um usuário especial, *visitante*, que não exige cadastro prévio. Ao mesmo tempo, para evitar que projetos sem usuários definidos fossem sincronizados, foi criada uma regra de negócio que restringe a sincronização apenas aos usuários autenticados.

A criação das telas em XML, linguagem usada para criação dos *layouts* de tela em *Android*, foi realizada juntamente com a implementação das funcionalidades. As telas foram inspiradas no protótipo mostrado anteriormente e seguiram os padrões do *Material Design*, um novo padrão de interfaces do Google cujo objetivo é unificar o visual do *Android* com os dos aplicativos *Chrome OS* e dos serviços *web*.

A comunicação com o serviço *web* foi implementada em classes do módulo *data* utilizando a biblioteca *Retrofit* [13] para facilitar as chamadas à interface REST disponível. Os casos de uso que fazem uso do serviço *web* são os que lidam com autenticação e cadastro de usuário e o da sincronização dos recursos.

Algoritmo de sincronização

Como a sincronização é unidirecional - os dados são enviados do aplicativo para o servidor - o algoritmo de sincronização é simples e existe apenas do lado do cliente. Este algoritmo baseia-se em algumas premissas: identificador único universal para cada registro, controle de atualizações nos registros e controle de deleções nos registros.

Cada recurso criado no aplicativo de cada usuário deve possuir um identificador que seja único universalmente. Isto é necessário para tornar o protocolo de sincronização simples. Caso contrário, durante a sincronização, o servidor seria o responsável por criar um identificador único no banco remoto e este identificador teria que ser armazenado no banco local para que, nas próximas sincronizações, o servidor pudesse reconhecer o registro ao invés de criar um novo.

Utilizar um identificador universalmente único gerado pelo aplicativo durante a criação do registro evita que seja necessário manter dois identificadores distintos no banco local. O servidor utiliza como identificador de cada registro o mesmo identificador gerado no aplicativo, e, a partir deste valor, consegue encontrar o registro para efetuar atualizações ou sua deleção.

Para gerar tal identificador único, o aplicativo utiliza três valores distintos: o primeiro é um identificador do dispositivo, numerado pelo fabricante, presente em todos os aparelhos *Android*. Este valor é concatenado com uma cadeia de caracteres hexadecimais com 64-bits de tamanho, gerada aleatoriamente. Por fim, ao resultado desta concatenação é adicionado a data e hora do aparelho, traduzida em milissegundos. A cadeia de caracteres resultante é universalmente única e pode ser utilizada como identificador do registro no servidor sem preocupações realistas com colisões entre diferentes clientes.

Para realizar o controle de atualizações, cada tabela sincronizável existente no banco local deve possuir um campo adicional que indica se o estado atual do registro no banco

[13] http://square.github.io/retrofit/

local está em sincronização com o servidor ou não. Este campo, traduzido para uma primitiva *booleana* no código Java do aplicativo, tem *falso* como seu valor inicial. Após a sincronização de cada registro, este valor é atualizado para *verdadeiro*. Se o usuário realiza alterações no registro, este valor é atualizado novamente para *falso*. O algoritmo de sincronização utiliza o valor deste campo para enviar ao servidor apenas os registros que estejam marcados como fora de sincronia.

Para evitar que a sincronização crie, no servidor, registros que contenham chaves estrangeiras inválidas, que apontem para registros que ainda não foram sincronizados, um tratamento necessário é que a sincronização ocorra na ordem dos registros nas tabelas de hierarquia mais alta, seguindo para os registros nas tabelas de hierarquia mais baixa, conforme demonstra o fluxograma da Figura 3.17. Neste caso, o banco local só pode considerar um registro mais alto na hierarquia como sincronizado após todos os registros dependentes dele direta ou indiretamente terem sido sincronizados.

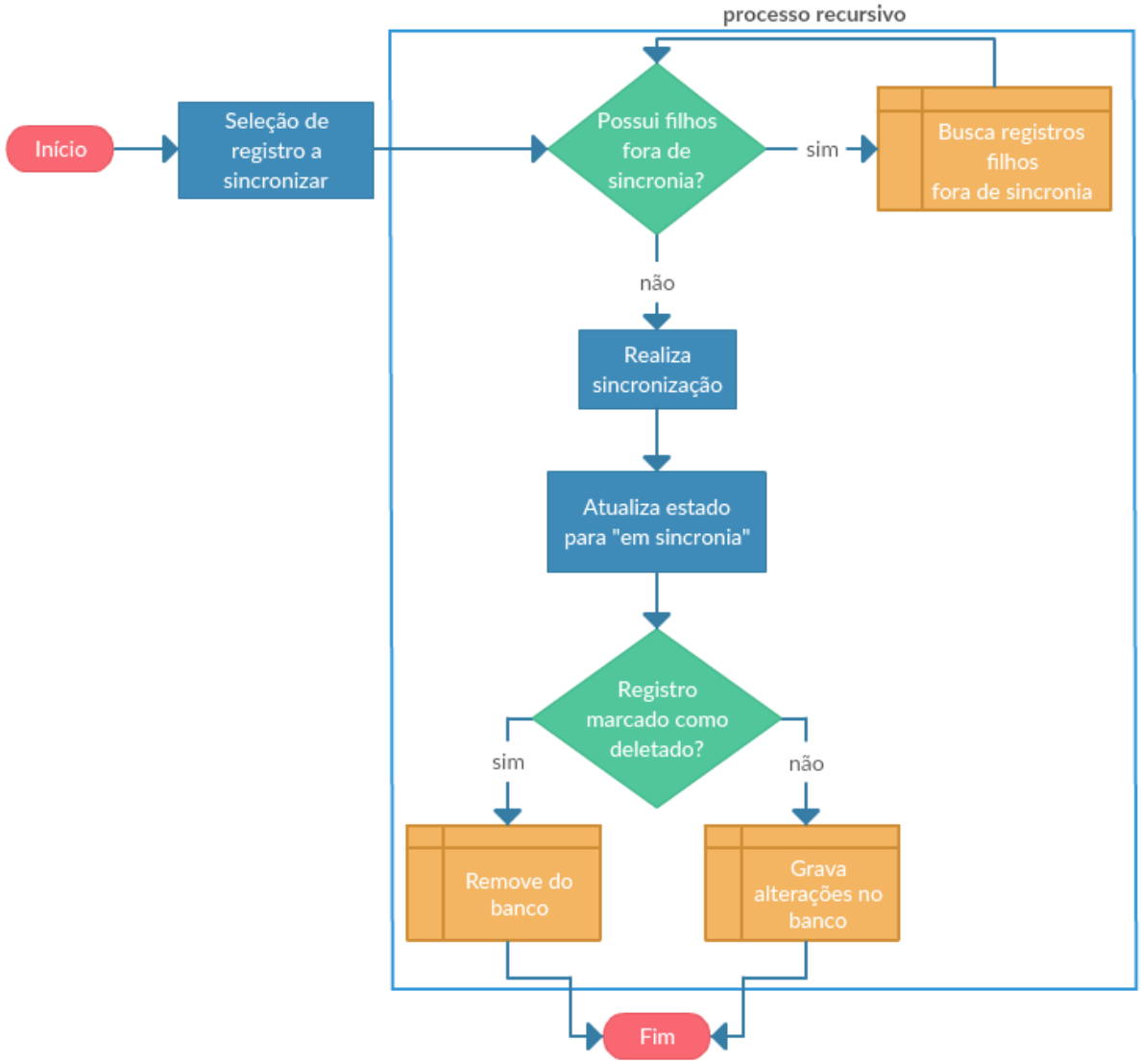

Figura 3.17: Fluxograma que Demonstra o Algoritmo de Sincronização.

A Figura 3.18 apresenta a hierarquia das entidades do sistema. Seguindo o procotolo de sincronização, um determinado projeto só pode ser considerado sincronizado após todas as suas etapas terem sido sincronizadas, e cada etapa só pode ser considerada sincronizada após todos os seus afloramentos terem sido sincronizados. Esta ideia se propaga por toda a hierarquia das entidades.

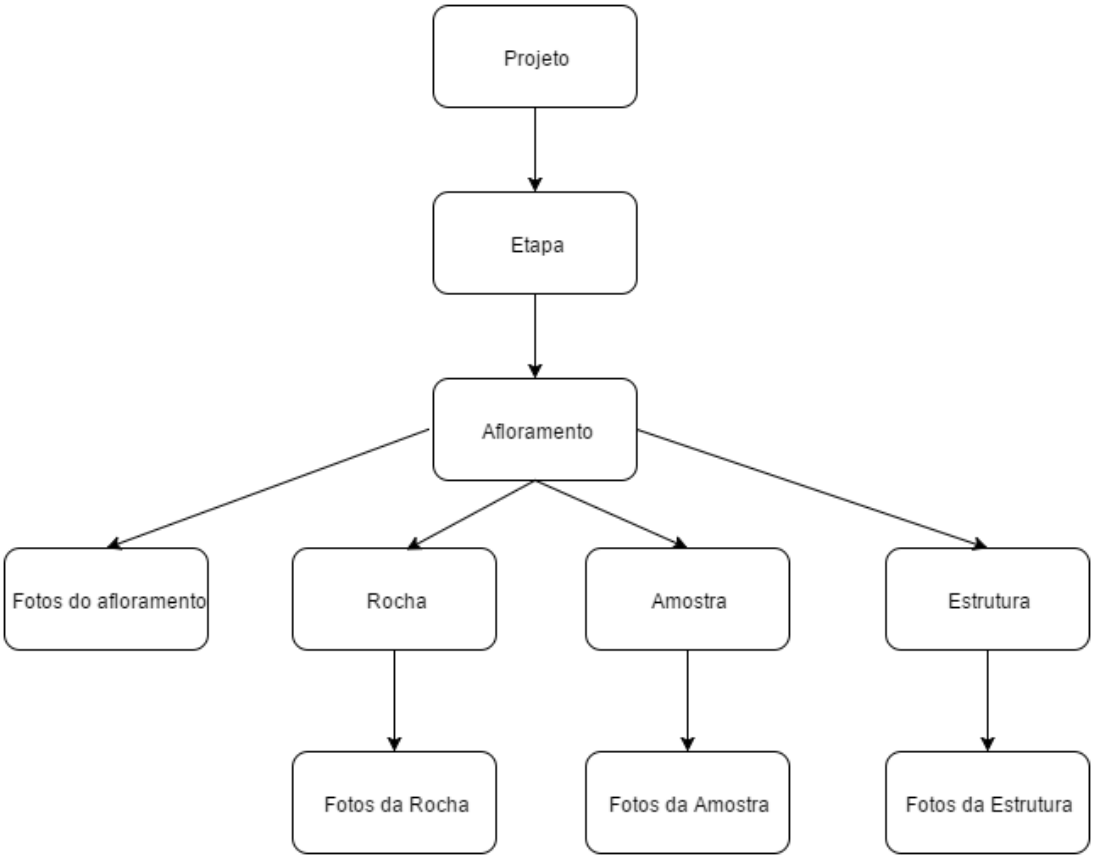

Figura 3.18: Hierarquia das Entidades do Sistema.

Esta metodologia de sincronização também lida com interrupções de rede ou demais erros que possam acontecer durante a sincronização. Em caso de uma exceção ocorrer, toda a hierarquia acima do registro que sofreu a interrupção será mantida localmente no estado **fora de sincronia**. Assim, a próxima sincronização irá continuar de onde a anterior parou, sem maiores consequências.

O controle de deleções funciona de forma semelhante, levando em consideração a mesma hierarquia apresentada na Figura 3.18. Quando um usuário requisita a deleção de um registro de uma entidade de hierarquia alta, todos os registros dependentes que estão em hierarquias mais baixas são deletados em cascata antes e só depois a deleção do registro requisitado realmente ocorre.

Apesar do registro ser deletado, a remoção real do registro do banco de dados local nem sempre acontece. Após um determinado registro ter sido sincronizado uma vez, a requisição de deleção não o remove de fato, mas altera um campo de controle que indica que aquele registro já foi deletado, informando a data que a deleção ocorreu. Isto permite que, na próxima sincronização, o registro seja marcado no banco remoto como deletado. Neste caso, a real remoção do banco local só acontece após esta segunda sincronização.

Por outro lado, um registro que nunca foi sincronizado pode simplesmente ser removido do banco local, já que o banco remoto nem sequer tinha conhecimento da existência deste registro.

Assim, o controle de deleções completo depende de três atributos adicionais em toda tabela sincronizável: um atributo que indica a data da deleção, um atributo que indica a data da ultima sincronização, cujos valores iniciais são nulos, e um atributo que indica se o estado atual do registro está sincronizado ou não, o mesmo utilizado para o controle de atualizações.

Desenvolvimento do serviço *web*

Diferentemente do aplicativo, o serviço *web* foi desenvolvido com foco em se obter uma interface de acesso aos dados que fosse fácil de ser entendida durante o desenvolvimento do aplicativo, que fosse genérica o suficiente para que possa servir a vários clientes distintos e que contivesse a menor quantidade possível de código, deixando parte das responsabilidades de validação de acesso para o banco de dados.

A arquitetura REST sobre o protocolo HTTP foi a escolhida pela sua simplicidade e pelo forte suporte aproveitado das bibliotecas de código aberto já existentes. Uma interface *RESTful* é capaz de prover um acesso ao serviço de maneira independente de clientes específicos e, assim, torna possível que o sistema opere com um conjunto heterogêneo de clientes, como, por exemplo, uma página *web*, um aplicativo do sistema operacional *iOS* ou, como no caso do RockDroid, um aplicativo *Android*.

Para que o código do serviço *web* pudesse ser desenvolvido de forma ágil, foi utilizado o *framework Spring Data Rest* [14] para expor os dados do banco de dados. Este *framework* permitiu o desenvolvimento de um código pequeno, já que o projeto herdou suas funcionalidades. Ao mesmo tempo, o uso desta biblioteca para criar a interface do serviço, proporcionou uma forte aderência aos conceitos HATEOAS, o que facilita a adequação dos clientes conforme novas versões da interface do serviço são implementadas.

Em relação à infraestrutura, os professores do Instituto de Geociências disponibilizaram um servidor para hospedar o serviço *web* do RockDroid. Esta máquina opera utilizando um sistema *Windows* conectado à uma rede de alta velocidade com endereço IP fixo [15] e já hospeda outros serviços relativos às pesquisas geológicas, por isso, possui um administrador designado para realizar sua manutenção.

A tecnologia Java JPA em conjunto com o *framework Hibernate* [16] foi usada para criar o mapeamento objeto-relacional entre as entidades do sistema e as tabelas do banco de dados. O sistema gerenciador de banco de dados escolhido para uso em produção foi o *PostgreSQL*, por ser de código aberto e pela sua compatibilidade com outros serviços já existentes no mesmo servidor de hospedagem.

O esquema do banco de dados do servidor é muito semelhante ao esquema do banco local desenvolvido para o aplicativo, conforme mostra a Figura 3.19. Uma das diferenças é que o banco remoto não armazena dados relativos à sincronização, já que é o aplicativo que gerencia localmente quais dados já foram sincronizados. A outra diferença está na forma de armazenar as imagens: enquanto no aplicativo só é necessário armazenar o caminho

[14] http://projects.spring.io/spring-data-rest/
[15] http://164.41.53.35/
[16] http://hibernate.org/orm/

para o arquivo no dispositivo, no banco central todas as imagens contêm um campo que guarda o nome do arquivo recebido e um campo para armazenar o seu conteúdo.

Para facilitar o processo de sincronização, o conteúdo da imagem é armazenado como um campo do tipo texto, resultante da codificação da imagem original para o formato *Base64*.

Nos termos do *framework Spring Data Rest*, um repositório é uma abstração de um gerenciador de um recurso, em que um recurso representa uma tabela do banco de dados. Ou seja, um repositório é uma classe que gerencia as transações de banco de dados sobre uma determinada tabela, podendo essa transação ser uma recuperação, remoção, alteração ou criação de registros, as operações básicas de um banco relacional.

O núcleo do serviço provido por este *framework* é a funcionalidade de exportar os métodos disponibilizados pelos repositórios em formato de URIs que seguem o modelo REST. A Tabela 3.5 exemplifica o mapeamento entre algumas transações executadas em uma tabela do banco de dados, o método provido pelo repositório que gerencia esta tabela, e a interface REST gerada para expor esta funcionalidade.

Tabela 3.5: Mapeamento entre a Transação Realizada na Tabela *project*, o Repositório e a Interface do Serviço

Transação	**Método do repositório**	**Interface REST**
Inserir registro	save(Project project)	Método HTTP: POST URI: http://164.41.53.35/projects/ Corpo: Dados em formato JSON
Atualizar um registro a partir de um id	save(Project project)	Método HTTP: PUT URI: http://164.41.53.35/projects/{id} Corpo: Dados atualizados em formato JSON
Remover um registro a partir de um id	delete(Project project)	Método HTTP: DELETE URI: http://164.41.53.35/projects/{id}
Recuperar um registro a partir de um id	findOne(String id)	Método HTTP: GET URI: http://164.41.53.35/projects/{id}
Recuperar todos	findAll()	Método HTTP: GET URI: http://164.41.53.35/projects/

A interface do serviço *web* foi desenvolvida completamente baseada no esquema do banco de dados. Por isso, foi criado um repositório para cada tabela do banco, gerando uma interface REST que possibilita executar as transações básicas no banco de dados sobre estas tabelas. Uma requisição *GET* feita à URI principal do servidor retorna uma listagem completa da interface, apresentando os serviços oferecidos em cada um dos repositórios, conforme mostra a Figura 3.20.

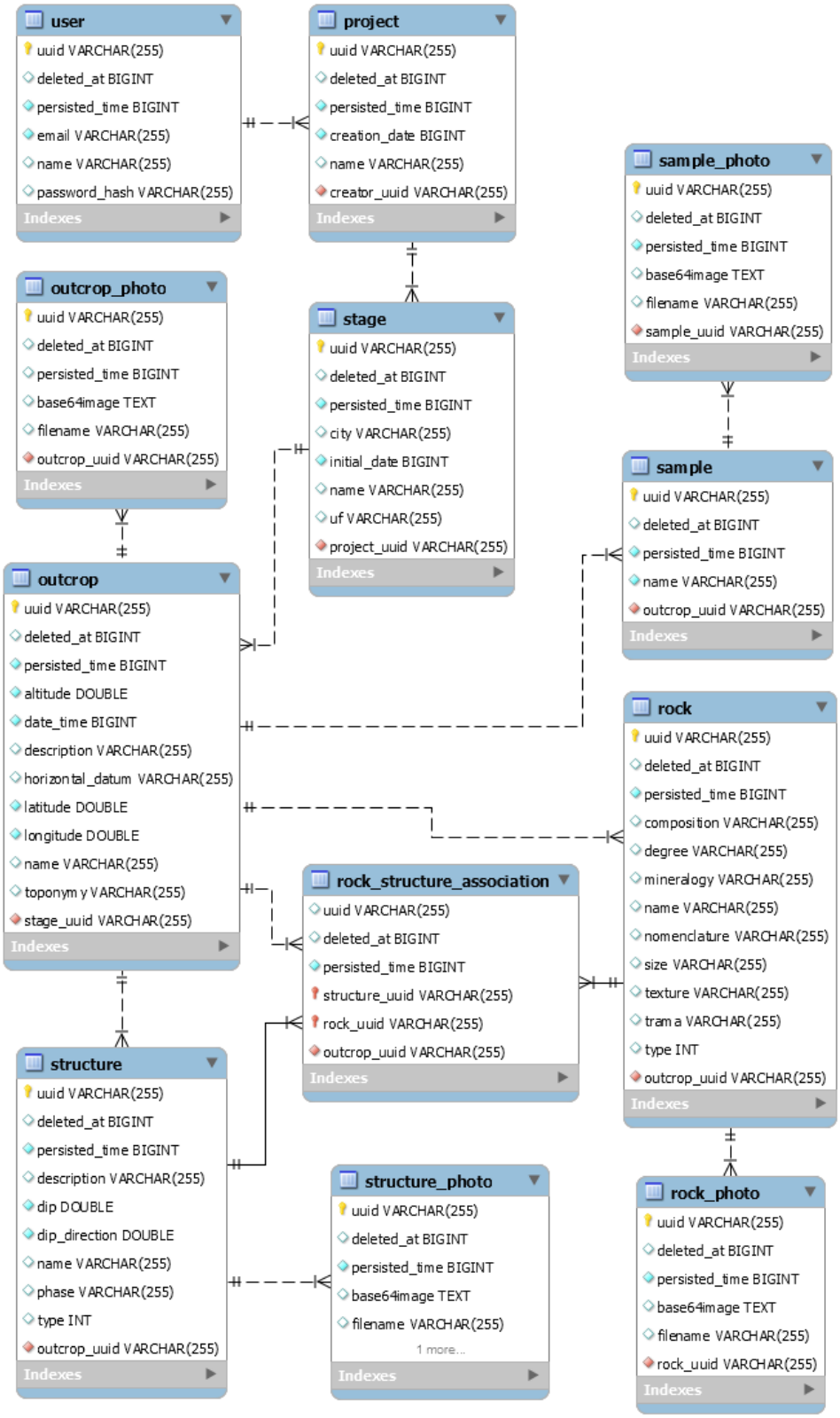

Figura 3.19: Modelo de Dados do Servidor.

```
{
  "_links" : {
    "rocks" : {
      "href" : "http://164.41.53.35/rocks{?page,size,sort}",
      "templated" : true
    },
    "outcrops" : {
      "href" : "http://164.41.53.35/outcrops{?page,size,sort}",
      "templated" : true
    },
    "structures" : {
      "href" : "http://164.41.53.35/structures{?page,size,sort}",
      "templated" : true
    },
    "structurePhotos" : {
      "href" : "http://164.41.53.35/structurePhotos{?page,size,sort}",
      "templated" : true
    },
    "outcropPhotos" : {
      "href" : "http://164.41.53.35/outcropPhotos{?page,size,sort}",
      "templated" : true
    },
    "stages" : {
      "href" : "http://164.41.53.35/stages{?page,size,sort}",
      "templated" : true
    },
    "associations" : {
      "href" : "http://164.41.53.35/associations{?page,size,sort}",
      "templated" : true
    },
    "samples" : {
      "href" : "http://164.41.53.35/samples{?page,size,sort}",
      "templated" : true
    },
    "rockPhotos" : {
      "href" : "http://164.41.53.35/rockPhotos{?page,size,sort}",
      "templated" : true
    },
    "projects" : {
      "href" : "http://164.41.53.35/projects{?page,size,sort}",
      "templated" : true
    },
    "users" : {
      "href" : "http://164.41.53.35/users{?page,size,sort}",
      "templated" : true
    },
    "samplePhotos" : {
      "href" : "http://164.41.53.35/samplePhotos{?page,size,sort}",
      "templated" : true
    },
    "profile" : {
      "href" : "http://164.41.53.35/profile"
    }
  }
}
```


Figura 3.20: Listagem Completa dos Serviços Expostos pela Interface REST.

Por fim, foi feito um controle simples de acesso, utilizando a autenticação básica do protocolo HTTP, que restringe o uso dos serviços apenas aos usuários cadastrados. A única URI em que não há este controle é a própria URI que executa o cadastro do usuário. Assim, de acordo com o protocolo de autenticação básica do HTTP, as demais requisições devem incluir em seu cabeçalho a propriedade *Authorization* cujo valor deve ser o resultado da codificação em *Base64* do seguinte texto: o email do usuário, seguido de dois-pontos, seguido da senha do usuário.

3.5 O Aplicativo

Ao final da fase de desenvolvimento do RockDroid, foi iniciada a etapa de testes, cujo objetivo era verificar o funcionamento da validação dos dados dos formulários, o comportamento do sistema em diferentes situações de conectividade e a sincronização dos dados. Tais testes são descritos mais detalhadamente no próximo capítulo deste documento. Uma vez que os resultados dos testes se tornaram satisfatórios, deu-se por encerrada a parte de programação deste projeto. O resultado final encontra-se descrito em detalhes nas próximas seções.

3.5.1 Telas do RockDroid

Nesta seção são apresentadas as principais telas do aplicativo.

Tutorial

Durante a primeira inicialização do RockDroid, é mostrado um pequeno tutorial descrevendo suas principais funcionalidades. A ideia é atrair a atenção do usuário, ensinar o básico sobre o aplicativo e destacar seus pontos fortes. Os *screenshots* do tutorial encontram-se na Figura 3.21.

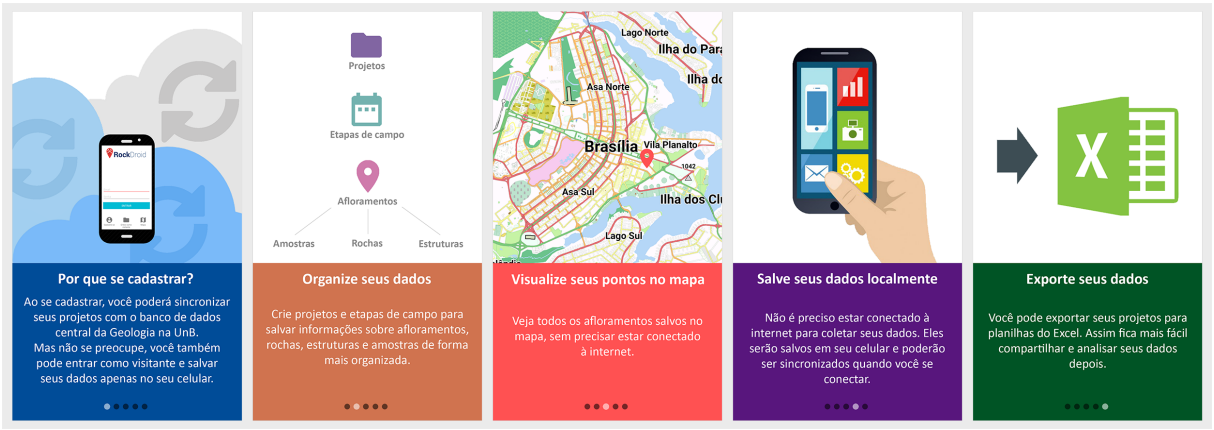

Figura 3.21: Telas do Tutorial.

Login e cadastro de usuário

A tela de *login* é mostrada ao final do tutorial, caso seja a primeira vez que o usuário acessa o aplicativo. Caso contrário, um código de decisão será executado para verificar se já existe um usuário logado no sistema e somente se não existir é que essa tela será mostrada (primeira tela da Figura 3.22). Se o usuário não tiver se cadastrado ainda, ele pode fazê-lo acessando a tela de cadastro de usuário (segunda tela da Figura 3.22) a partir da opção "Cadastre-se" da tela de *login*.

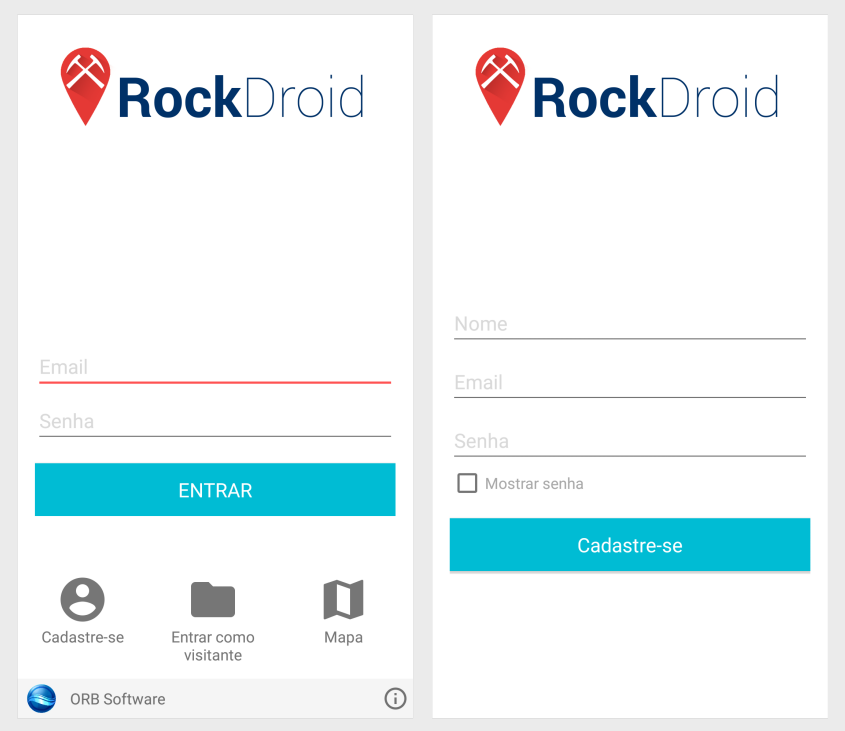

Figura 3.22: Telas de *Login* e de Cadastro de Usuário.

As opções de *login* e de cadastro de usuários só podem ser completadas se houver conexão com a *Internet*. Sabendo que um dos principais problemas enfrentados pelos pesquisadores no campo é a falta de conectividade, foi criada a opção "Entrar como visitante", que permite que o usuário entre na aplicação e colete seus dados normalmente, com a diferença de que ele não poderá sincronizar suas informações até que esteja conectado e efetue *login*.

Mapa

Outra funcionalidade que pode ser acessada a partir da tela de *login* é o "Mapa". A primeira ação que o código do mapa realiza é verificar se já existe um arquivo de mapa no dispositivo do usuário. Caso não exista, o aplicativo oferece a opção de "Baixar mapa", como na primeira tela da Figura 3.23. Caso o usuário já possua o mapa, este será carregado na tela, como na segunda tela da Figura 3.23.

Após verificar a existência do arquivo de mapa, o aplicativo verifica no banco de dados local a existência de afloramentos cadastrados para exibi-los na tela. Pontos que foram criados pelo usuário Visitante serão sempre mostrados no mapa, não importando se o usuário não fez *login*, entrou como Visitante ou entrou com credenciais válidas. Neste último caso, além dos pontos do Visitante, serão também mostrados os pontos criados por aquele usuário. A tela do mapa com marcadores é a última imagem da Figura 3.23.

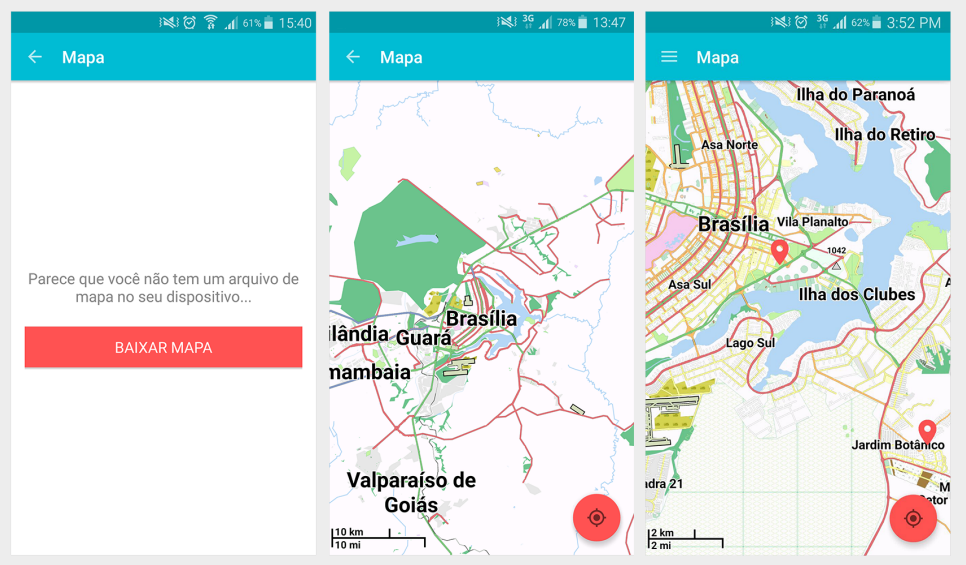

Figura 3.23: Telas do Mapa.

Listagem de projetos

Uma vez logado (seja usando suas credenciais ou entrando como Visitante), o usuário será redirecionado para a tela de listagem de projetos. Se ele estiver logado com suas credenciais, ele irá direto para esta tela sempre que abrir o aplicativo, até que seja escolhida a opção "Sair" do menu lateral, mostrado na primeira tela da Figura 3.24. Ao iniciar esta tela, o aplicativo irá buscar no banco de dados os projetos pertencentes ao usuário logado, incluindo projetos *offline* (projetos que o usuário criou como Visitante). Se nenhum resultado for encontrado, a segunda tela da Figura 3.24 será exibida. Caso contrário, aparecerá uma lista de projetos, como na última tela da figura.

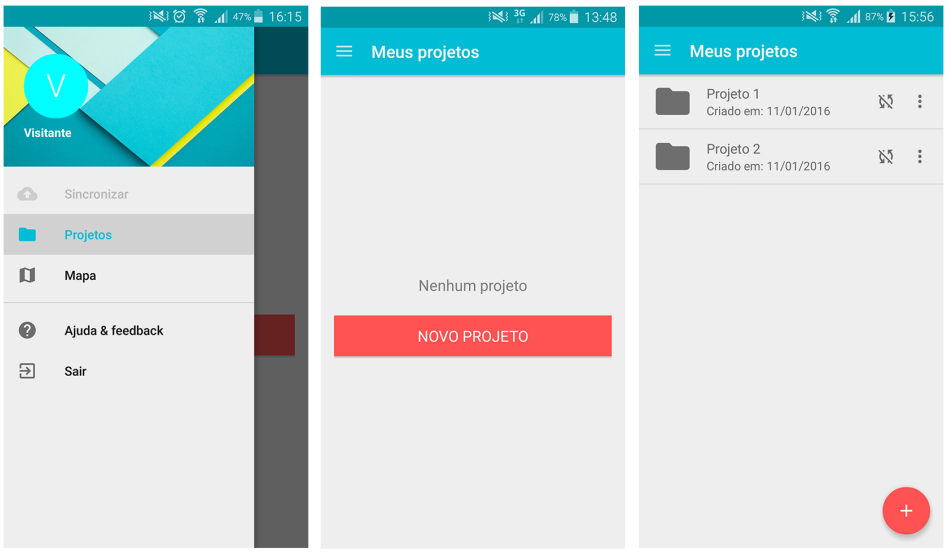

Figura 3.24: Menu Lateral e Tela de Listagem de Projetos.

Exportação de projetos

Cada projeto pode ser exportado individualmente para a memória do dispositivo do usuário em formato de planilhas do Excel. A exportação se inicia quando o usuário escolhe a opção "Exportar para Excel" no menu de *overflow* de um projeto, mostrado na primeira tela da Figura 3.25. Quando a exportação for concluída, uma mensagem informando o sucesso da operação e o diretório onde os arquivos foram salvos aparecerá na parte inferior da tela, como na segunda tela da Figura 3.25. A terceira tela mostra os arquivos criados pela exportação no diretório do RockDroid no dispositivo do usuário.

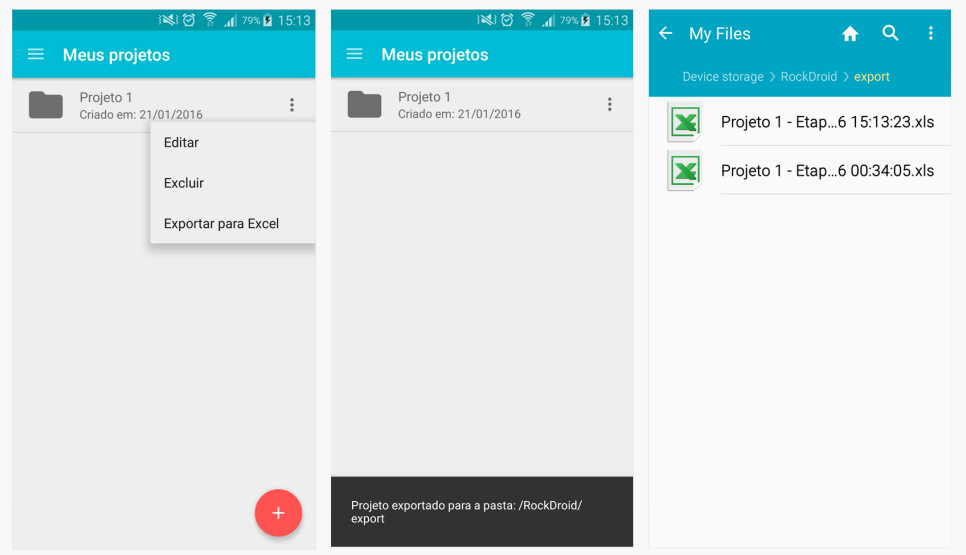

Figura 3.25: Exportação de Projetos.

Durante a exportação, o aplicativo cria um arquivo Excel para cada etapa de campo pertencente ao projeto exportado. Cada arquivo contém uma planilha para cada entidade (Afloramento, Rocha, Estrutura e Amostra). Na planilha referente aos afloramentos, estão presentes todos os seus atributos, mas apenas referências às rochas, estruturas e amostras pertencentes a eles, como mostra a Figura 3.26. Todas as entidades possuem fotos, cujos nomes são exibidos nas planilhas para que o usuário saiba encontrá-las em seu dispositivo.

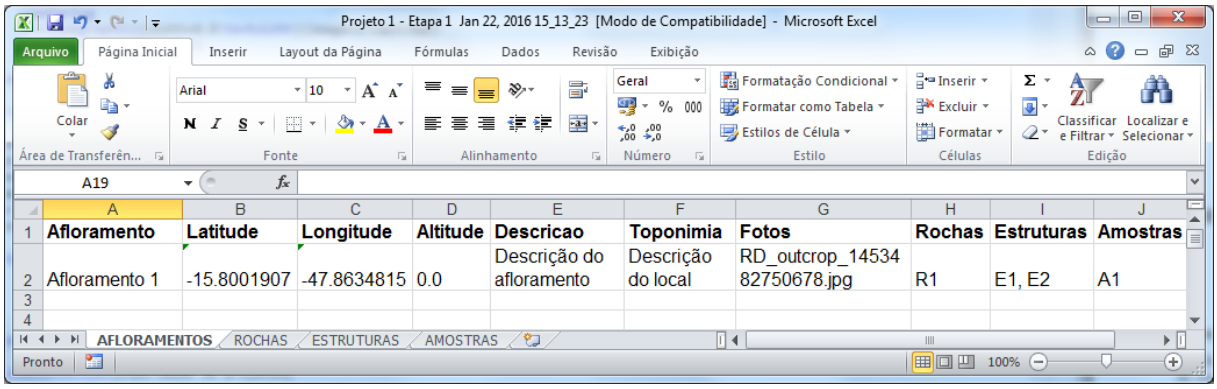

Figura 3.26: Planilha de Afloramentos.

A planilha de rochas mostra todos os atributos pertencentes a uma rocha, não importando seu tipo (sedimentar, ígnea ou metamórfica), e encontra-se na Figura 3.27.

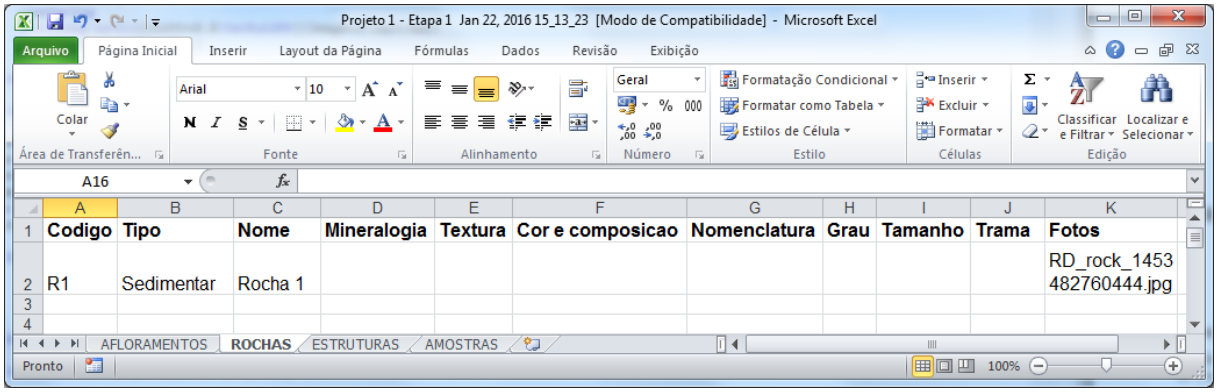

Figura 3.27: Planilha de Rochas.

A planilha de estruturas mostra tanto as estruturas primárias quanto as secundárias e por esse motivo possui campos para os atributos de ambos os tipos de estruturas, como mostra a Figura 3.28.

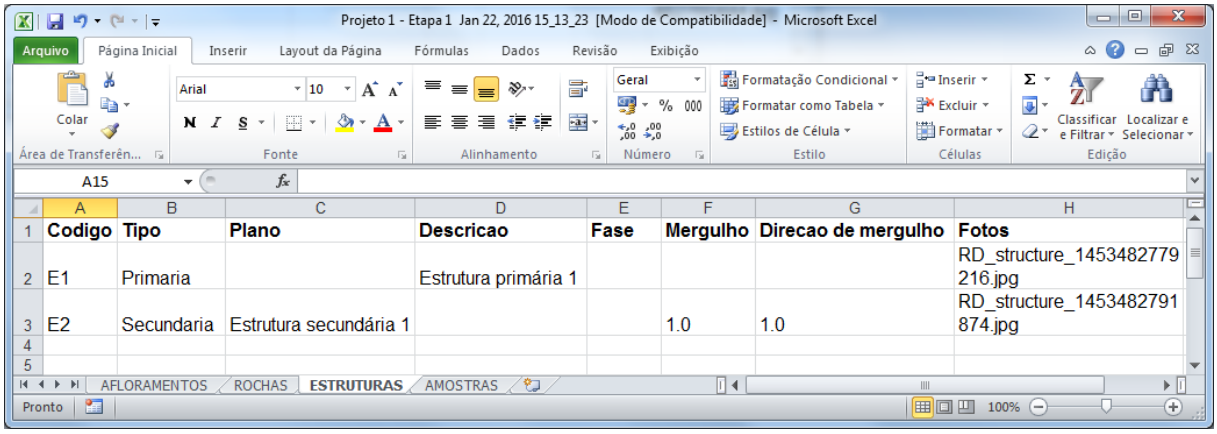

Figura 3.28: Planilha de Estruturas Primárias e Secundárias.

A planilha de amostras contém sua descrição e fotos e pode ser vista na Figura 3.29.

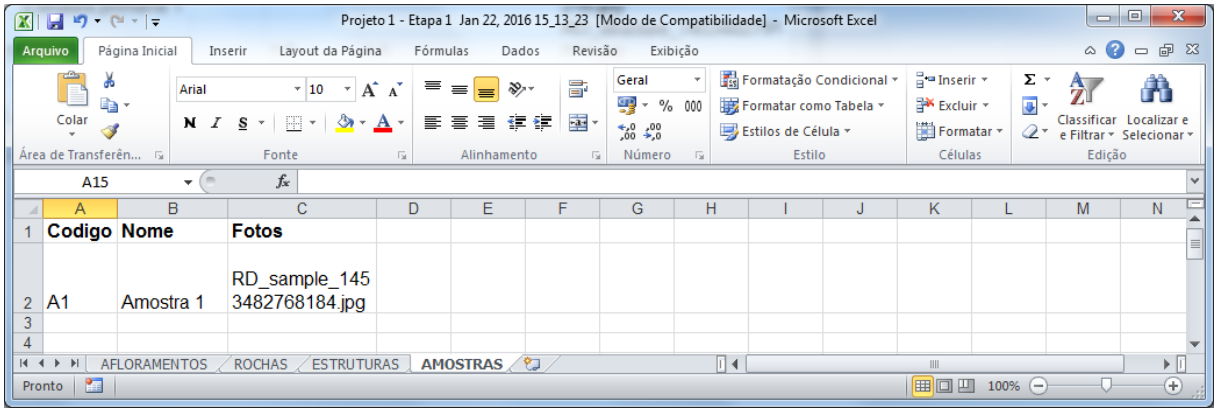

Figura 3.29: Planilha de Amostras.

Criação de projetos

Quando o usuário escolhe criar um novo projeto, o aplicativo o redireciona para a tela de criação de projeto, mostrada na primeira imagem da Figura 3.30. Tendo em vista o modelo de dados, foi definido que não existiria um projeto sem uma etapa de campo. Por esse motivo, quando o usuário cria um projeto, ele é obrigado a criar também uma etapa de campo (segunda imagem da Figura 3.30). Uma vez que o usuário salva o formulário, é exibida a tela de listagem de etapas de campo (última imagem da Figura 3.30).

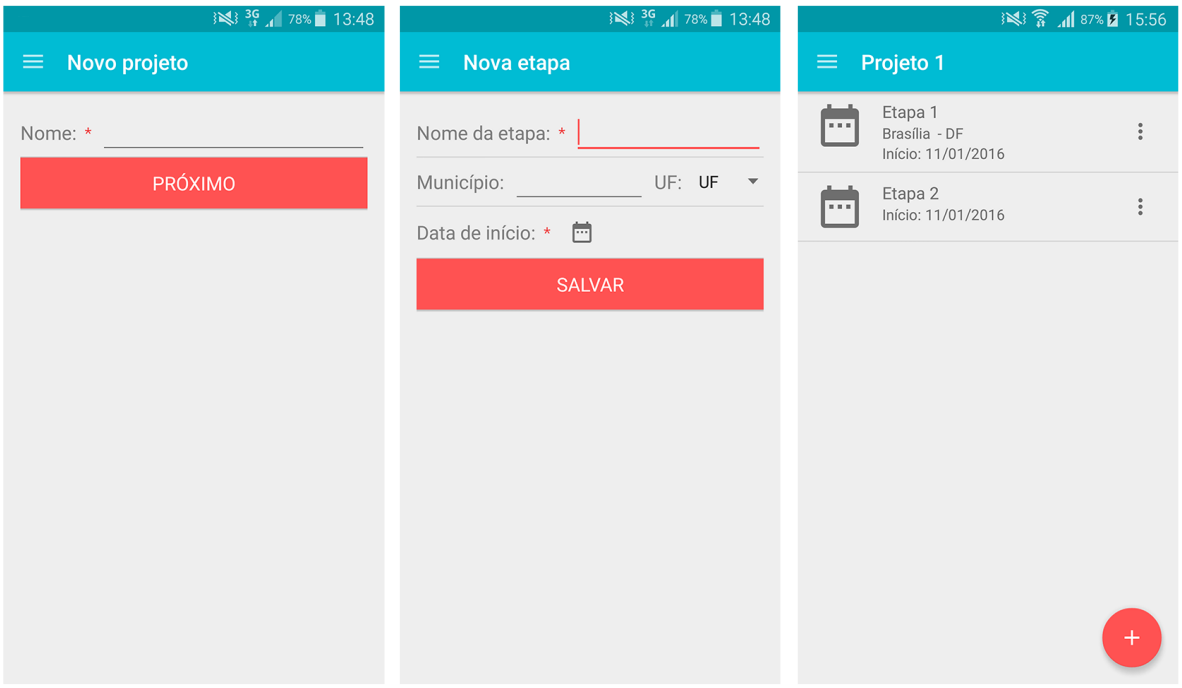

Figura 3.30: Telas de Criação de Projetos e Etapas.

Criação de afloramentos

Após o usuário escolher uma etapa de campo da lista mostrada anteriormente, o aplicativo tenta buscar os afloramentos já existentes naquela etapa de campo. As duas

opções possíveis para a tela com os resultados desta busca encontram-se na Figura 3.31.

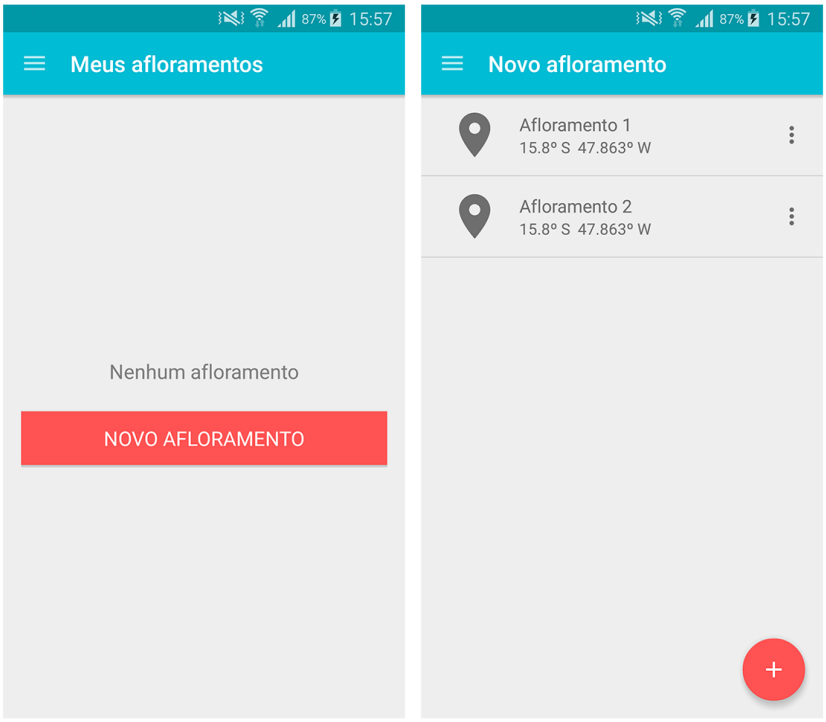

Figura 3.31: Tela de Listagem de Afloramentos.

Se o usuário escolher criar um novo afloramento, a tela de criação de afloramentos será mostrada (primeira tela da Figura 3.32). A segunda tela é exibida quando o usuário seleciona a opção "UTM" nas Coordenadas. Uma vez que o usuário salva o formulário, o aplicativo mostra a tela de informações do afloramento, exibida nas duas últimas imagens da Figura 3.32.

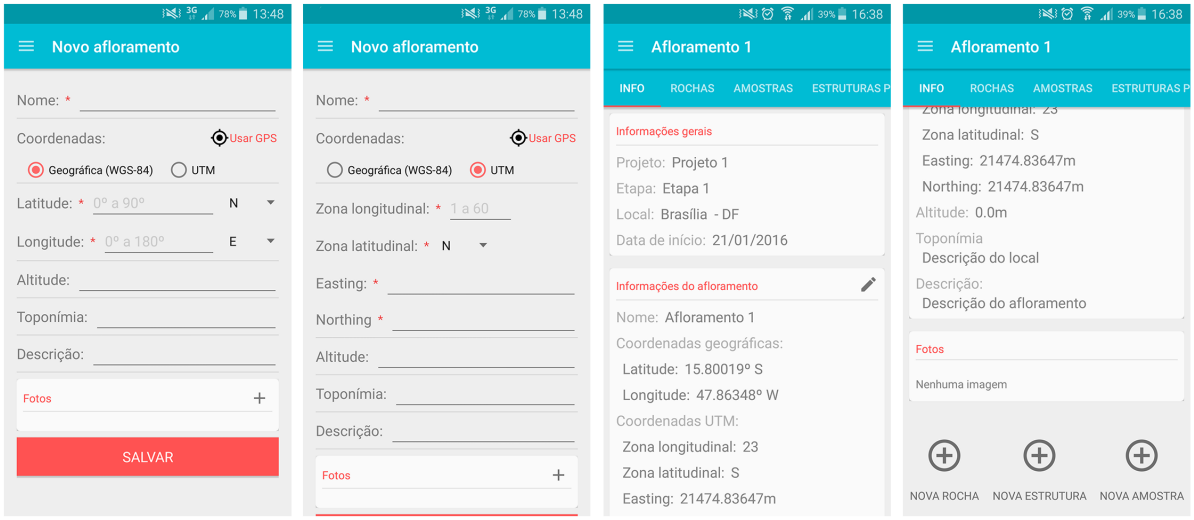

Figura 3.32: Telas de Criação e de Informações do Afloramento.

Listas de rochas, amostras e estruturas

A tela de informações do afloramento possui uma aba para cada entidade pertencente ao afloramento (Rocha, Amostra, Estrutura primária e Estrutura secundária). Cada aba contém uma lista de com as entidades recuperadas do banco de dados. Caso as listas estejam vazias, as telas que aparecem são como as da Figura 3.33.

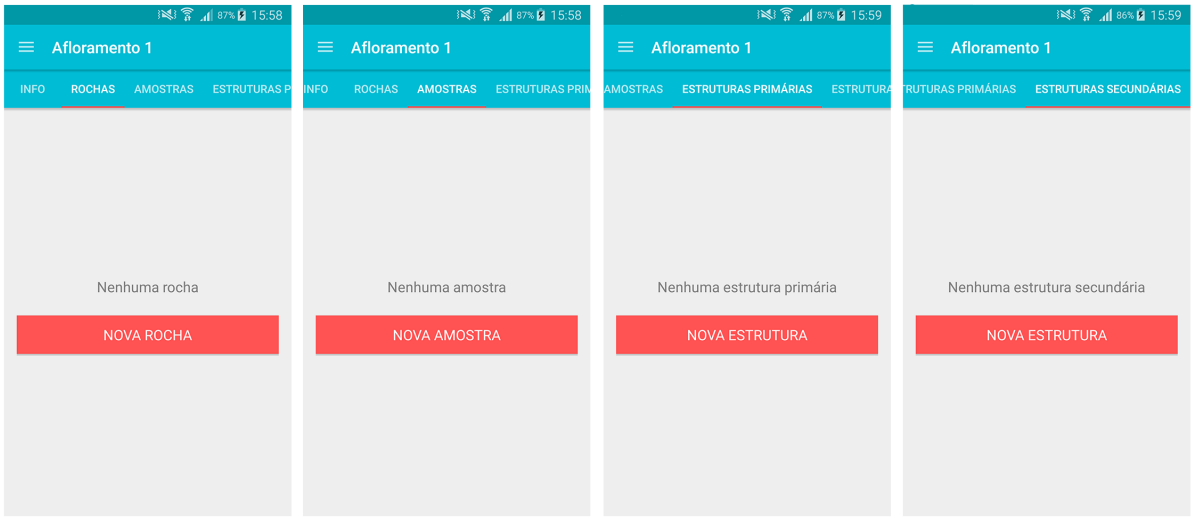

Figura 3.33: Listas Vazias.

Caso a busca no banco retorne resultados, as listas preenchidas são como as da Figura 3.34.

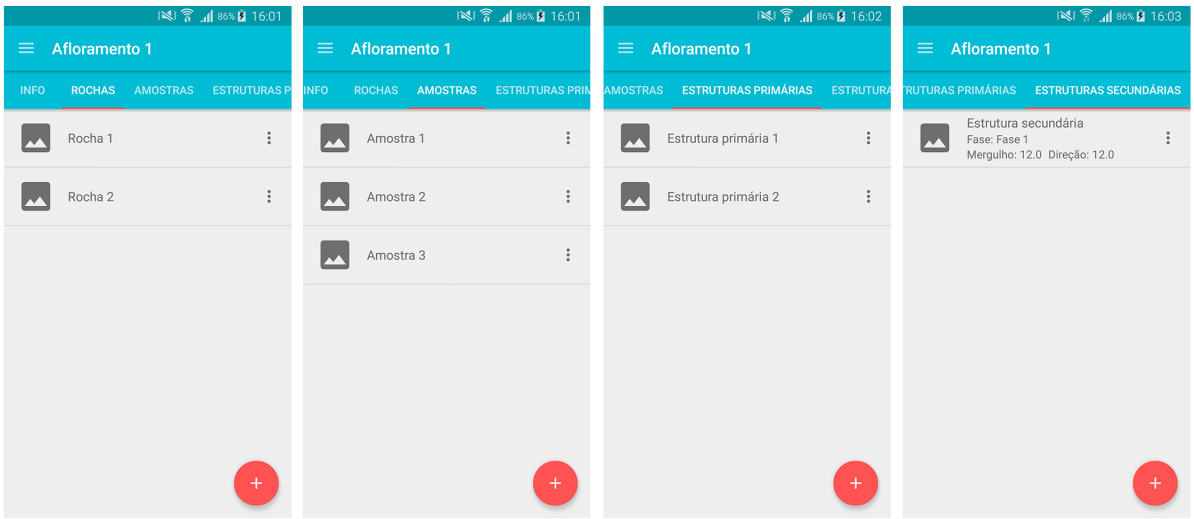

Figura 3.34: Listas Preenchidas.

Criação de rochas

Existem três tipos de rocha, cada uma com seus atributos específicos, e o formulário se adapta a cada um desses tipos. Na Figura 3.35 pode-se ver os formulários para criação de rochas sedimentares, ígneas e metamórficas, respectivamente.

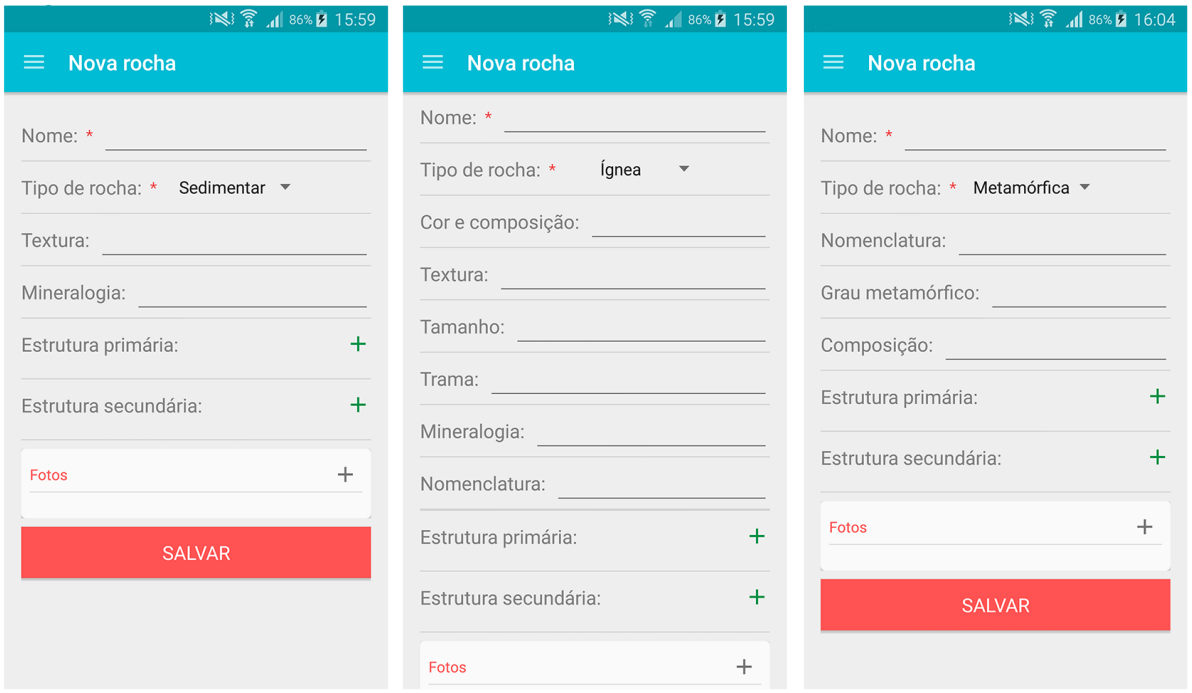

Figura 3.35: Telas de Criação de Rochas.

Criação de amostras

As amostras possuem um formulário de criação bem simples, com campos apenas para a inserção do nome e de fotos da amostra, como pode ser visto na Figura 3.36.

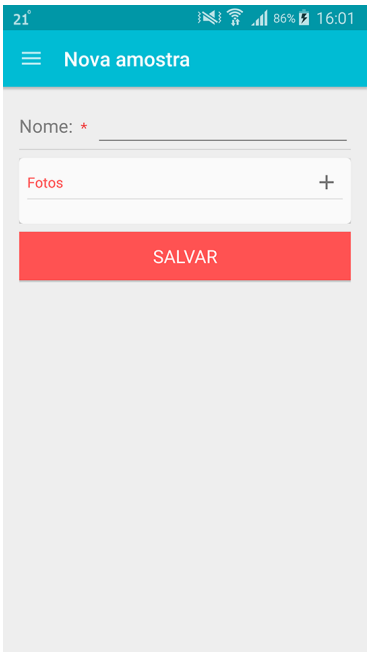

Figura 3.36: Tela de Criação de Amostras.

Criação de estruturas

As estruturas podem ser primárias ou secundárias e seus respectivos formulários encontram-se na Figura 3.37. Estruturas primárias possuem apenas uma descrição e fotos, enquanto as estruturas secundárias possuem um formulário mais detalhado, com campos como tipo de plano, mergulho e direção de mergulho.

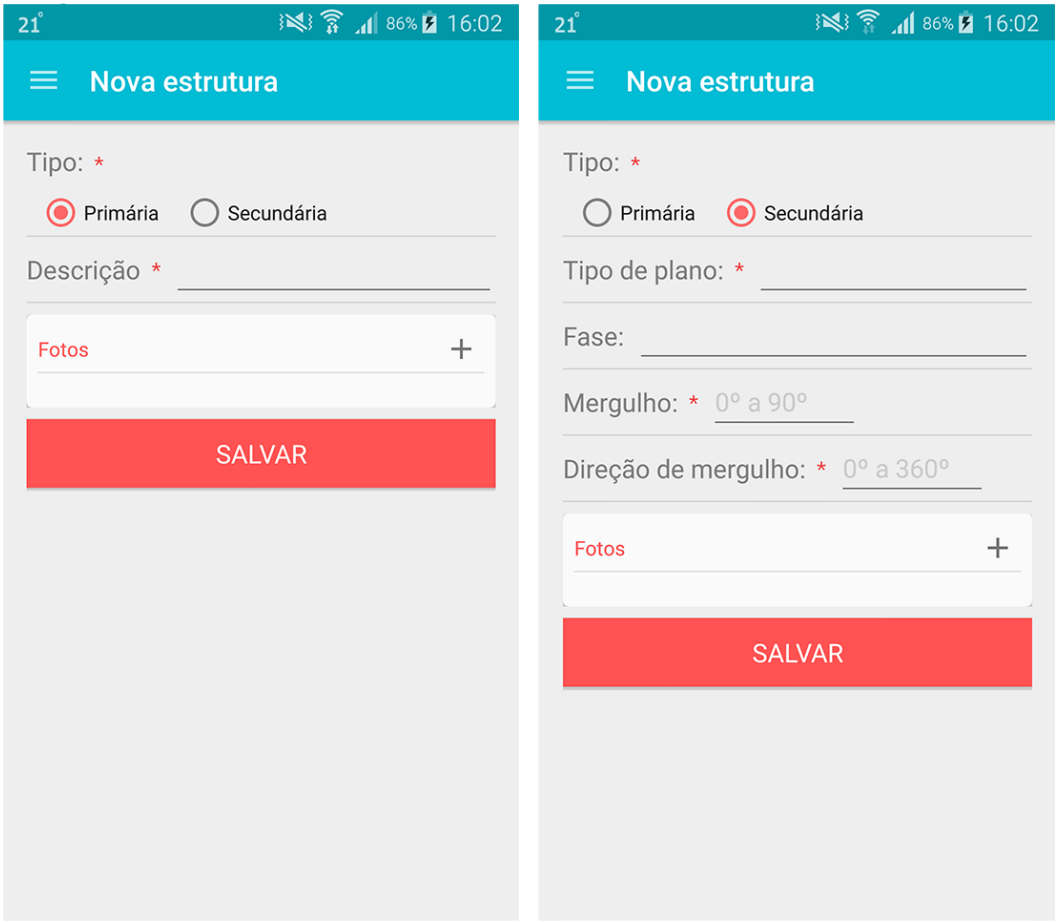

Figura 3.37: Telas de Criação de Estruturas Primárias e Secundárias.

Capítulo 4

Prova de Conceito

Neste capítulo são descritas as metodologias de testes de aceitação do aplicativo e de validação de sua usabilidade. Os testes de aceitação foram categorizados em testes de validação dos dados, testes de conexão e testes de sincronização. A metodologia de validação da usabilidade do aplicativo, descrita com detalhes neste capítulo, foi realizada em conjunto com um grupo restrito de usuários. Para validar o aplicativo, estes usuários simularam o seu uso, avaliando as funcionalidades do sistema.

4.1 Testes

Os testes de validação dos dados foram categorizados desta forma porque demonstram a resposta do aplicativo quando o usuário insere dados inválidos, incluindo as mensagens de erro esperadas. Os testes de conexão mostram o comportamento esperado quando há uma limitação na conectividade durante o uso do *app*. Finalmente, os testes de sincronização apresentam com detalhes o funcionamento esperado da sincronização dos dados.

4.1.1 Ambiente Computacional de Execução dos Testes

Para realizar os testes iniciais no aplicativo, foram utilizados dois dispositivos móveis:

- Samsung Galaxy S5
  - Sistema operacional: *Android* 5.1.1 (*Lollipop*)
  - Memória RAM: 2GB
  - Memória interna: 16GB
  - Processador: 2.5 GHz Quad-Core
- Samsung Galaxy Note 3
  - Sistema operacional: *Android* 5.0 (*Lollipop*)
  - Memória RAM: 3GB
  - Memória interna: 32GB
  - Processador: 2.3 GHz Quad-Core

Para hospedar o serviço *web* durante os testes, o servidor utilizado possui as seguintes características:

- Sistema operacional: Windows Server 2008 R2
- Memória RAM: 16GB
- Processador: Intel Core i7
- Disco rígido: 1TB
- SGBD: PostgreSQL versão 9.1

Os testes que foram realizados são descritos nas próximas seções deste capítulo.

4.1.2 Testes de Validação dos Dados

Para verificar se a validação dos dados está correta, foram realizadas tentativas de cadastramento de entidades (projetos, etapas, afloramentos, rochas, amostras e estruturas) com dados de entrada variados. Cada entidade possui alguns atributos que devem ser validados conforme as regras a seguir:

- Projeto:
  - Nome: campo obrigatório.
- Etapa de campo:
  - Nome: campo obrigatório.
  - Data de início: campo obrigatório.
- Afloramento:
  - Nome: campo obrigatório.
  - Coordenadas geográficas:
    - Latitude: campo obrigatório. Deve ser maior que 0 e menor que 90.
    - Longitude: campo obrigatório. Deve ser maior que 0 e menor que 180.
  - Coordenadas UTM:
    - Zona longitudinal: campo obrigatório. Deve ser maior que 1 e menor que 60.
    - Zona latitudinal: campo obrigatório.
    - *Easting*: campo obrigatório. Deve ser maior que 166,000 e menor que 834,000.
    - *Northing*: campo obrigatório. Seu valor depende da Zona latitudinal. Se esta for Norte, varia de 0 a 9,350,000. Se for Sul, varia de 1,100,000 a 10,000,000.
- Rocha:

  - Nome: campo obrigatório.
  - Tipo de rocha: campo obrigatório.
- Amostra:
  - Nome: campo obrigatório.
- Estrutura primária:
  - Descrição: campo obrigatório.
- Estrutura secundária:
  - Tipo de plano: campo obrigatório.
  - Mergulho: campo obrigatório. Deve ser maior que 0 e menor que 90.
  - Direção de mergulho: campo obrigatório. Deve ser maior que 0 e menor que 360.

Para os atributos que são obrigatórios, os testes consistiam em deixar seus campos vazios em seus respectivos formulários e tentar salvar a entidade. O aplicativo não permitiu que os formulários fossem salvos, exibindo mensagens de erro para os campos em branco com a mensagem "Campo obrigatório" até que eles fossem preenchidos. Para os atributos numéricos que possuem limites inferiores e superiores definidos, os testes envolveram a inserção de números fora do alcance de cada um. O aplicativo exibiu mensagens de erro específicas para cada atributo, informando ao usuário quais eram os valores permitidos. O formulário só pôde ser salvo quando valores válidos foram inseridos.

Na Figura 4.1, é mostrado um exemplo de validação dos dados de um afloramento. Na primeira tela, todos os campos obrigatórios exibem erro quando são deixados em branco. Na segunda tela, é mostrada a mensagem de validação do campo latitude e na terceira, do campo longitude.

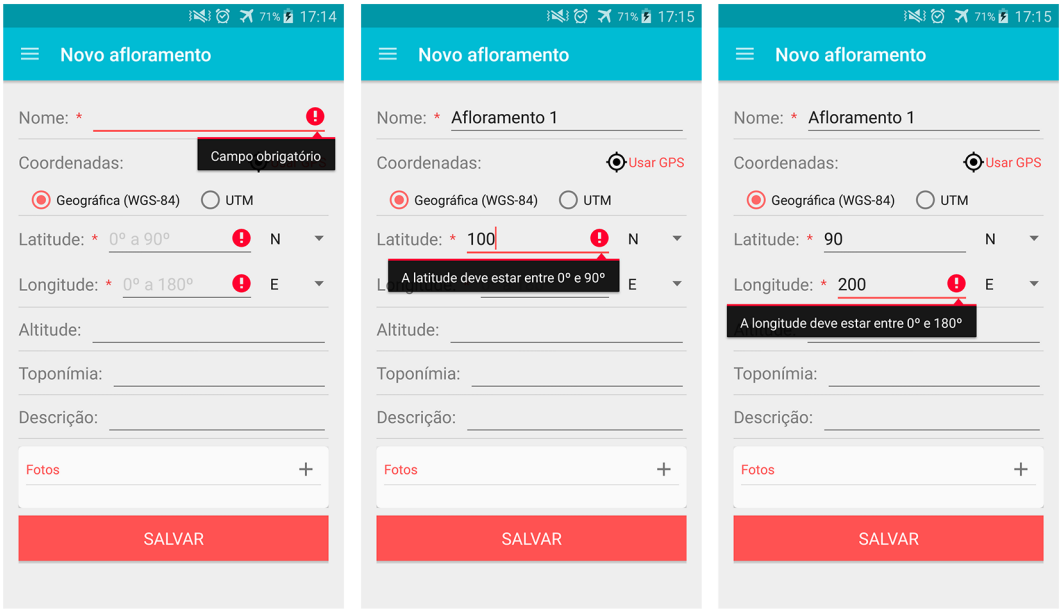

Figura 4.1: *Screenshots* do Aplicativo Realizando Validação dos Dados.

4.1.3 Testes de Conexão

Os testes de conexão tinham como objetivo verificar o comportamento do sistema quando funcionalidades que exigem conexão com a *Internet* eram realizadas sem acesso à rede. Os seguintes testes foram realizados:

- Teste 1: Fazer *login* no aplicativo sem conexão com a *Internet*;
- Teste 2: Cadastrar-se no aplicativo sem conexão com a *Internet*;
- Teste 3: Fazer *download* do mapa sem conexão com a *Internet*;
- Teste 4: Interromper a conexão durante o *download* do mapa.

Teste 1

Para realizar este teste, os seguintes passos foram realizados: na tela de *login*, foram informadas credenciais válidas (já cadastradas no sistema) nos campos de email e senha e o botão "Entrar" foi pressionado. O resultado obtido foi a mensagem de erro "Conexão indisponível" na parte inferior da tela (ver primeira tela da Figura 4.2).

Teste 2

Para este teste, foram inseridos dados válidos no formulário de cadastro de usuário e foi pressionado o botão "Cadastrar". O aplicativo então exibiu a mensagem "Conexão indisponível" (ver segunda tela da Figura 4.2).

Teste 3

Este teste foi realizado de três formas distintas, tendo em vista que há três formas de acessar o mapa no aplicativo. A primeira forma é a partir da tela de *login*, selecionando-se o botão "Mapa". A segunda forma é após efetuado o *login* como Visitante, acessando o menu lateral da aplicação e escolhendo a opção "Mapa". A terceira forma é efetuando *login* como um usuário cadastrado e acessando o mapa também pelo menu lateral, como na opção anterior.

Em todos os casos, a tela de *download* do mapa foi exibida, mas o botão "Baixar mapa" encontrava-se inativo e a mensagem "Conexão indisponível" era mostrada (ver terceira tela da Figura 4.2). Se nesse momento a conexão for recuperada, a mensagem de erro some e o botão é ativado.

Teste 4

Neste teste, foi iniciado o *download* do mapa normalmente, com o celular conectado à rede, e durante a transferência a conexão foi interrompida. Ao selecionar "Baixar mapa", o aplicativo exibiu a mensagem "Processando *download*...". Logo depois, o *download* foi iniciado. Após 20% do mapa baixados, o "Modo avião" foi ativado e a conexão foi interrompida. Poucos segundos depois, o aplicativo reconheceu que não tinha acesso à rede e pausou o *download*, exibindo a mensagem "Conexão indisponível" (ver quarta tela da Figura 4.2). Uma vez desligado o "Modo avião", levou poucos segundos para o aplicativo remover a mensagem de erro e continuar o *download* de onde havia parado.

A Figura 4.2 mostra os resultados obtidos nos Testes 1, 2, 3 e 4 descritos, respectivamente:

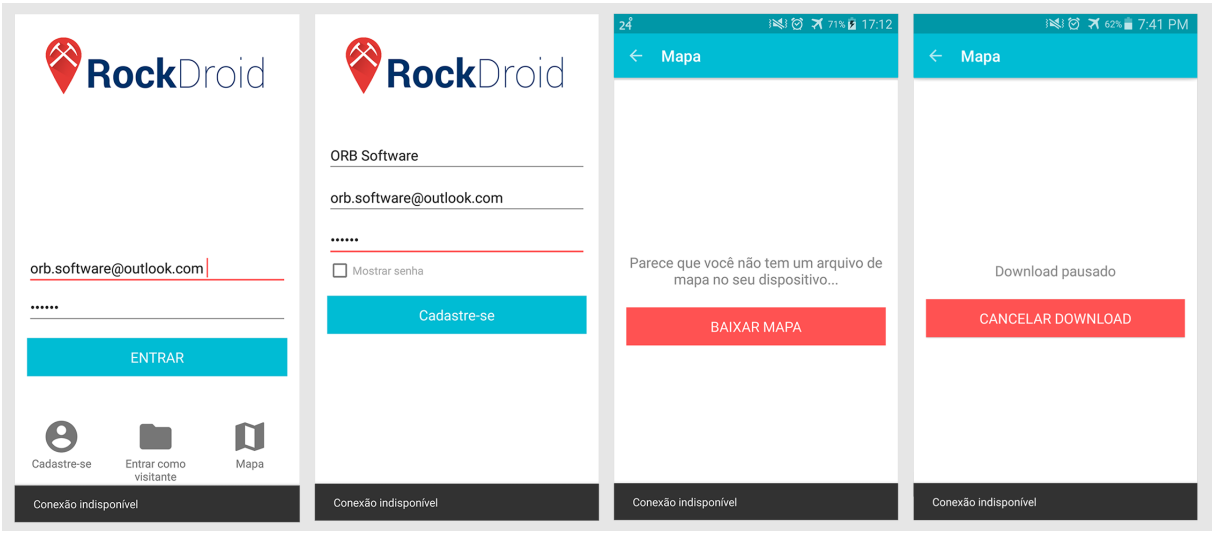

Figura 4.2: Resultados Obtidos.

4.1.4 Testes de Sincronização

Os testes de sincronização também envolveram testes de conexão, mas foram agrupados nesta seção para melhor organização do trabalho. Primeiramente foi testada a sincronização com acesso à *Internet*. O segundo teste foi para verificar qual seria o comportamento do aplicativo caso a conexão fosse interrompida durante uma sincronização. Já o terceiro teste foi a realização da sincronização sem *Internet*.

Para todos os testes, foi sincronizado apenas um projeto contendo uma etapa de campo, um afloramento com uma foto, uma rocha com uma foto, uma amostra com uma foto, uma estrutura primária com uma foto e uma estrutura secundária com uma foto.

Sincronização com acesso à *Internet*

A sincronização se inicia quando um usuário logado escolhe a opção "Sincronizar" no menu lateral. Uma mensagem de confirmação é exibida e informa sobre o tempo que a sincronização pode levar e que taxas podem ser aplicadas (caso o usuário deseje usar esta funcionalidade em redes de dados móveis, por exemplo), como é mostrado na primeira tela da Figura 4.3. Uma vez que o usuário confirma a ação, a sincronização se inicia e o aplicativo mostra uma mensagem na parte inferior da tela, bem como adiciona uma notificação indicando que a sincronização está em andamento e mostrando ao usuário qual entidade está sendo sincronizada no momento, como pode ser visto na segunda e na terceira tela da Figura 4.3. Quando o processo é completado com sucesso, o aplicativo informa o usuário com uma mensagem e uma notificação informando "Sincronização bem sucedida!" (última tela da Figura 4.3).

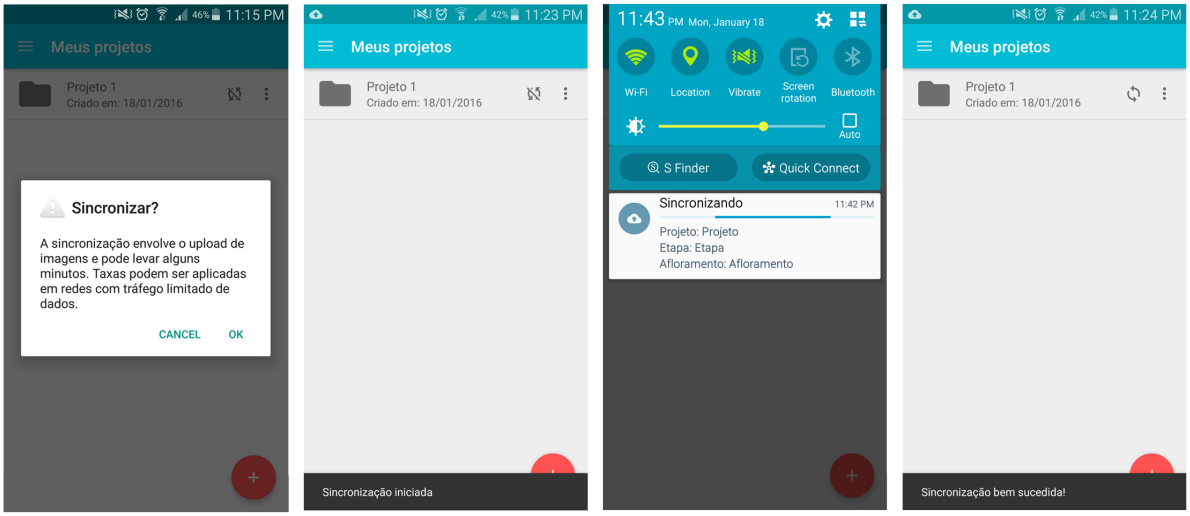

Figura 4.3: Telas de uma Sincronização Bem Sucedida.

O resultado da sincronização pode ser visualizado no banco de dados do servidor. A Figura 4.4 apresenta o resultando de uma consulta sobre a tabela de afloramentos após a realização do teste de sincronização bem sucedido. Pode-se observar que o último registro listado corresponde ao afloramento sincronizado durante o teste.

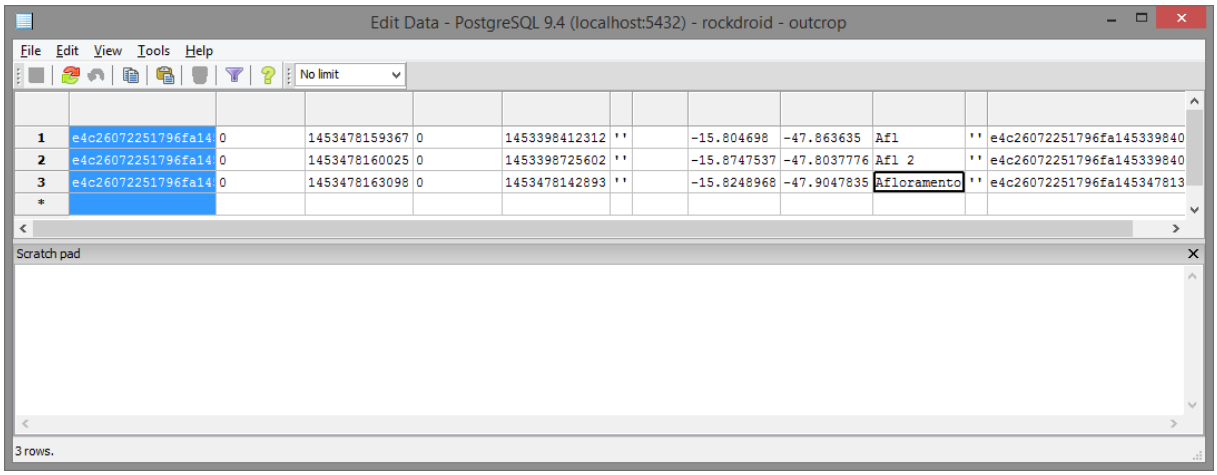

Figura 4.4: Resultado de uma Consulta na Tabela de Afloramentos no Banco de Dados do Servidor.

Conexão interrompida durante a sincronização

Uma vez iniciada a sincronização do projeto, o acesso à *Internet* foi interrompido através da opção "Modo avião" do celular. Imediatamente o aplicativo interrompeu a sincronização e informou a mensagem "Falha na sincronização", como na Figura 4.5. O celular foi então reconectado, mas a sincronização não foi reiniciada ou continuada. Para fazer isso, é necessário solicitar novamente a sincronização dos dados através da opção no menu lateral.

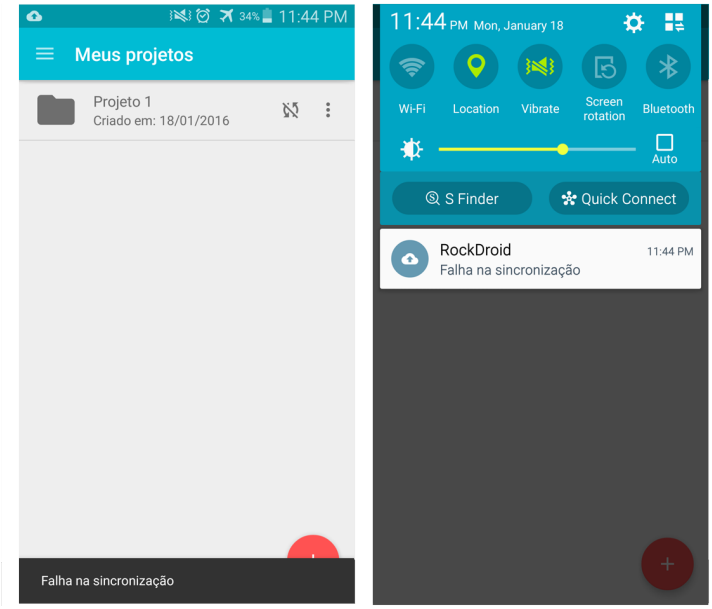

Figura 4.5: Mensagem de Falha na Sinronização.

Sincronização sem acesso à *Internet*

Quando não há acesso à *Internet*, a sincronização não pode ser realizada. Neste caso, o aplicativo mostra a mensagem de confirmação ao usuário (primeira tela da Figura 4.3) e só depois verifica se há conexão. Uma vez constatado que o dispositivo está *offline*, a mensagem de falha na sincronização é mostrada, como na Figura 4.5.

4.2 Validação

Após obter resultados satisfatórios nos testes básicos mencionados acima, foi criada uma conta na plataforma de aplicativos *Google Play* e o aplicativo foi disponibilizado para testes.

Esta disponibilização foi restrita a um grupo seleto de pessoas que se propuseram a realizar a validação do aplicativo. Foi criado um grupo fechado na rede social *Facebook* para os testadores com informações sobre como baixar o aplicativo no *Google Play* e com um *link* para o formulário de usabilidade, que pode ser visto na Figura 4.6.

Figura 4.6: Grupo do *Facebook* Criado para os Testadores do Aplicativo.

Este grupo era composto por especialistas da área, entre os quais estavam alunos de geologia, geólogos já formados e pesquisadores de áreas relacionadas, como a geofísica, que também atuam em pesquisas de campo. Trinta e duas pessoas se cadastraram, mas ao todo foram obtidas avaliações de apenas dez indivíduos.

A validação consistia em um processo de dois passos: no primeiro passo o usuário instalava o aplicativo e testava suas funcionalidades, simulando o uso em um ambiente de campo. No segundo passo, o usuário preenchia um formulário de usabilidade no qual ele fornecia suas opiniões sobre a facilidade de uso, a utilidade do sistema, e provia sugestões de melhorias para as próximas versões.

O formulário se dividia em duas seções: a primeira seção tinha o objetivo de traçar o perfil do usuário (Figura 4.7) e a segunda, de recolher seu *feedback* sobre o aplicativo

e suas funcionalidades (Figuras 4.8 e 4.9). O resultado da compilação das respostas dos formulários apresentou as características apresentadas a seguir.

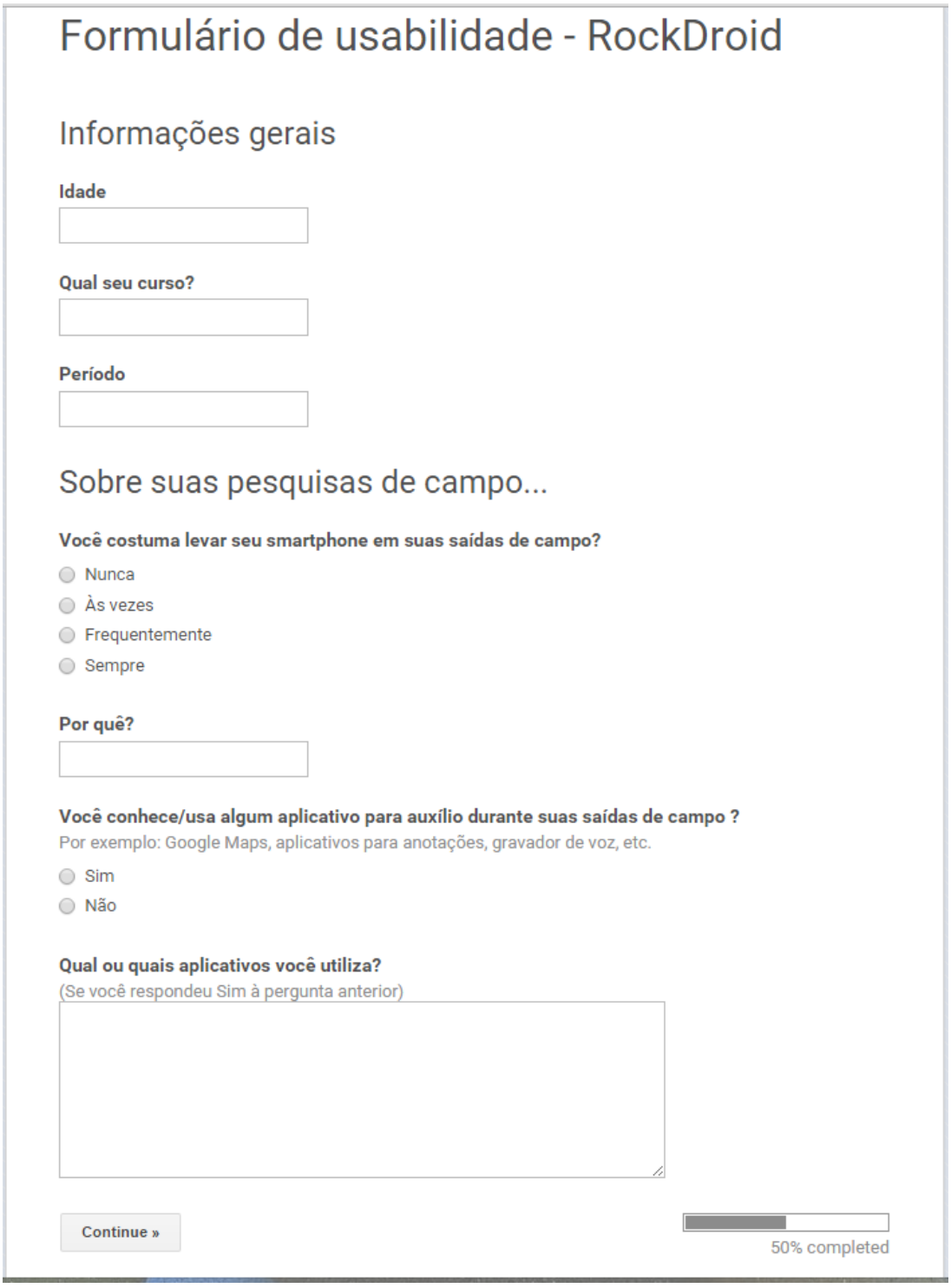

Figura 4.7: Primeira Seção do Formulário.

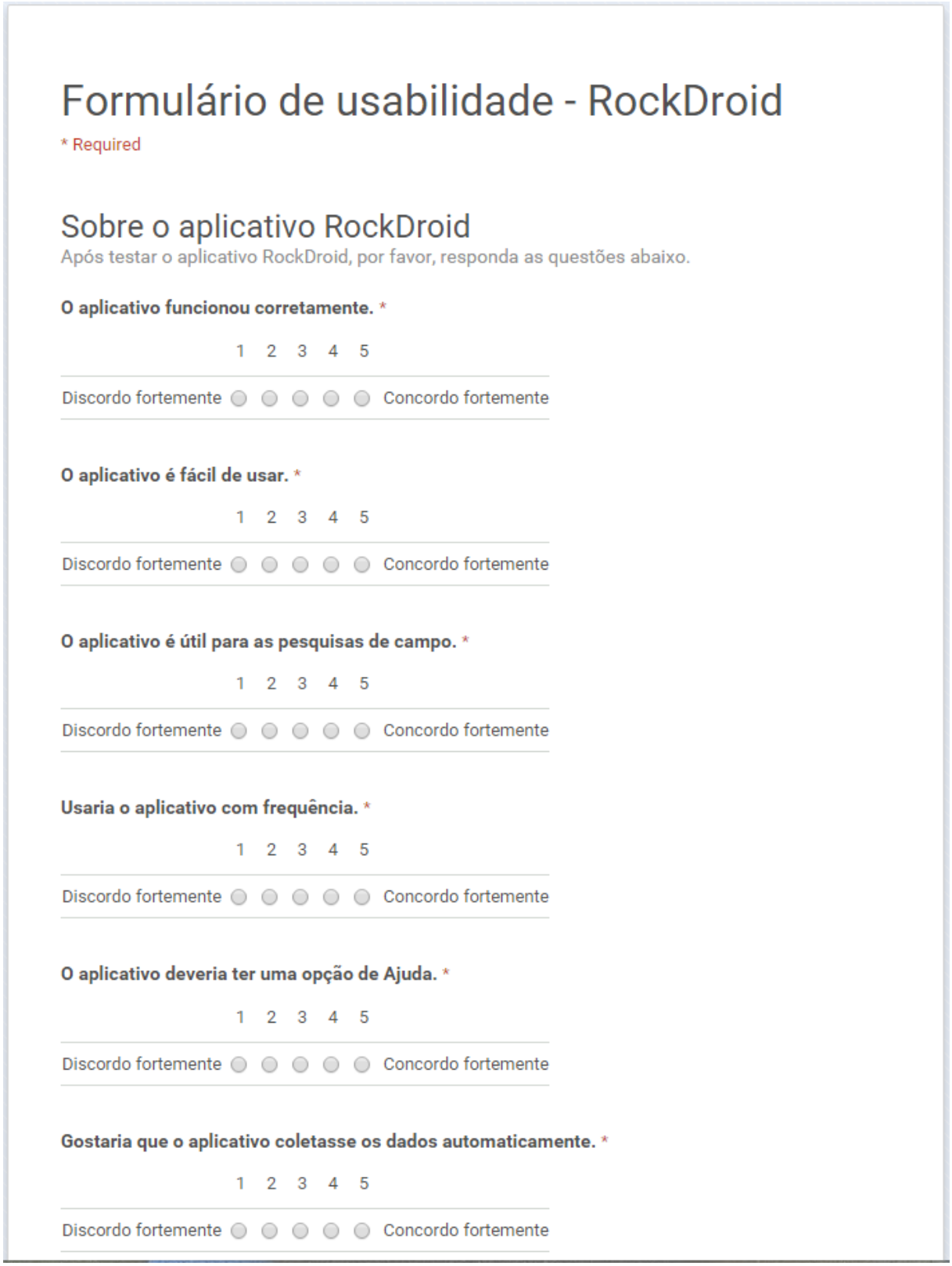

Figura 4.8: Segunda Seção do Formulário (parte 1).

Gostei da organização do aplicativo e da divisão dos dados em projetos, etapas, afloramentos, rochas, estruturas e amostras. *

	1	2	3	4	5	
Discordo fortemente	○	○	○	○	○	Concordo fortemente

O aplicativo tem poucas funcionalidades, não substitui completamente o método caneta/papel.

	1	2	3	4	5	
Discordo fortemente	○	○	○	○	○	Concordo fortemente

Perco mais tempo inserindo dados no aplicativo que anotando em meu bloco de notas. *

	1	2	3	4	5	
Discordo fortemente	○	○	○	○	○	Concordo fortemente

Que instrumentos/ferramentas você utiliza em suas saídas de campo que gostaria que fossem implementados no aplicativo, se possível?

Exemplos: bússola, clinômetro, GPS, câmera, bloco de notas, gravador de voz, etc.

Possui sugestões para a evolução do aplicativo?

Observações e críticas também são úteis para nós.

« Back Submit

100%: You made it.

Never submit passwords through Google Forms.

Figura 4.9: Segunda Seção do Formulário (parte 2).

4.2.1 Perfil dos Usuários

A primeira parte da seção de perfil do usuário do questionário procurava descobrir, inicialmente, informações gerais sobre a experiência do usuário em relação às pesquisas de campo. Por isso, incluía perguntas sobre a idade do pesquisador, o curso e o período que o pesquisador cursava. O restante das perguntas desta seção visava entender o comportamento do usuário em relação a *smartphones*, incluindo perguntas que indicavam a frequência de uso do aparelho em campo e o costume de utilizar aplicativos para auxiliar na coleta dos dados.

Apesar da amostragem relativamente pequena, a idade dos participantes da avaliação variou entre 19 e 45 anos, o que representa um conjunto de pesquisadores com níveis de experiência distintos. Um dos pesquisadores pertencia ao curso de Geofísica, enquanto os demais eram da Geologia. No grupo de testadores haviam três pesquisadores já formados, e os demais estavam entre o quarto e o último período do curso.

Conforme apresenta a Figura 4.10, oito entre os dez testadores responderam que sempre levam seus *smartphones* para as pesquisas de campo. O usuário que selecionou a opção "Frequentemente" justificou afirmando que, apesar de ser útil para tirar fotos e usar o GPS, ele tem medo de quebrar, perder ou molhar o celular durante uma chuva. O outro usuário, que escolheu a opção "Às vezes", deu o motivo de não haver conectividade nas áreas das pesquisas.

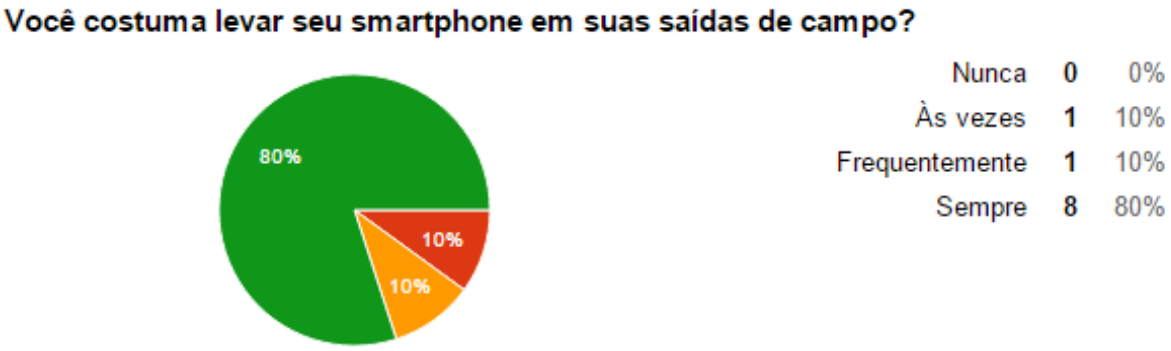

Figura 4.10: Gráfico Resultante das Respostas da Primeira Questão do Formulário.

Quando questionados sobre o uso de outros aplicativos para auxiliar durante a pesquisa, 70% dos usuários responderam positivamente. A Figura 4.11 mostra o gráfico relativo à esta pergunta. Para exemplificar, estes usuários indicaram os seguintes aplicativos: *Google Maps*[1], *Gravador de voz avançado*[2], *ViewRanger GPS*[3], *Google Earth*[4] e *Weather Signal*[5]. Foram citadas também as funções nativas do aparelho como bloco de notas, câmera e GPS.

[1] https://play.google.com/store/apps/details?id=com.google.android.apps.maps
[2] https://play.google.com/store/apps/details?id=com.enlightment.voicerecorder
[3] https://play.google.com/store/apps/details?id=com.augmentra.viewranger.android
[4] https://play.google.com/store/apps/details?id=com.google.earth
[5] https://play.google.com/store/apps/details?id=com.opensignal.weathersignal

Figura 4.11: Gráfico Resultante das Respostas da Segunda Questão do Formulário.

4.2.2 Usabilidade do Aplicativo

A segunda seção do formulário possuía nove questões sobre a usabilidade e utilidade do sistema e duas questões finais que pediam sugestões de melhorias e funcionalidades para o aplicativo. As primeiras nove questões apresentavam afirmações e aceitavam respostas valoradas entre um e cinco, em que o valor mais baixo significava uma forte discordância e o valor mais alto, uma forte concordância em relação à afirmação apresentada.

A primeira questão desta seção procurava saber se o testador conseguiu usar o aplicativo sem encontrar defeitos. A Figura 4.12 demonstra que 50% dos testadores obtiveram sucesso em todas as funcionalidades testadas. Os demais concordaram que o aplicativo funcionou, porém não ficaram completamente satisfeitos com o seu funcionamento. No total, 90% das pessoas concordaram parcialmente com o bom funcionamento do aplicativo.

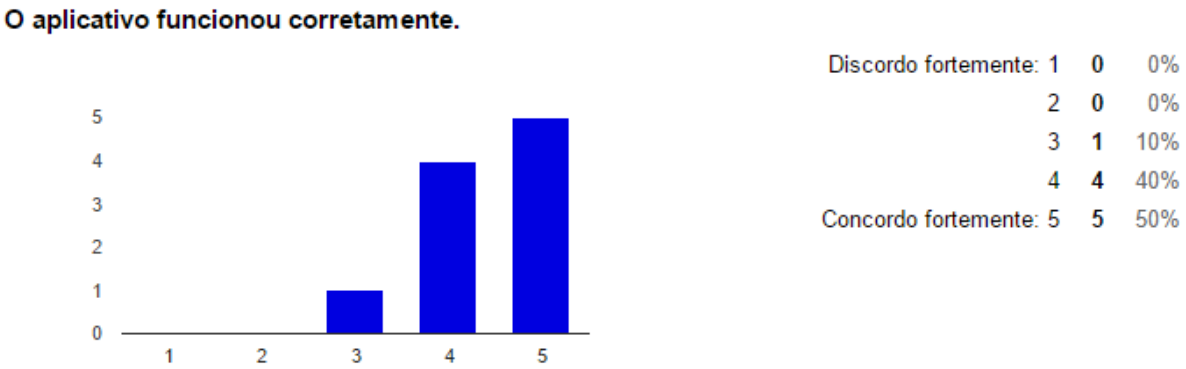

Figura 4.12: Gráfico Resultante das Respostas da Primeira Questão sobre Usabilidade.

A próxima questão visava descobrir a opinião dos testadores sobre a facilidade de uso do *app*. Em geral, todos concordaram que o aplicativo é fácil de usar, sendo que dentre estes, 70% concordaram completamente com esta afirmação. A Figura 4.13 mostra que os demais deram o valor quatro para esta afirmação.

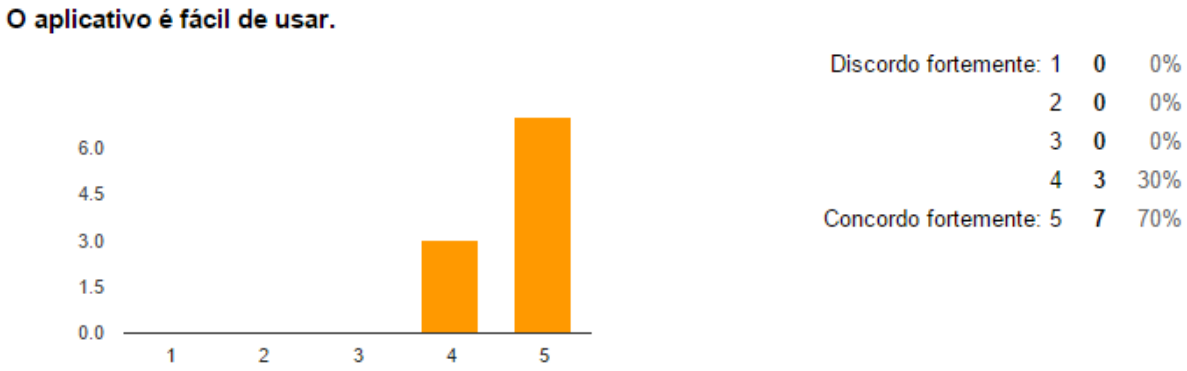

Figura 4.13: Gráfico Resultante das Respostas da Segunda Questão sobre Usabilidade.

A terceira questão desta seção afirmava que o aplicativo é útil para as pesquisas de campo. Conforme apresenta a Figura 4.14, a maioria dos usuários deu valor quatro como resposta. Houve ainda 40% de plena concordância com a afirmação.

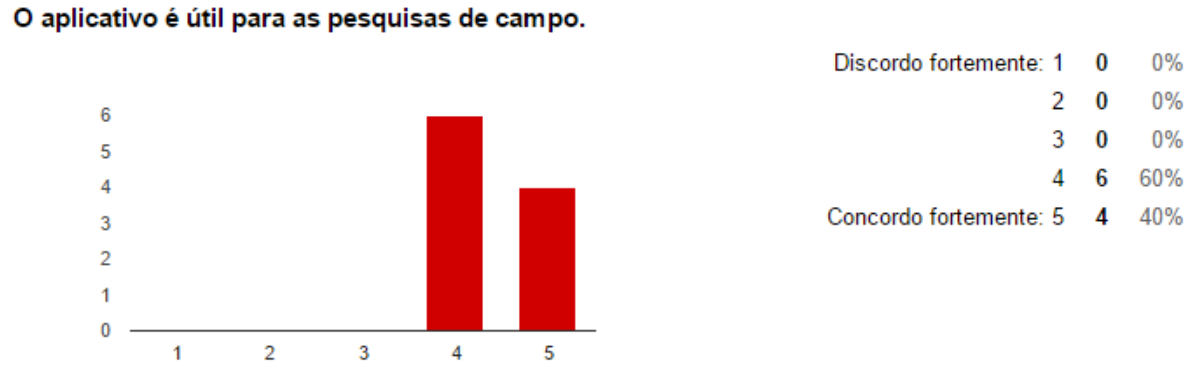

Figura 4.14: Gráfico Resultante das Respostas da Terceira Questão sobre Usabilidade.

A Figura 4.15 demonstra que, em relação à frequência de uso do aplicativo estimada pelos testadores, houve uma separação de opiniões. Enquanto 40% dos usuários concordou plenamente que o usariam com frequência, outros 40% deram uma resposta intermediária, no valor três. Os 20% restantes deram valor quatro.

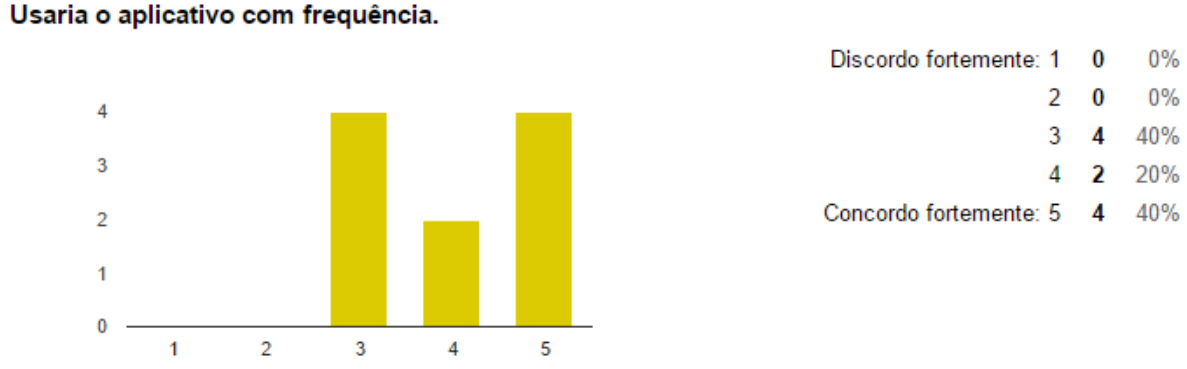

Figura 4.15: Gráfico Resultante das Respostas da Quarta Questão sobre Usabilidade.

Uma das possibilidades de evolução do sistema é a implementação de telas de ajuda, que apresentassem informações sobre conceitos geológicos e dicas sobre como preencher os campos das coletas. Por isto, a quinta pergunta desta seção do formulário foi voltada especificamente a esta funcionalidade. O resultado apurado desta questão, conforme mostra a Figura 4.16, demonstra que a maioria dos usuários discorda parcialmente da utilidade de uma opção de ajuda adicional.

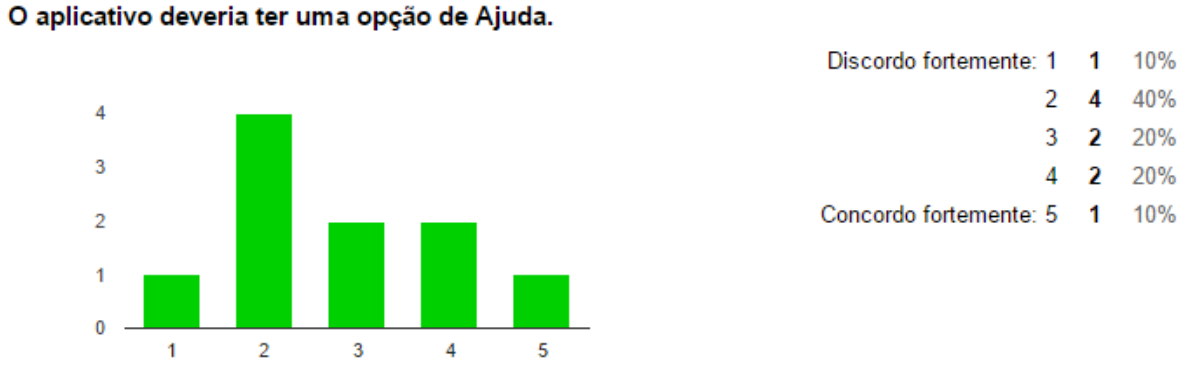

Figura 4.16: Gráfico Resultante das Respostas da Quinta Questão sobre Usabilidade.

Visando facilitar a coleta de dados, outra possível evolução do aplicativo seria uma maior automatização das coletas dos dados. Por isso, os testadores foram questionados sobre a necessidade desta melhoria. A questão introduzia a afirmação *Gostaria que o aplicativo coletasse os dados automaticamente* e as respostas dos usuários distribuíram-se homogeneamente entre os valores três, quatro e cinco, com 30% deles selecionando cada uma destas opções, conforme apresenta a Figura 4.17. Houve ainda um usuário que discordou parcialmente desta afirmação, respondendo o valor dois.

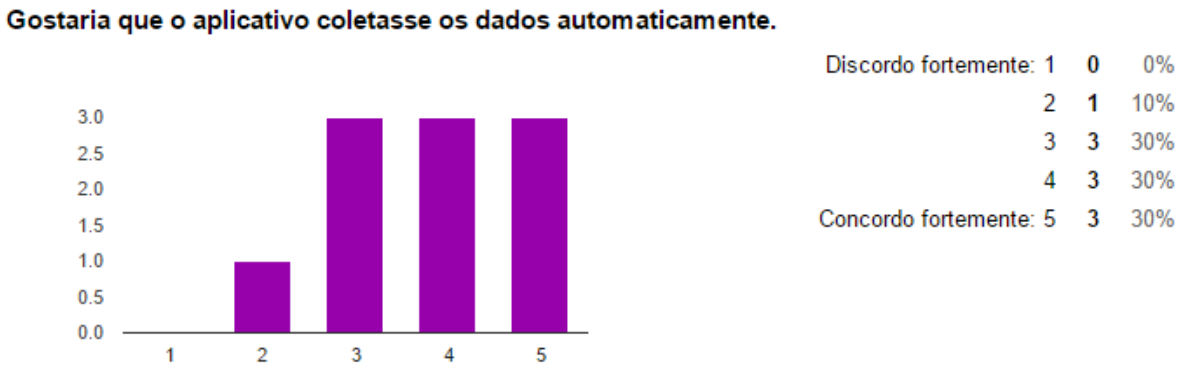

Figura 4.17: Gráfico Resultante das Respostas da Sexta Questão sobre Usabilidade.

Oitenta por cento dos testadores concordaram plenamente que a divisão dos dados em projetos, etapas, afloramentos, rochas, amostras e estruturas foi uma boa decisão. Até mesmo os 20% restantes concordaram parcialmente com esta afirmação, conforme mostra a Figura 4.18

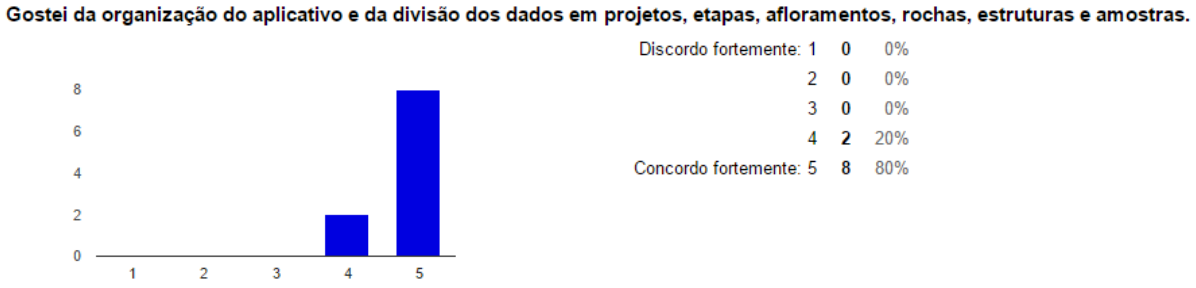

Figura 4.18: Gráfico Resultante das Respostas da Sétima Questão sobre Usabilidade.

O objetivo do aplicativo é a substituição, mesmo que parcial, da utilização da caderneta de papel pelos formulários eletrônicos para a aquisição dos dados das pesquisas em campo. Neste ponto, a questão de número oito desta seção afirmava que *O aplicativo tem poucas funcionalidades, não substitui completamente o método caneta/papel*. A Figura 4.19 mostra que o resultado desta questão foi variado, mas a maioria das respostas foram concentradas nos valores dois e três, demonstrando que, em geral, houve uma discordância parcial desta frase.

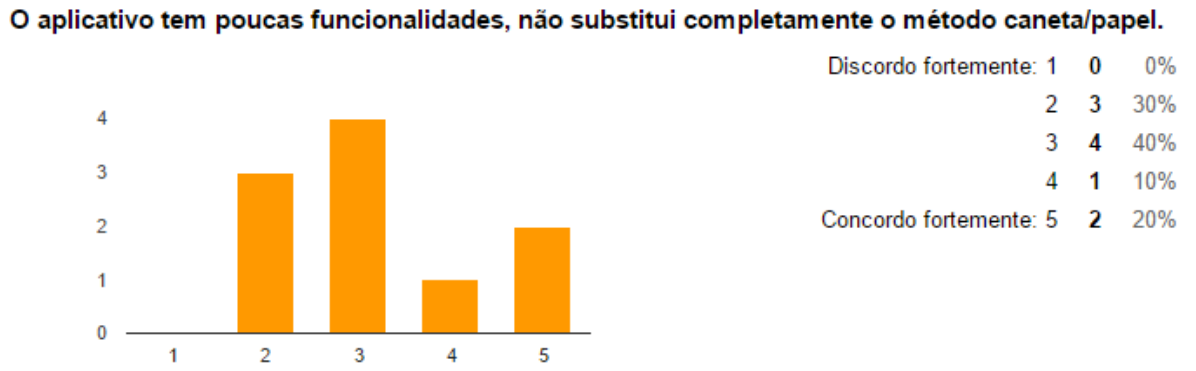

Figura 4.19: Gráfico Resultante das Respostas da Oitava Questão sobre Usabilidade.

A última das nove questões específicas de usabilidade afirmava que o usuário iria gastar mais tempo inserindo dados no aplicativo do que anotando em sua caderneta de papel. A Figura 4.20 apresenta as respostas dadas pelos testadores. Pode-se observar que a maior parte concentrou-se nos valores dois e três, significando uma discordância parcial.

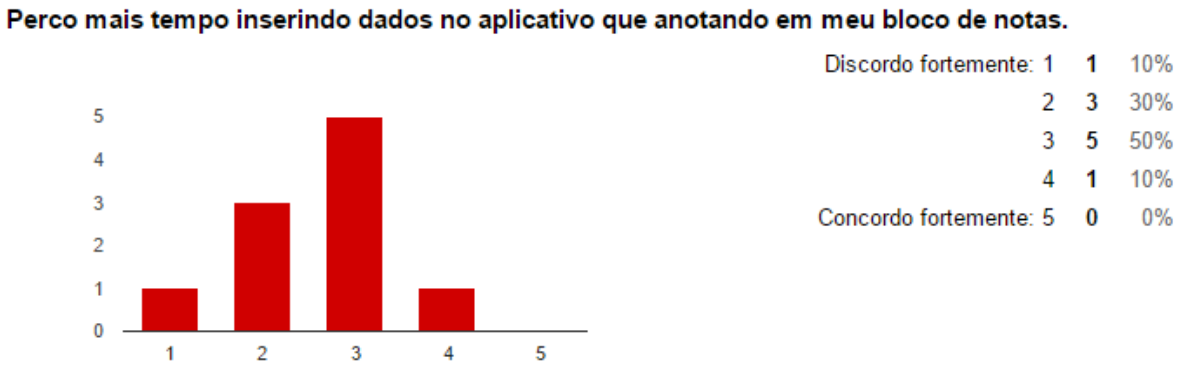

Figura 4.20: Gráfico Resultante das Respostas da Nona Questão sobre Usabilidade.

As duas próximas questões, as últimas do formulário, possuíam caráter descritivo. A primeira pedia ideias de ferramentas úteis para se inserir como novas funcionalidades do aplicativo em versões posteriores. Dentre as respostas mais frequentes, a bússola foi a ferramenta mais sugerida. A gravação de voz e a substituição do mapa comum por um mapa geológico também foram sugeridas por mais de um usuário testador. A adição de imagens aéreas, clinômetro, medidor de temperatura, pressão, umidade e velocidade do vento também foram comentadas.

A última questão pedia por opiniões e sugestões adicionais que os testadores pudessem prover. As sugestões foram bastante variadas. Alguns usuários requisitaram pela funcionalidade de desenho em cima de uma foto. Outros sugeriram automatizar a nomeação das rochas baseando-se em seu tipo e textura. Um dos usuários comentou que seria interessante adicionar as mesmas informações geológicas que vêm por padrão em suas cadernetas de papel. Outra sugestão foi de exportar os dados em um formato compatível com o *Google Earth*. Por fim, um usuário sugeriu que as informações fossem gravadas no cartão de memória externo do dispositivo, ao invés de em sua memória interna.

Capítulo 5

Conclusão

O RockDroid vem para prover uma solução mais uniforme para os geólogos, criando uma interface digital que padroniza a coleta dos dados e torna a posterior análise mais rápida e eficiente. A união de diferentes ferramentas em um único aplicativo (câmera fotográfica, GPS, mapa, bloco de notas, etc.) traz benefícios aos pesquisadores de campo, facilitando seu trabalho.

O desenvolvimento focado na qualidade do código e na conformidade com a arquitetura inicialmente projetada criou uma base bem estruturada e facilmente modificável para comportar novas funcionalidades. A própria arquitetura do aplicativo foi pensada para maximizar o reúso dos componentes e a adição de novos casos de uso, limitando as dependências de *frameworks* externos e organizando o código em módulos e camadas bem definidos.

A construção de um serviço *web RESTful* forneceu a segurança necessária aos dados transmitidos, além de facilitar a programação do aplicativo. A interface simples, que é mapeada diretamente para as operações básicas do banco de dados, em conjunto com a escolha de bibliotecas que possibilitam um desenvolvimento rápido, fez com que o serviço disponibilizado seja simples de usar e de manter, já que contém um código reduzido e segue padrões bem definidos.

A sincronização dos dados com o servidor, requisito fundamental para este aplicativo, foi desenvolvida com foco na eficiência na transmissão dos dados e no tratamento de erros de conexão durante a transação. Esta abordagem possibilitou que a sincronização fosse consistente mesmo em um ambiente com conectividade intermitente, conforme demonstraram os testes de aceitação. Tais testes, baseados nos requisitos iniciais, foram também importantes para a qualidade geral do sistema. Simulados metodicamente nos dispositivos dos desenvolvedores, os testes possibilitaram uma maior confiança no aplicativo resultante.

Com tudo isso, pode-se concluir que o RockDroid satisfez os objetivos deste trabalho. Todos os passos, desde a pesquisa por aplicativos semelhantes, a elicitação dos requisitos específicos dos geólogos da UnB, o projeto e o desenvolvimento voltado à interface com o usuário, resultaram em um sistema que, dadas as avaliações recebidas pelos usuários testadores, foi, em geral, bem recebido.

5.1 Trabalhos Futuros

Futuramente espera-se implementar mais funcionalidades no aplicativo de forma a torná-lo mais completo, solucionando assim o problema da necessidade de utilização de diversas ferramentas para obtenção de diferentes dados por parte do pesquisador. Dentre as ferramentas pensadas para este propósito, estão: gravação de voz e vídeos, bússola, clinômetro e criação de mapas (o usuário poderia escolher suas próprias imagens e georreferenciá-las, de forma a criar seu próprio mapa e acessá-lo mesmo sem possuir conexão com a *Internet*), seguindo as sugestões dos próprios usuários.

Além disso, seria interessante implementar funcionalidades para auxiliar no gerenciamento de projetos e equipes. Os projetos seriam administrados pelo usuário responsável por sua criação e armazenados no repositório de dados central. Outros usuários buscariam o projeto desejado nesse repositório e solicitariam participação. O administrador do projeto aprovaria ou não as solicitações recebidas e seria responsável também por gerenciar as etapas de campo. Os usuários participantes seriam agrupados em equipes e poderiam apenas enviar dados sobre afloramentos coletados, não possuindo autorização para alterar dados do projeto ou da etapa.

Nesta versão do aplicativo já foi implementada a exportação de projetos, o que é importante para que os pesquisadores possam transferir, analisar e fazer *backup* de seus próprios dados. Porém seria interessante criar uma forma de permitir aos usuários visualização e manipulação dos dados do banco de dados central da Geologia através de uma página *web*. Assim, os usuários só precisariam sincronizar os dados em seu dispositivo móvel para poder acessá-los a partir de seu computador.

Referências

[1] Smartphone OS Market Share, 2015 Q2. http://www.idc.com/prodserv/smartphone-os-market-share.jsp, 2015. Acessado em 14/12/2015. 17

[2] Rosa Alarcon, Erik Wilde, and Jesus Bellido. Hypermedia-driven RESTful service composition. *Computer Science Department of School of Information of Pontificia Universidad Catolica de Chile*, page 10, 2011. 15, 16

[3] Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform resource identifier (URI): Generic syntax. Internet RFC 3986, 2004. 16

[4] Philip Bianco, Rick Kotermanski, and Paulo Merson. Evaluating a service-oriented architecture. Relatório técnico, Carnegie Mellon University, Setembro 2007. 14

[5] Ed Burnette. *Hello, Android: introducing Google's mobile development platform.* Pragmatic Bookshelf, 2. edition, 2009. 17

[6] Dário Vieira Conceição. *Consistência de Dados em um Ambiente de Computação Móvel.* Dissertação de mestrado, Universidade Estadual de Campinas (UNICAMP). Instituto de Computação, Campinas, Agosto 1999. 5, 6, 7, 8

[7] Henrique Pereira de Freitas Filho. *Arquitetura de Coleta de Dados para Pesquisas de Campo em Ambientes Computacionais Heterogêneos.* Dissertação de mestrado, Universidade de Universidade de Brasília, 2014. vii, 3, 4, 8, 11, 12, 13, 14

[8] Ramez Elmasri and Pearson Navathe. *Sistemas de Banco de Dados.* Pearson Education do Brasil, 6. edition, 2005. 11, 12

[9] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures.* Dissertação de mestrado, University of California, Irvine, 2000. 14, 15, 16

[10] Giani Carla Ito, Maurício Ferreira, and Nilson Sant'Ana. Computação Móvel: Aspectos de Gerenciamento de Dados. *INPE - Instituto Nacional de Pesquisas Espaciais*, 10:17–18, 2003. vii, 1, 5, 6, 7, 9, 10, 11, 12, 13

[11] Frank Leymann. Web Services: Distributed Applications Without Limits. In *Proc. BTW'03*, pages 2–23. Citeseer, 2003. 14

[12] Geraldo Robson Mateus and Antonio Alfredo Ferreira Loureiro. *Introdução à computação móvel.* Universidade Federal de Minas Gerais, Universidade Federal do Rio de Janeiro, 1998. 5, 7, 8, 9, 11

[13] José Maria Monteiro. Consistência de Dados em Computação Móvel. *Departamento de Informática, PUC-Rio*, Novembro 2004. 6, 8, 10

[14] Eduardo Freire Nakamura and Carlos Maurício Seródio Figueiredo. Computação Móvel: Novas Oportunidades e Novos Desafios. *T&C Amazônia*, (2):16–28, Junho 2003. 1, 8, 9

[15] Donald Norman and Stephen Draper. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., 1986. vii, 31

[16] Mike Owens and Grant Allen. *The Definitive Guide to SQLite*. Apress, Berkely, CA, USA, 2. edition, 2010. 24

[17] Yvonne Rogers, Helen Sharp, and Jenny Preece. *Interaction Design: Beyond Human-Computer Interaction*. Wiley, 3. edition, 2011. 30, 32, 38

[18] Fred Schneider. Least Privilege and More. In *Computer Systems*, Monographs in Computer Science, pages 253–258. Springer New York, 2004. 18

[19] Abraham Silberschatz, Henry Korth, and Sudarshan. *Sistema de Banco de Dados*. Elsevier, 5 edition, 2006. 11

[20] Andrew Tanenbaum and David Wetherall. *Redes de computadores*. Campus Elsevier, 4. edition, 2003. vii, 5

[21] Kamala Thriemer, Benedikt Ley, Shaali Ame, Mahesh Puri, Ramadhan Hashim, Na Yoon Chang, Luluwa Salim, Leon Ochiai, Thomas Wierzba, John Clemens, Lorenz von Seidlein, Jaqueline Deen, Said Ali, and Mohammad Ali. Replacing paper data collection forms with electronic data entry in the field: findings from a study of community-acquired bloodstream infections in Pemba, Zanzibar. *BMC Research Notes*, 5(1), 2012. 3

[22] Graziela Simone Tonin and Alfredo Goldman. Tendências em Computação Móvel. *Departamento de Ciências da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo*, Julho 2012. 4

[23] Yi-Hua Weng, Fu-Shing Sun, and Jeffry Grigsby. GeoTools: An android phone application in geology. *Computers & Geosciences*, 44:24–30, Março 2012. 26