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

Banco de Dados Distribuído em Redes Ad Hoc no Ambiente de Coletas de Dados Geológicos

Murilo Zaffalon Marra

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientadora
Prof.a Dr.a Maristela Terto de Holanda

Brasília
2018

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

Banco de Dados Distribuído em Redes Ad Hoc no Ambiente de Coletas de Dados Geológicos

Murilo Zaffalon Marra

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia 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. Dr. Márcio de Carvalho Victorino
CIC/UnB

Prof.a Dr.a Coordenador
Coordenadora do Curso de Engenharia da Computação

Brasília, 15 de Abril de 2018

Dedicatória

Dedico aos meus pais, pela confiança demonstrada e pelo apoio durante todo esse tempo. A minha namorada, amigos e colegas que fizeram parte desta jornada. A todos os professores que ajudaram a construir o conhecimento durante esta fase.

Agradecimentos

Agradeço aos meus pais, Lisiene Nogueira Zaffalon e Sergio Marra por todo o apoio e amor que me foi dado em todos os momentos da minha vida. Agradeço, também, aos meus irmãos Isabella Zaffalon Marra, Victor Zaffalon Marra e minha namorada Martha Carolina Borges pela paciência e apoio durante a construção do presente trabalho. Agradeço também à Universidade de Brasília por me proporcionar essa experiência que é a graduação.

Resumo

O presente trabalho aborda o desenvolvimento de um aplicativo, *RockDroid*, para o sistema operacional Android, cujo objetivo é auxiliar geólogos na coleta de dados durante as pesquisas de campo. O *RockDroid* fornece ferramentas e formulários que possibilitam que os pesquisadores acompanhem o processo de coleta durante as pesquisas e uma das principais características abordadas neste trabalho é capacidade de armazenar os dados adquiridos durante a coleta e a possibilidade de sincronizar dados com outros pesquisadores presentes na coleta sem uso de Internet por meio das redes *ad hoc* e posteriormente sincronizar com o servidor central com uso de Internet.

Palavras-chave: RockDroid, Android, Coleta de dados, Banco de dados, Geologia

Abstract

This brief paper reports the development of an mobile application for Android devices that assists geological field researches. Due to the deploiment of data exchange architecture between devices, through ad hoc networks, the RockDroid provides tools that enable researchers to store and send aquired data in their equipments even in non acessible internet areas.

Keywords: RockDroid, Android, Data gathering, Database, Geology

Sumário

Lista de Figuras

Capítulo 1

Introdução

Os constantes avanços tecnológicos trazem consigo mudanças na forma de comunicação entre as pessoas. Nos dias atuais, uma boa parte da população aventura-se no Mundo dos dispositivos móveis e, por isso, existe uma grande facilidade na troca de informação entre as pessoas. Segundo [8], no final de 2015 existiam 4.7 bilhões de assinantes únicos de redes móveis no Mundo, equivalente a 63% da população mundial. Com isso, vê-se um grande investimento no ecossistema móvel, que terá uma considerável parte no crescimento tecnológico e nas inovações de serviços.

Com o aumento do uso de *smartphones* no cotidiano das pessoas, os dados presentes nos dispositivo possuem cada vez mais valor, tanto para o usuário quanto para o uso institucional das informações presentes em cada dispositivo. A adoção de aparelhos móveis permite ao usuário o acesso a uma infraestrutura de rede na maior parte dos grandes centros urbanos. Nesse contexto, surge uma necessidade de acesso aos dados, armazenados de forma distribuída, por vários aparelhos móveis [15].

O ambiente móvel permite que o usuário possa gravar informações e submetê-las a outros dispositivos ligados em uma rede de comunicação de dados. Cria-se, assim, uma metodologia de armazenamento de dados mais simples e com resultados mais rápidos, uma vez que a utilização das redes distribuídas permite a transmissão de dados a todos que necessitem do mesmo, trazendo benefícios às mais variadas áreas de pesquisa.

Durante suas pesquisas de campo, geólogos precisam armazenar informações sobre os materiais encontrados, muitas vezes fazendo anotações e observações refentes a esses dados em folhas de caderno, para só depois passarem tais informações para os computadores e compartilharem com outros pesquisadores. Desse modo, foi desenvolvido um aplicativo para auxiliar na forma como a coleta de dados ocorre, permitindo aos geólogos anotarem os dados diretamente na interface criada para cada tipo de coleta, e transmiti-los para que sejam compartilhados com outros pesquisadores. Neste trabalho dá-se foco a coleta, armazenamento e transmissão de dados no ambiente da Geologia.

O aplicativo criado é denominado RockDroid [22]. O RockDroid foi desenvolvido para um ambiente com acesso a Internet e a transferência de dados sendo realizada apenas entre dispositivos móveis e um servidor central. Grande parte das áreas utilizadas para coleta de dados pelos geólogos, porém, fica fora do alcance do sinal das operadoras de telefonia móvel, o que impossibilita o acesso à Internet na maior parte do tempo. Dessa forma, há a necessidade por parte dos pesquisadores de, durante a coleta, transmitir dados de um dispositivo móvel para outro sem o uso de Internet. O presente trabalho visa o desenvolvimento de uma arquitetura de troca de dados geológicos entre dispositivos móveis de maneira *ad hoc*, sem o auxilio da Internet. Redes *ad hoc* possibilitam que os dispositivos móveis possam se comunicar diretamente entre si sem o uso de um roteador auxiliar para encaminhamento das informações. Esse tipo de rede é conhecida como *Mobile ad-hoc network (MANET)* [9]. Nas MANETs, os dispositivos móveis são responsáveis por rotear os próprios dados, atuando como estações bases e ao mesmo tempo como dispositivos móveis [13].

1.1 Objetivo

O presente trabalho tem como objetivo desenvolver uma arquitetura de sincronização de dados geológicos entre os dispositivos móveis participantes da coletas de dados em campo, permitindo ao pesquisador trocar dados com dispositivos de outros pesquisadores, mesmo em ambientes sem acesso a Internet. Com a conclusão do trabalho, o aplicativo *Rockdroid* permitirá a sincronização de dados com outros pesquisadores presentes na coleta por meio da utilização de redes MANET.

1.1.1 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Criar uma arquitetura de sincronização de dados que permita o envio dos dados entre os dispositivos móveis sem acesso a Internet.
- Atualizar o mecanismo de sincronização para que um dispositivo móvel que recebe os dados de outro dispositivo possa sincronizar com o servidor central, sem risco de perda de dados entre o servidor e o dispositivo móvel gerador dos dados.

1.2 Estrutura do Trabalho

Este documento é composto pelos seguintes capítulos:

- Capítulo 2 - Fundamentação Teórica: apresentação dos conceitos principais para entendimento da pesquisa. Aborda-se nesse capítulo banco de dados móveis, redes móveis *ad hoc* (MANET), trabalhos relacionados a redes MANET, informações sobre o desenvolvimento do RockDroid;
- Capítulo 3 - Banco de Dados Distribuído em Redes *Ad hoc* no Ambiente de Coletas de Dados Geológicos: apresentação das fases de desenvolvimento do projeto, incluindo elicitação de requisitos, requisitos, *design*, protótipo e o próprio processo de desenvolvimento;
- Capítulo 4 - Teste e Validação: descrição dos testes realizados para garantir a validade das funcionalidades desenvolvidas, apontando possíveis problemas;
- Capítulo 5 - Conclusões: apresentação das conclusões obtidas e de propostas de futuros projetos a serem desenvolvidos.

Capítulo 2

Fundamentação Teórica

Este capítulo visa apresentar os conceitos necessários para o entendimento deste trabalho, incluindo uma breve discussão sobre banco de dados móveis, redes móveis *ad hoc* (MANET), trabalhos relacionados às redes MANET, *RockDroid* e coleta de dados geológicos.

A arquitetura dos bancos de dados também é influenciada pelos usuários de redes móveis. Devido a movimentação constante desses usuários nas redes, possíveis quedas de serviço e larguras de banda limitadas fazem com que as soluções precisem ser capazes de lidar com essas situações [17]. Nesse contexto, o acesso ao banco de dados por meio de dispositivos móveis precisa ser pensado de forma a reduzir o número de requisições aos servidores, diminuindo assim o tráfego na rede.

Com essa visão, novos paradigmas de transação devem ser desenvolvidos para tratar os problemas recorrentes de uma conexão móvel. Devem ser desenvolvidos mecanismos para garantir a consistência dos dados, e para tratar consultas ao banco de dados necessárias em situações em que o dispositivo fica sem conexão com a Internet [25].

2.0.1 Banco de Dados

Nesse contexto um banco de dados é uma coleção de dados inter-relacionados, que representam informações sobre um conteúdo específico. O dado tem seu significado implícito e pode ser registrado por um fato gerador do dado. Em ambientes computacionais, o dado é o que realmente é armazenado no banco, ficando a geração do significado desses dados por parte do usuário ou de softwares de visualização de dados [18].

Entre o banco de dados e a aplicação de visualização e inserção de dados, tem-se uma camada de software, chamada de Sistema de Gerenciamento de Dados (SGBD). O SGBD é responsável por gerenciar o acesso, manipular e organizar os dados, isolando assim o usuário de ver detalhes de hardwares desnecessários. Resumidamente, o SGBD é uma

coleção de programas que habilita o usuário a criar e manter um banco de dados de forma fácil, sem necessitar de conhecimento específicos sobre seu funcionamento [10].

Fazendo operações de acesso a banco de dados (inserção, remoção, edição e consultas) precisa-se de um meio para garantir a consistência dos dados. Com esse propósito em vista, transações são unidades lógicas de processamento de dados que tem como função garantir que os dados sejam escritos corretamente, mesmo que sejam executadas várias transações concorrentes no mesmo banco. Com essa concorrência, o banco de dados fica sujeito a ocorrência de possíveis falhas [13].

Em bancos de dados tradicionais, conhecidos como relacionais, as transações devem então ser atômicas, consistentes, isoladas e duráveis. Essas propriedades são conhecidas como as propriedades ACID [31]:

- Atomicidade: uma transação deve ser executada totalmente ou não terá efeito;
- Consistência: uma transação isolada leva o banco de dados de um estado consistente a um novo estado também consistente;
- Isolamento: o sistema gerenciador deve garantir que não há interferência entre as transações. Diversas transações podem ocorrer de forma concorrente e cada uma deve ocorrer de forma isolada;
- Durabilidade: quando uma transação se completa, seus efeitos devem persistir, mesmo que ocorram falhas no sistema.

2.0.2 Arquitetura de Banco de Dados Móveis

Vários tipos de arquiteturas são possíveis para bancos de dados móveis. A arquitetura básica consiste em um servidor central de banco de dados e aplicações clientes móveis [17].

Os dispositivos móveis se comunicam com o servidor central de forma a obter informações do banco de dados. As informações no servidor central são dinâmicas e podem ser usadas pelos dispositivos móveis [17]. Por vezes esses dados são salvos no dispositivo móvel apenas em memória volátil, portanto, podem ser perdidos durante o processamento e o escalonamento dos processos no sistema operacional. Uma solução para esse problema é o uso de um banco de dados que armazena os dados recebidos por esse servidor central [35].

As aplicações dos dispositivos móveis tem pouco recurso de hardware e banda de conexão disponíveis, além de lidar com a constante movimentação entre redes, causando desconexões e outros problemas que necessitam tratamento para não impactar na performance das aplicações. Por outro lado, o servidor central normalmente é um computador

com alto poder de processamento, especializado no gerenciamento das requisições feitas pelos dispositivos móveis de forma a reduzir a carga de processamento sobre os dispositivos [17].

2.0.3 Características dos Banco de Dados Móveis

Equipamentos móveis frequentemente sofrem com desconexões causadas pela movimentação entre várias antenas de telefonia e, por causa da bateria, podem subitamente perder a conexão por um grande período de tempo [17]. Isso torna a comunicação com um servidor central bem complexa, considerando que os enlaces disponibilizados por grande parte das telefonias têm um custo elevado e muitas vezes uma baixa velocidade de banda [38]. Assim, há uma grande necessidade de prover um suporte a desconexões e a utilização da aplicação móvel mesmo com as adversidades do meio.

A maioria das aplicações móveis possuem um banco de dados local. No sistema operacional Android, têm-se o banco de dados SQLite como ferramenta nativa. Da mesma forma, a maior parte das aplicações móveis utilizam o SQLite, principalmente por este ser um banco de dados que utiliza poucos recursos de memória [33]. Para aplicações que transferem e recebem dados por meio da Internet, o banco de dados móvel é utilizado como forma de realizar o *caching* dos dados e a replicação deles [17].

O objetivo do *caching* é o acesso eficiente aos dados. No *caching*, os dados frequentemente requeridos pela aplicação tendem a ficar armazenados localmente na unidade móvel. Quando uma requisição ao banco de dados é realizada, primeiro busca-se o dado no cache e, caso o dado não seja encontrado, uma requisição solicita ao banco de dados central a informação para que ela seja atualizada localmente [14].

A replicação de dados é o processo aonde as transações, feitas de forma serializadas e assíncronas, são replicadas para um ou vários bancos de dados além do banco local [24]. Na replicação, deve ser mantida a consistência dos dados entre as unidades móveis e o servidor central. Para as aplicações móveis isso significa gravar os dados no banco de dados local até que exista uma conexão com o banco de dados central, para que esse seja sincronizado. Da mesma forma, atualizações no banco de dados central devem ser replicadas ao banco local quando existir uma conexão.

2.0.4 Banco de Dados SQLite

A conectividade limitada e intermitente e as limitações de memória e de bateria fazem necessário o emprego de um banco de dados que possa ser utilizado em dispositivos com poucos recursos. Com esse objetivo em vista, foi desenvolvido o SGBD *SQLite* em ambientes móveis. O *SQLite* consegue gerenciar o banco de dados usando apenas 250 kilobytes

da memória de um dispositivo. Os tipos de dados implementados são NULL, TEXT, INTEGER ou REAL. Para outros tipos de dados, deve ser implementada uma conversão na aplicação [36].

Além do baixo consumo de recursos, o *SQLite* é muito utilizado por possuir propriedades que o diferencia dos tradicionais gerenciadores de bancos de dados. Não existe a necessidade de instalação para ser utilizado e nem de uma configuração obrigatória. O administrador do banco, por sua vez, não precisa conceder permissões nem acesso aos usuários. Aplicações que querem acessar o banco de dados apenas realizam operações de escrita ou leitura no próprio arquivo em disco. Um banco *SQLite* é armazenado por inteiro em um único arquivo, possibilitando a transferência total dos dados para outra localização apenas pela cópia do arquivo do respectivo banco.

Existem algumas situações onde o *SQLite* não é recomendado: quando a aplicação necessita que mais de uma operação de escrita seja realizada concorrentemente, o *SQLite* não é a solução recomendada. O *SQLite* permite apenas um escritor por vez, logo todas as ações de inserção ou atualização de dados devem ser realizadas sequencialmente [16].

O *SQLite* possui uma equipe internacional de desenvolvedores que trabalha para expandir a capacidade e melhorar o desempenho e a confiabilidade do banco [5]. O SQLite é um projeto de código aberto, o que permite que qualquer membro da comunidade possa contribuir com o projeto.

2.1 Redes Móveis

Grande parte dos dispositivos móveis atuais permitem a conexão com a Internet. Para que isso ocorra, muitos deles vêm com a possibilidade de conectar a uma rede por meios não físicos de transmissão, conhecidas como redes Wi-Fi.

Redes Wi-Fi ligam dois ou mais dispositivos a um ambiente em comum, com o intuito de permitir a conexão destes dispositivos a Internet. Para essa conexão, o Wi-Fi utiliza do protocolo IEEE 802.11 que prevê dois modos de operação [12]. O primeiro e mais comum usa de pequenos roteadores, conhecidos como pontos de acesso, que fazem uma ponte entre os vários aparelhos conectados. O segundo modo de operação, conhecido como ponto-a-ponto (P2P), usa os próprios usuários da rede para distribuição de informações, removendo a necessidade de um ponto de acesso central para coordenar a passagem e distribuição das informações [12].

Em comparação com as redes com ponto de acesso, as redes P2P fornecem um custo reduzido de infraestrutura com a criação de uma rede auto organizada e descentralizada, com maior tolerância a falhas, e um suporte a redes *ad hoc* [11]. Em organizações *ad hoc*, vários dispositivos comunicam-se entre si sem que haja a comunicação com redes externas.

2.1.1 Redes Móveis *ad hoc*

As redes *ad hoc* são redes sem fio que dispensam o uso de um ponto de acesso em comum com cada dispositivo móvel conectado a ela, de modo que todos os dispositivos da rede funcionam como se fossem um roteador. Essa é a rede conhecida como *Wireless ad-hoc network* ou também *Mobile ad-hoc network* (MANET). A MANET é um tipo de rede sem fio descentralizada [12].

Uma rede MANET é uma das possíveis arquiteturas de rede sem fio. Nessa rede, um dispositivo pode se comunicar sem fio através das tecnologias implementadas no próprio aparelho. Alguns anos atrás usava-se apenas redes *Bluetooth* para este tipo de comunicação, hoje em dia já há a possibilidade de criar redes conhecidas como *Wi-Fi Direct* que usam do protocolo IEEE 802.11 e suas vantagens para manipulação da rede [6]. Nas redes MANET cada dispositivo móvel é responsável por rotear seus próprios dados, atuando assim como ponto de acesso e cliente simultâneamente. Devido a essa estrutura, os protocolos devem ser robustos o suficiente para aceitarem mudanças constantes na topologia da rede, como a entrada de novos clientes ou a saída deles.

Cada dispositivo móvel em uma MANET é um nó autônomo, o que significa que cada nó tem as habilidades básicas de enviar, receber e rotear os dados. O algoritmo de roteamento em um MANET pode ser de conexão direta com o nó receptor ou usar outros nós da rede para encontrar o nó receptor. A conexão direta é simples em termo de estrutura e implementação, mas tem menos funções se comparada com redes com vários nós. Em redes com vários nós, o receptor pode estar longe da área de transmissão coberta pelo dispositivo emissor e, dessa forma, os pacotes devem ser transmitidos por meio de outros nós na rede que consigam alcançar o nó receptor, passando de nó em nó ate o encontrar. A Figura 2.1 mostra uma MANET com vários dispositivos e sua área de alcance de transmissão. O *nó 1* e o *nó 2* são vizinhos e os *nós 1, 4 e 5* não tem uma conexão direta. Por isso, dados enviados entre o *nó 3* e o *nó 5* devem ser retransmitidos pelos *nós 1, 2, 3, 4* e chegar ao *nó 5* [19].

2.1.2 Redes Móveis *Wi-Fi Direct*

A tecnologia *Wi-Fi Direct* foi desenvolvida pela *Wi-Fi Alliance* com o objetivo de conectar dispositivos entre si, removendo a necessidade de um roteador, e acelerando o processo de conexão. Essa conexão é essencial entre dispositivos como impressoras, televisões e outros dispositivos nos quais a sincronização e compartilhamento do conteúdo é um requisito necessário para o funcionamento do dispositivo. As especificações desenvolvidas pela *Wi-Fi Alliance* operam em dispositivos que utilizam o protocolo 802.11, com conexões na faixa

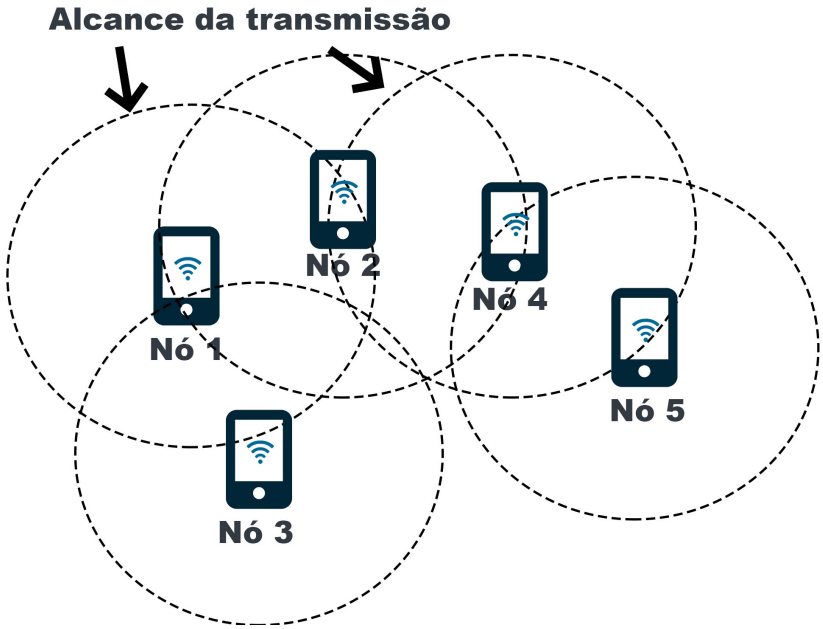

Figura 2.1: Nós e alcance de transmissão em uma MANET [19].

de valores entre 2.4 GHz a 5 GHz. A comunicação possui um alcance de transmissão de aproximadamente 200 metros, tornando seu uso bem eficiente em pequenos ambientes [6].

Wi-Fi Direct foi construído com base nos mesmos atributos de qualidade do Wi-Fi como segurança, desempenho, facilidade de uso e ubiquidade, removendo a necessidade de uso de um ponto de infraestrutura fixa. Um cliente em vez de se ligar diretamente a uma infraestrutura, pode se ligar diretamente ao dispositivo que deseja, usando apenas os serviços que necessitar. Essa funcionalidade permite que, em ambientes sem infraestrutura física, seja possível que um cliente conecte-se a uma televisão e mostre fotos do seu celular diretamente na tela, ou até mesmo consiga imprimir documentos conectando-se diretamente a sua impressora.

Os dispositivos que utilizam dessa tecnologia suportam os mesmos padrões de desempenho de dispositivos Wi-Fi regulares. Assim, as taxas de transferência de dados acompanham os padrões 802.11 b/g/n, que vão ate 450 Mbps, sendo seu alcance estendido até 200 metros em condições ótimas. Essas especificações trazem benefícios ao *Wi-Fi Direct* que são definidos por [6]:

- **Mobilidade e Portabilidade:** dispositivos com *Wi-Fi Direct* podem se conectar de qualquer lugar a qualquer momento, se suas redes estiverem ao alcance;
- **Facilidade de Uso:** clientes tem a possibilidade de buscar as redes *Wi-Fi Direct* antes de se conectarem de forma a reconhecer o serviço e o dispositivos antes de estabelecerem uma conexão;

- **Conexões Seguras:** dispositivos *Wi-Fi Direct* usam da tecnologia *Wi-Fi Protected Setup* para simplificar a criação e a conexão entre os dispositivos. Essa tecnologia cria um Número de Identificação Pessoal (PIN) que deve ser digitado no dispositivo que pede a conexão para verificar a integridade dessa conexão.

Para melhorar o funcionamento das redes *Wi-Fi Direct*, dispositivos ligados nessa rede podem criar grupos conhecidos como Wi-Fi P2P, podendo atuar tanto como ponto de acesso como cliente. Quando o dispositivo atua como ponto de acesso, ele é conhecido como proprietário do grupo (*group owner*) [6]. Existem duas formas de definir o proprietário do grupo, a primeira vem de uma escolha manual pelo usuário. Normalmente, é definido como proprietário o cliente que iniciou o grupo. A segunda forma, mais eficiente, cria uma negociação entre os membros do grupo, essa negociação é tratada por um valor numérico conhecido como valor de intenção (*intent*) [6]. O valor de intenção é calculado baseado em várias métricas como porcentagem de bateria, força do sinal e se ele já foi ou é proprietário do grupo. Desse modo, o dispositivo que tiver o maior valor de intenção é aceito como o novo proprietário do grupo [6].

Os dispositivos *Wi-Fi Direct* podem, então, se conectarem formando grupos com uma topologia um-para-um ou um-para-muitos. O protocolo também permite que um cliente se conecte a uma rede Wi-Fi comum além da rede *Wi-Fi Direct*. A tecnologia fornece ao dispositivo a capacidade de descobrir outros dispositivos e informações limitadas sobre eles antes da conexão. Nessa conexão existem alguns mecanismos chaves [6]:

- **Descoberta do dispositivo (*discovery*):** mecanismo usado para descobrir outros dispositivos Wi-Fi Direct no alcance do dispositivo;
- **Descoberta do serviço (*service discovery*):** recurso opcional que permite a publicação de serviços suportados a outros dispositivos *Wi-Fi Direct*. Essa descoberta de serviço pode ser realizada a qualquer momento.;
- **Formação do grupo:** um grupo pode ser criado por um único dispositivo. Quando existe uma conexão entre dois dispositivos, um grupo pode ser formado automaticamente. Os dispositivos vão negociar para determinar quem vai gerenciar o grupo. Depois do grupo ser formado, outros dispositivos podem se juntar ao grupo, como visto na Figura 2.2;
- **Gerenciamento de energia:** o uso de energia para aparelhos móveis é essencial para o tempo de vida que essa conexão se mantêm aberta. Por isso, o *Wi-Fi Direct* inclui mecanismos de gerenciamento de energia, que podem reduzir o consumo de energia.

Figura 2.2: Adicionando um dispositivo a rede Wi-Fi Direct por procedimento de convite [37].

2.2 Trabalhos Relacionados

Esta seção busca identificar outros softwares que estão inseridos no mesmo contexto de transmissão de dados por meio das redes *ad hoc*, buscando encontrar funcionalidades semelhantes as desejadas para o aplicativo *RockDroid*.

2.2.1 Aplicativos Relecionados com *ad hoc*

Alguns dos aplicativos encontrados na literatura serviram de base para identificar o funcionamento das redes MANETs em aplicativos móveis. Os aplicativos escolhidos foram *EIKO: A Social Mobile Network for MANET*, que tem como foco o emprego de uma MANET para criação de uma rede social [23]; *WiFiDirect* feito pela *SparkPlug*, que encontra-se disponpível na *Google Play Store* e tem a função de criar uma rede e compartilhar arquivos; *EduConnect*, que busca facilitar a transferência de arquivos entre alunos e professores em ambientes sem Internet[21].

O *EIKO* [23] tem como base a criação de redes MANET por meio da conexão *Bluetooth* IEEE 802.15.1, ele foi escritos em *Java Micro Edition (JME)* e funciona em aparelhos que aceitam aplicações JME como antigos aparelhos da Nokia e Sony Ericson. A Figura 2.3 mostra o ambiente de comunicação da aplicação *EIKO*.

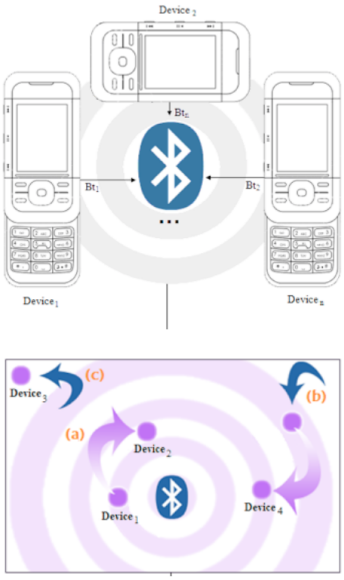

Figura 2.3: Ambiente de comunicação criado pela aplicação *EIKO* [23].

O caso idealizado no artigo para o aplicativo *EIKO* é que membros em um ambiente de conferência que possuem interesses em comum podem compartilhar seus perfis de interesse, criando um mini currículo de cada integrante da conferência. Como em grandes conferências é quase impossível a comunicação entre todos os membros, o EIKO aproveita das redes *Bluetooth* para enviar os currículos para visualização ou até iniciar um chat entre os integrantes da rede [23].

O aplicativo *WiFiDirect* tem uma proposta mais educativa e foi criado para ensinar sobre o funcionamento do *Wifi Direct* nos aparelhos *Android*. O *WiFiDirect* compartilha arquivos de um dispositivo para outro usando a tecnologia do *Wifi Direct* que é a mesma tecnologia utilizada no presente trabalho para a transmissão de dados e criação de redes MANET [34]. A Figura 2.4 mostra a tela de apresentação do aplicativo *WiFiDirect*.

Com uma visão mais prática e focada em auxiliar desenvolvedores, os responsáveis pelo aplicativo *WiFiDirect* liberaram seu código publicamente em um repositório *git* hospedado no *GitHub* de forma a ajudar outras pessoas que pretendem adicionar a tecnologia do *Wifi Direct* em suas aplicações *Android* [34]. Por esse motivo, o *WiFiDirect* é um dos aplicativos usados como base de auxílio ao desenvolvimento do projeto.

O aplicativo *EduConnect* [21] é uma proposta de apoio à aprendizagem colaborativa, que visa facilitar a transferência de arquivos entre professores e alunos, onde o uso da Internet não é possível. O *EduConnect* também utiliza-se da tecnologia *Wifi Direct* [21]. Como visto na Figura 2.5, o aplicativo mostra uma lista de pessoas conectadas a rede e uma lista de pessoas que estão próximas e podem se conectar a rede para iniciar uma

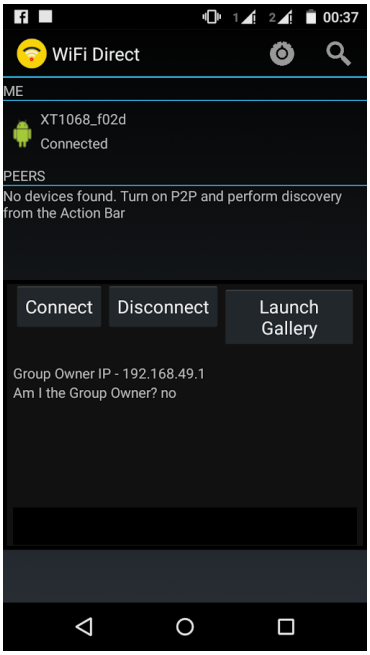

Figura 2.4: Tela principal do aplicativo *WiFiDirect*.

transferência de arquivos.

Figura 2.5: Interface do aplicativo *EduConnect* em tablets e smartphones [21].

Os aplicativos citados anteriormente, apesar de não estarem incluídos no mesmo contexto da Geociência que o *RockDroid*, se aproveitam das redes MANET e serviram de base para o desenvolvimento das funcionalidades adicionadas ao *RockDroid*. Com essas aplicações, foi possível entender na prática como são implementadas as redes MANET no sistema operacional *Android* e como são enviados dados entre dois dispositivos nessa rede.

2.3 RockDroid - Uma Arquitetura para Coleta de Dados Geológicos

O *RockDroid* é um sistema proposto para auxiliar os pesquisadores do Instituto de Geociências da Universidade de Brasília (IG/UnB) quando estes vão fazer pesquisas de campos, local onde é necessária a coleta de dados geológicos em um ambiente com conectividade limitada e no qual o pesquisador está em constante movimentação pelo terreno.

Um grande grupo de pesquisadores saem para fazer a coleta de dados usando o aplicativo *RockDroid*. Eles são divididos em pequenos grupos e separados por áreas de coleta cada membro do grupo então registra em seu próprio aplicativo todos os dados encontrados no local, e no fim da coleta sincroniza esses dados com o servidor central. Com os dados sincronizados os pesquisadores podem fazer análises para uso em suas pesquisas. Como visto na Figura 2.6 tem-se a arquitetura desta coleta.

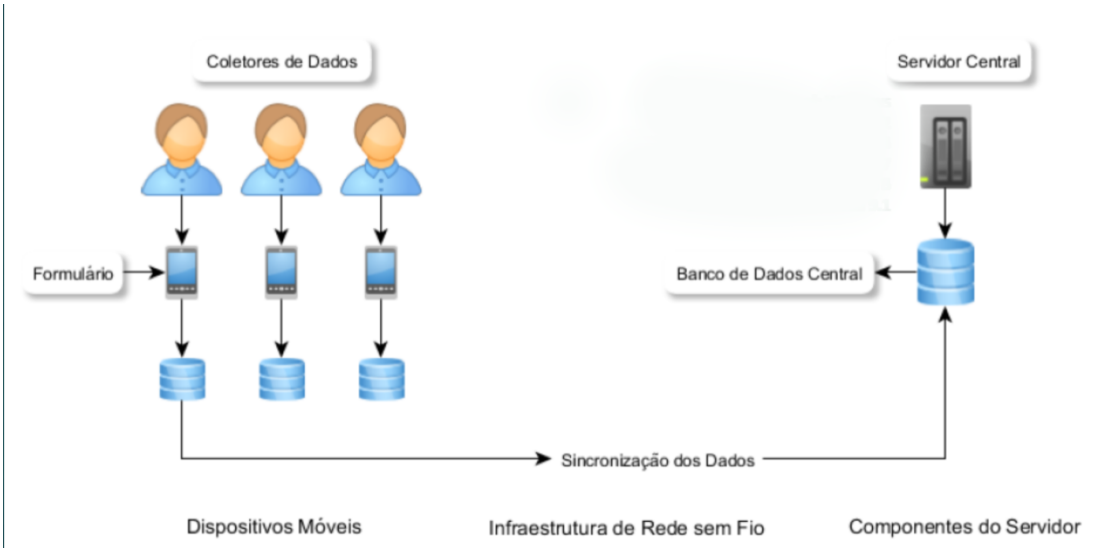

Figura 2.6: Arquitetura da coleta de dados.

O *RockDroid* implementa as funcionalidades citadas a seguir:

- **Cadastro de usuários:** o usuário após o download do aplicativo pode se cadastrar por um formulário que é enviado ao o servidor de suporte da aplicação.
- **Autenticação:** usuários devem se autenticar para usar o sistema e sincronizar dados com o servidor, porém uma conta de visitante permite com algumas limitações o uso do sistema sem cadastro.
- **Cadastro de projetos:** o usuário pode cadastrar, modificar e excluir projetos na sua conta. Depois de autenticado, esses projetos só podem ser vistos por ele.

- **Cadastro de etapas de campo:** o usuário pode cadastrar, modificar e excluir etapas. Dados de entrada: nome, data de início, município e UF;
- **Cadastro de afloramentos:** o usuário pode cadastrar, modificar e excluir afloramentos. Dados de entrada: *nome, latitude, longitude, altitude, toponímia, descrição e fotos*;
- **Cadastro de rochas:** o usuário pode cadastrar, modificar e excluir rochas. Dados de entrada: *nome, tipo de rocha, textura, mineralogia, cor e composição, tamanho, trama, nomenclatura, grau metamórfico, composição e fotos*;
- **Cadastro de estruturas:** o usuário pode cadastrar, modificar e excluir estruturas. Dados de entrada: *tipo de estrutura, descrição, tipo de plano, fase, mergulho, direção de mergulho e fotos*;
- **Cadastro de amostras:** o usuário pode cadastrar, modificar e excluir amostras. Dados de entrada: nome e fotos;
- **Obtenção de coordenadas via GPS:** utilização do sistema de geolocalização do dispositivo para obtenção das coordenadas do usuário;
- **Registro fotográfico dos recursos cadastrados:** o usuário pode utilizar a câmera presente no dispositivo para obter fotos e associar aos dados cadastrados;
- **Exportação de dados:** o usuário pode exportar de dados em formato de planilhas do Excel (XLS);
- **Sincronização:** o usuário pode sincronizar os dados entre o banco de dados local do dispositivo e o banco de dados do servidor central;
- **Sincronização com outro dispositivo:** o usuário pode sincronizar um projeto com outro dispositivo ao alcance da rede *Wifi Direct*, replicando os dados de um dispositivo para outro;
- **Visualização de afloramentos no mapa:** o usuário pode visualizar os dados de afloramentos coletados no mapa;
- **Download de mapa:** o usuário pode previamente fazer o download dos mapas para visualização sem necessidade de conexão com a Internet.

O trabalho proposto neste documento é a continuação do desenvolvimento do software *RockDroid*, voltado para o uso em coletas de dados por pesquisadores em ambientes da Geociência. Os usuários desse segmento estão em constante movimento, sem conexão com a Internet, na maior parte do tempo.

Capítulo 3

Banco de Dados Distribuído em Redes Ad Hoc no Ambiente de Coletas de Dados Geológicos

Melhorando as funcionalidades do *RockDroid*, este documento propõe uma arquitetura que possibilite a transmissão de dados entre pesquisadores durante o momento da coleta sem o uso de Internet. Para tornar isso possível, foi proposto a utilização de redes MANET em conjunto com o emprego do protocolo *Wifi Direct* para criação de redes e transmissão de dados.

3.1 Elicitação de Requisitos

Para iniciar este projeto, foi necessário entender quais eram as dificuldades encontradas pelos usuários ao utilizar o aplicativo *RockDroid*. Percebeu-se, então, que o *design* proposto pelo aplicativo foi bem aceito pelos usuários, porém durante os testes de uso, os usuários do aplicativo encontraram demandas de funcionalidades que ainda não existiam na versão inicial do aplicativo. Algumas dessas funcionalidades podem ser descritas como:

- A criação de uma aplicação Web que englobasse com facilidade todos os dados sincronizados entre o aplicativo e o servidor. No modelo do aplicativo antes da elaboração deste projeto, os dados podiam ser exportados para planilhas Excel pelo próprio aplicativo, mas não existia a facilidade de visualizar todos os dados que foram sincronizados com o servidor em uma página Web para elaboração de uma análise aprofundada dos dados;

- A constante determinação da posição do usuário e a atualização em um servidor central, para informar a segurança dos usuários em campo e tomar as medidas cabíveis em caso de algum acidente;
- Um banco de dados que permitisse receber os dados de diversas fontes e tratar os mesmos para um formato aceito como padrão pela comunidade de pesquisadores em Geologia. Dessa forma, o *RockDroid* entraria apenas como mais uma fonte de dados;
- A falta de Internet durante as pesquisas de campo torna a sincronização de dados com o servidor uma tarefa complexa, gerando a necessidade de transferência de dados entre os diferentes usuários tanto para análise individual dos dados por cada usuário, quanto para sincronizar os dados com o servidor pelo usuário que possuir acesso a Internet.

O objetivo deste projeto, então, é melhorar o software *RockDroid*, solucionando o último requisito descrito anteriormente. Para isso, foi seguido o modelo de desenvolvimento proposto pelo projeto inicial do *RockDroid* que foi centrado no usuário, buscando seus objetivos, tarefas e habilidades, como visto na Figura 3.1.

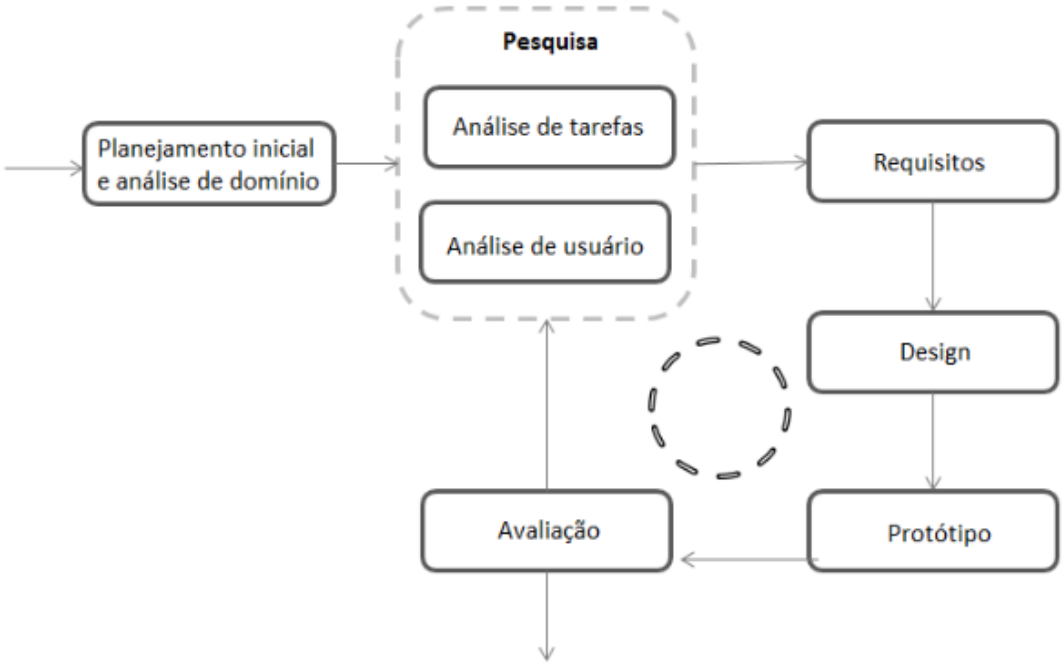

Figura 3.1: Processo de desenvolvimento centrado no usuário, adaptada de [27].

Por esse modelo, o primeiro passo que se segue é a análise de domínio. Com essa parte completa, pode-se passar para parte de pesquisa que inclui a análise de tarefas e a análise

de usuário. Por fim, espera-se ter definido os requisitos para a criação do *design* da nova funcionalidade e a implementação de um protótipo funcional para avaliação do usuário. O usuário é essencial nessa avaliação para dar novas informações sobre sua experiência. Caso não seja favorável, o processo de pesquisa pode ser reiniciado, atentando a novas informações adicionados pelo usuário.

3.2 Análise de Domínio

A análise de domínio é uma parte essencial do projeto aonde os desenvolvedores podem ser incluídos no contexto em que o projeto está inserido, uma vez que esses podem não ter entendimento sobre a área que o projeto engloba. Grande parte da análise de domínio deste projeto foi feita previamente pela equipe inicial do *RockDroid* e a professora orientadora deste projeto. Em um nova reunião com os professores do IG/UnB e a professora orientadora, uma segunda análise foi feita focada apenas nos novos problemas apresentados pelos professores e atuais usuários do aplicativo.

3.3 Análise de Tarefas e Análise de Usuário

A análise de tarefas é o processo de coleta de informações sobre as tarefas que o usuário executa e os procedimentos que podem ser utilizados por aquele usuário. Como tarefa, pode-se considerar qualquer processo que o usuário executa para concluir um objetivo definido [32].

A análise de usuário engloba a observação do usuário no ambiente onde ele deve concluir suas tarefas. Pode-se analisar, desse modo, quais as dificuldades encontradas pelo usuário para se identificar a melhor forma de auxiliar na conclusão das mesmas [32].

Para a nova funcionalidade que deseja-se implementar, a análise de usuário foi de grande importância, pois por meio dessa análise sobre o uso do *RockDroid* percebeu-se quais dos problemas foram devidamente solucionados e se tornou evidente quais dos problemas iniciais não foram resolvidos. Como resultado, é possível elicitar novos requisitos para dar continuidade ao projeto.

3.4 Requisitos

Iniciallmente, o fluxo principal de uso do aplicativo não é alterado. O usuário abre o aplicativo, faz o *login* ou cadastra um novo usuário caso tenha Internet. O usuário também pode entrar como Visitante se houver necessidade. Dentro do aplicativo o usuário pode criar um novo projeto ou alterar projetos já criados, o que possibilita a adição de novas

etapas ao projeto selecionado. Assim, inicia-se a coleta de dados. Quando encerrada a coleta, o usuário pode sincronizar os dados com o servidor ou exportá-los para uma tabela Excel.

Com a introdução do novo requisito de transferência de dados pela rede *ad hoc*, a forma como utiliza-se o aplicativo pode não variar caso o usuário assim deseje, porém pelo menos dois novos casos de uso do aplicativo foram adicionados no processo de utilização da aplicação. O compartilhamento de projetos entre os pesquisadores e o envio de projetos para outros pesquisadores pela rede, antes da sincronização com o servidor central, constituem dois novos fluxos de execução que devem ser desenvolvidos.

Durante a análise de usuário, os pesquisadores informaram que, em campo, é difícil a obtenção do sinal de Internet por todos os membros da expedição, e que já havia ocorrido vários casos em que apenas uma determinada operadora de telefonia móvel possuía antenas na região. Desse modo, a ideia é criar um meio de enviar projetos para outros usuários sem a utilização de Internet, de forma que o usuário com a operadora móvel funcionando pudesse sincronizar os dados de todos os pesquisadores gerados naquele dia.

A segunda forma possível de uso é o compartilhamento do projeto entre vários usuários. O usuário principal e dono do projeto possui a habilidade de compartilhar seu projeto com outros usuários. Os usuários podem, então, visualizar, atualizar e editar as informações antes de realizar a sincronização com o servidor. O novo modelo de processo encontra-se na Figura 3.2.

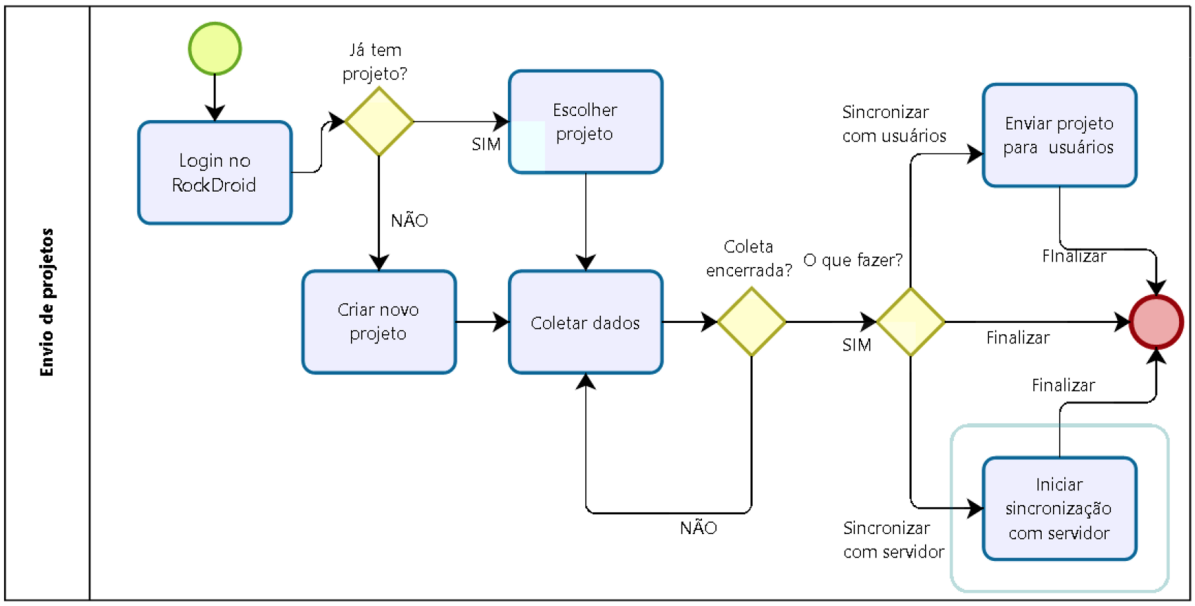

Figura 3.2: Fluxograma envio de projetos.

Com a criação do projeto por um usuário, tem-se uma segunda forma de interagir com

o processo. Como visto na Figura 3.3 pode-se receber o projeto iniciado de um usuário e continuar a coleta do ponto que o usuário parou.

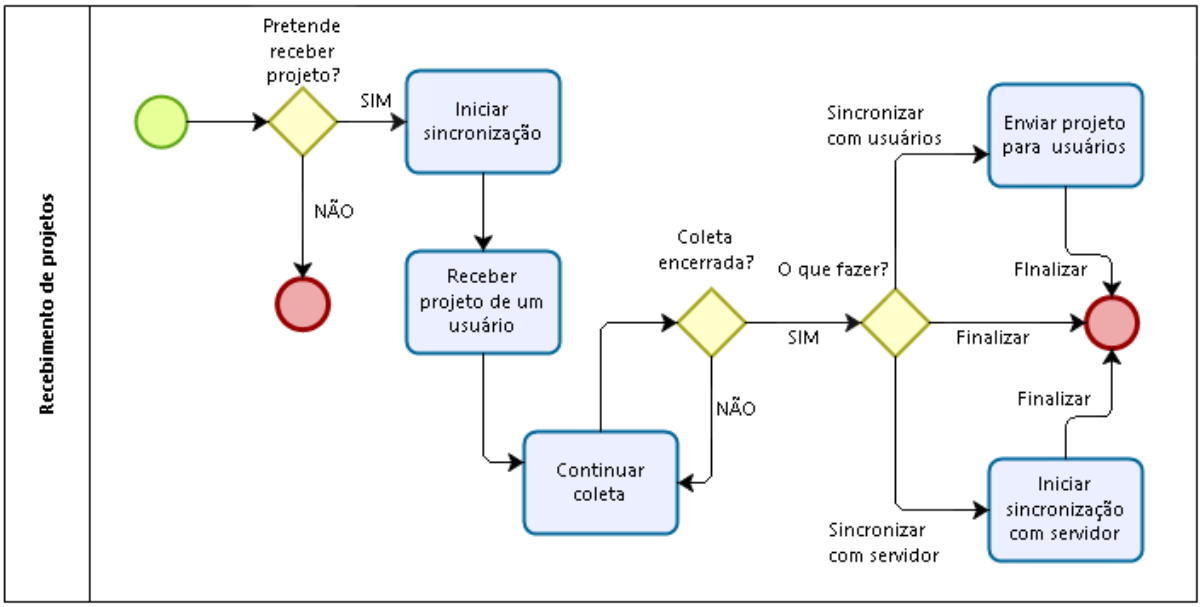

Figura 3.3: Fluxograma do recebimento de projeto.

O processo inicia-se com um usuário **A** escolhendo qual projeto deseja enviar, veja a Figura 3.2. Com isso, uma nova tela é aberta no qual deve ser escolhido o outro aparelho que deseja-se conectar. Ao mesmo tempo, um outro usuário **B** deve abrir no menu a tela '*Receber de outro celular*' e escolher com que aparelho deseja conectar para a troca de informações veja a Figura 3.3.

Com os requisitos definidos, foi elaborado um *sketch* baseado no *design* atual do *Rock-Droid*, de modo que o *design* original não tivesse grandes alterações, uma vez que ele já foi validado pelos usuários. A Figura 3.4 apresenta os *sketches*.

Uma vez criado os *sketches*, foi decidido não criar protótipos dessa funcionalidade por ser apenas uma atualização do aplicativo e não uma reformulação completa.

3.5 O Processo de Desenvolvimento

O *Rockdroid* foi projetado para ser modular, fazendo com que os módulos pudessem ser reutilizados e que grandes modificações como troca de banco de dados não afetassem o desenvolvimento do aplicativo. Para concluir essa tarefa, o aplicativo foi construído seguindo o paradigma de *Model-View-Presenter (MVP)* [30], com injeção e inversão de dependências, seguindo uma divisão de camadas bem definidas.

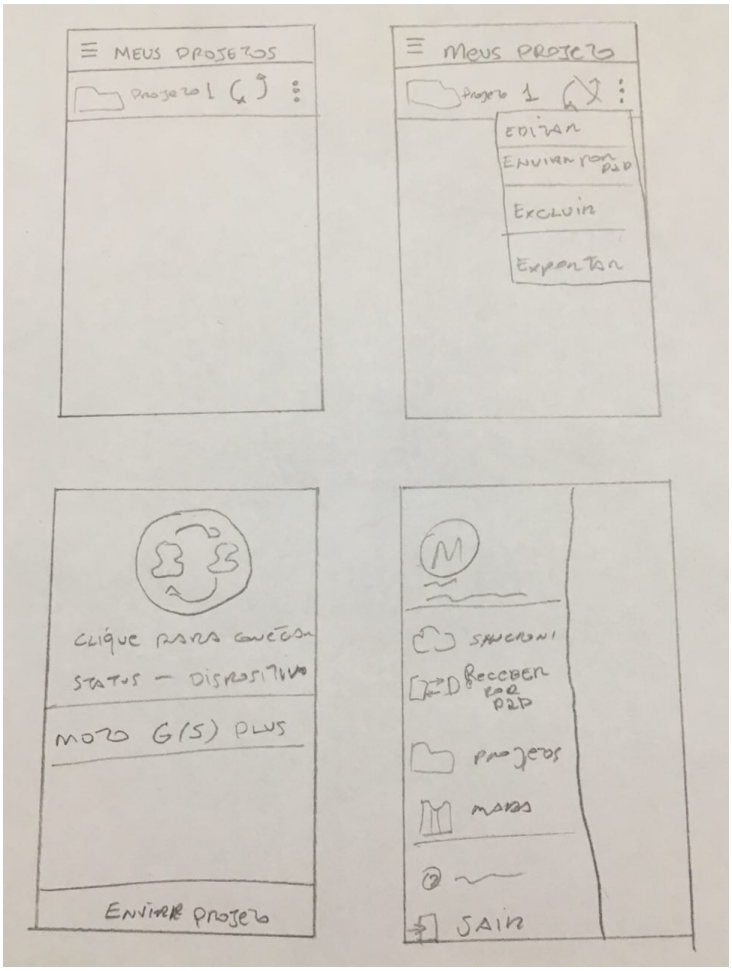

Figura 3.4: Telas do *sketch* criado.

3.5.1 Camadas

A maioria das arquiteturas de projeto atuais são baseadas em camadas. Cada camada tem funções especificas e deve conseguir completar suas tarefas sem necessidade de acesso a camadas de mais alto nível, ao mesmo tempo em que funcionam como uma interface que esconde detalhes específicos de camadas de mais baixo nível do sistema [20]. Assim, com o código das camadas consideradas superiores ficando separado do código das camadas inferiores, o entendimento da funcionalidade de um componente fica facilitado. A troca de um componente tem menor impacto no código como um todo.

No *Rockdroid* a arquitetura foi planejada com quatro camadas distintas que criam uma ordem de dependência entre si. As camadas mais externas tem conhecimento das camadas internas, mas as camadas internas não conhecem as mais externas. A primeira camada e mais interna é a camada de entidades; a segunda é a camada de casos de usos; a terceira constitui a camada de acesso e, por fim, a camada de apresentação, sendo esta a mais

externa e com conhecimento de todas as outras. Cada camada é detalhada e mostrada na Figura 3.5:

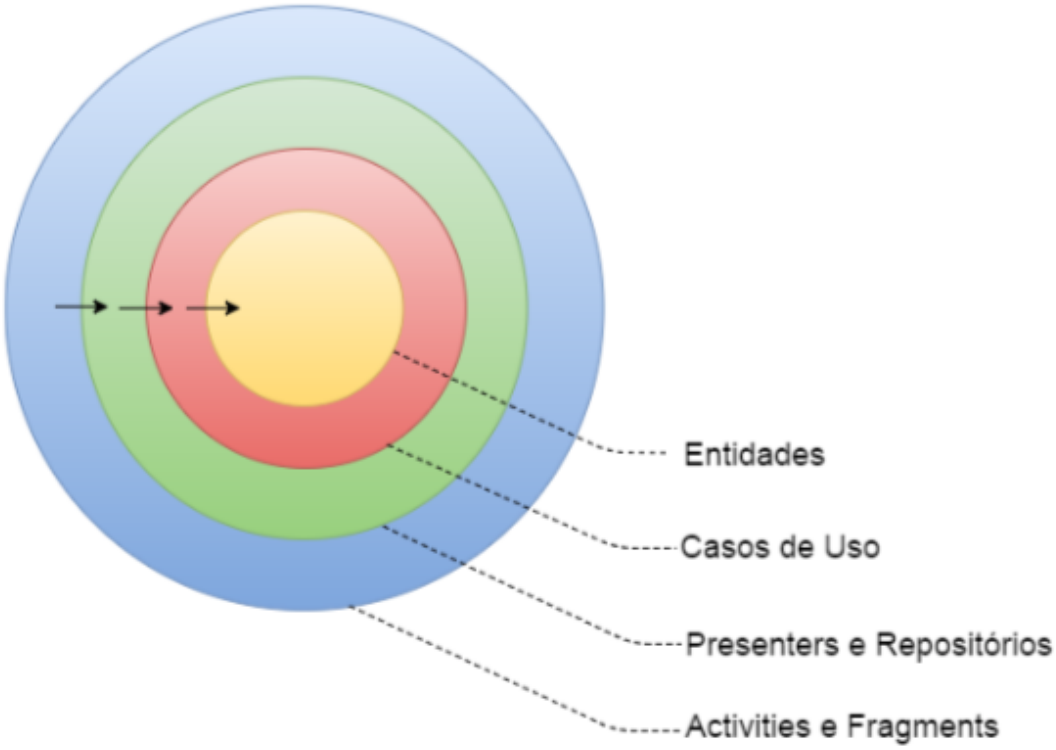

Figura 3.5: Camadas da arquitetura [22].

- **Camada de entidades:** é a camada mais interna de sistema, lida com as classes que fazem parte dos modelos de domínio. Essa é a parte do sistema que envolve o contexto que o projeto se insere e que deve fazer parte de qualquer aplicação que envolva esse contexto. No caso do *Rockdroid*, pode-se citar as classes User, Project e Rock. Essas classes fazem parte tanto do projeto do aplicativo como do servidor Web utilizado;
- **Camada de casos de uso:** são as classes do sistema que representam as regras de negócio da aplicação. Essa é a camada que se comunica com a camada de entidades, criando as regras de uso e realizando processamentos em cima dessas entidades. Para o projeto aqui proposto, pode-se citar as classes que modelam a criação e atualização de projetos no aplicativo;
- **Camada de acesso:** pode-se considerar essa camada como o núcleo da aplicação. Esta é a camada que recebe informações das camadas inferiores e repassa para a camada superior, fazendo o controle de tudo que acontece no modelo. Ela fornece

uma interface de acesso às funcionalidades e esconde do usuário a maior parte da lógica da aplicação;

- **Camada de apresentação:** esta é a camada em que o usuário interage com o sistema. Ela é uma camada ditada pelo *framework* utilizado que, no caso deste projeto, é o *Android*. Os componentes principais dessa camada são as *Activities* e *Fragments*, que instanciam componentes como botões e campos de formulário e permitem a interação humano-computador. A camada de apresentação interage com a camada de acesso para chamada de funções a partir de uma interface. A camada de acesso conclui a interação chamando os componentes necessários das camadas inferiores para realização da operação requisitada pelo usuário.

3.5.2 Módulos

O ambiente de programação do *Android* permite que o projeto seja criado com um módulo principal e vários módulos conectados a ele na forma de bibliotecas, por meio da ferramenta de automação de *builds* conhecida como *Gradle* [35].

Com a criação desses módulos separados, garante-se a separação de camadas. Por questão de controle de dependências, uma classe interna não possui acesso a uma classe externa. Com isso, o aplicativo *RockDroid* possui quatro módulos distintos:

- **Módulo *app*:** este é o módulo mais externo e contém o mecanismo para injeção de dependências que provê as referências que são necessárias para criação das classes no sistema. Nesse módulo se encontra componentes do *framework* do sistema *Android* que fornecem mecanismos para a criação de telas no aparelho móvel, como *Activities*, *Fragments* e *Dialogs*. Essas telas são construídas a partir da definição de elementos em um arquivo no formato XML. Os *layouts* do sistema são construídos passando-se esses arquivos XML como argumento da função do sistema operacional Android responsável por construir nativamente esses elementos na tela do aparelho. Esse é o módulo que forma a camada de apresentação;

- **Módulo *domain*:** esta camada foi projetada para depender apenas das bibliotecas Java utilizadas e para facilitar o processamento das regras de negócio. Ela lida com as componentes do domínio da aplicação, formando a camada de casos de uso e a camada de entidades;

- **Módulo *presentation*:** este módulo faz a ligação entre o módulo de interface com as regras de negócio do sistema. Ela é a responsável por fazer requisições às camadas internas e mapear as respostas às camadas externas de apresentação. Esse módulo forma a camada de acesso;

- **Módulo *data*:** esta módulo depende do *framework Android* por ser responsável por acessar os dados armazenados no aparelho e nos repositórios. Essa classe acessa o banco de dados local, *SQLite*, e o banco de dados remoto através de classes que fazem as requisições Web, arquivos em memória e preferências do *Android*. Esse módulo junto com o *presentation* completam as classes da camada de acesso da arquitetura.

3.5.3 Model-View-Presenter (MVP)

A arquitetura MVP é uma derivação da arquitetura mais conhecida *Model-View-Controller* (MVC), comum de ser usada para criação de interfaces gráficas. No MVP as telas também conhecidas como *Views* podem ser consideradas passivas, e não possuem função além de chamar a camada de controle conhecida como *Presenter*. A classe *Presenter* fornece respostas assíncronas aos eventos lançados pela *View*.

Com a implementação desse padrão, as *views* ficam responsáveis apenas por tratar os atributos do próprio *layout* do *Android*. Assim, as *views* não precisam conhecer a lógica de negócio diretamente, o que simplifica as responsabilidades de cada camada.

Pela Figura 3.6, é possível ver como se organiza as camadas na arquitetura MVP em que as componentes de interface View correspondem ao módulo app e o *Model* correspondem ao módulo *domain*. O módulo *presentation* tem as componentes dos *Presenters*, que recebe as ordens das *views* e envia para o Model se necessário, posteriormente devolvendo as respostas.

Da forma como a arquitetura foi utilizada no projeto, a *View* tem controle da navegação entre as diferentes telas do sistema. Desse modo, os *Presenters* respondem sempre a classe de apresentação que os chamou. Com essa conexão criada entre a *View* específica e o *Presenter*, é possível uso de vários *Presenters* projetados para uma mesma *View*.

Inversão de dependências

A inversão de dependência tem o propósito de permitir que uma camada interna do projeto possa utilizar algum serviço de uma camada mais externa a essa, sem destruir a proposta da arquitetura MVP.

Este principio se baseia no uso de classes de interface no Java para oferecer uma abstração dos módulos mais internos e ser implementado nos módulos mais externos. Com isso, o módulo interno não fica com suas dependências presas ao módulo externo e pode ser compilado de forma simples, mas em tempo de execução o fluxo de dependência se inverte.

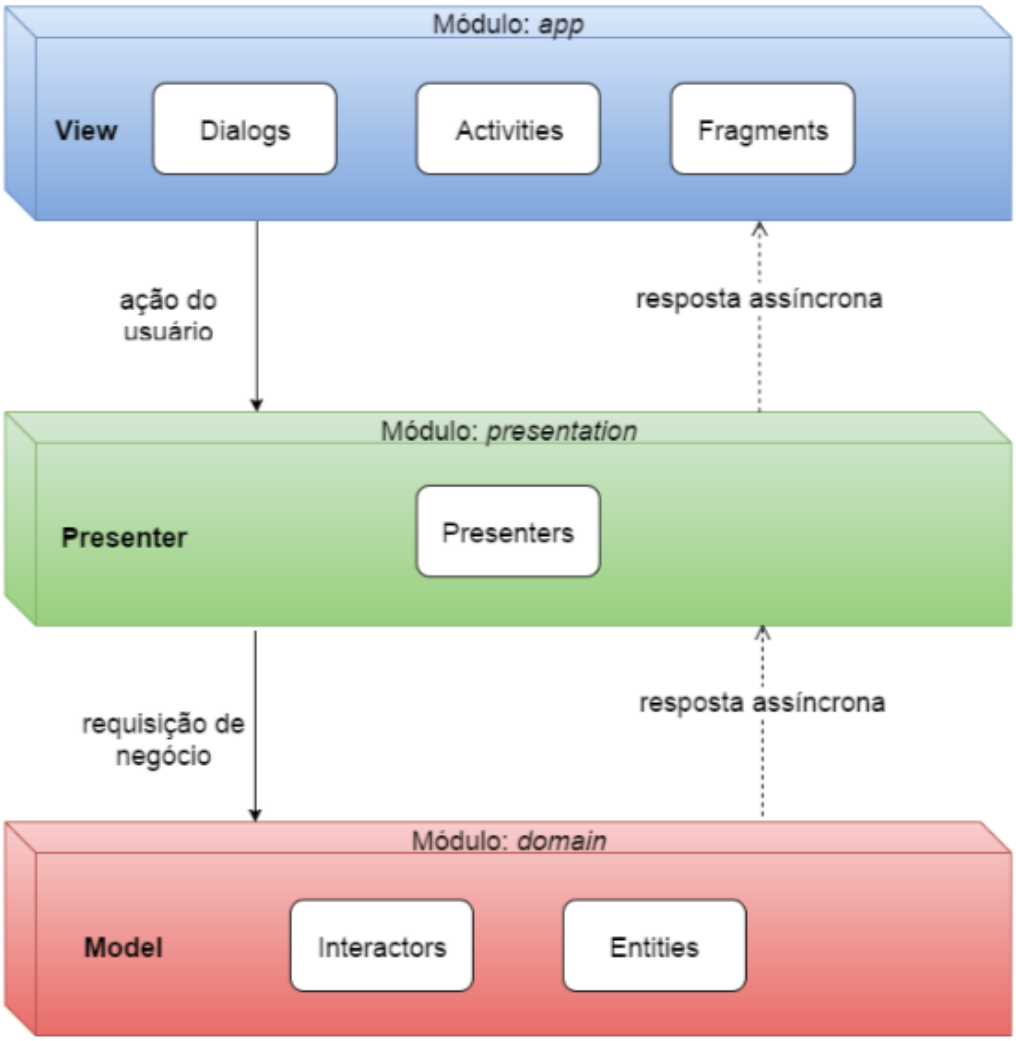

Figura 3.6: Visão do padrão MVP integrado aos módulos da arquitetura [22].

No *RockDroid* usa-se essa inversão de dependência para que o módulo de domínio utilize os recursos do módulo data, recupere dados dos vários repositórios sem precisar criar uma dependência do módulo de data ao módulo de domínio.

A Figura 3.7 mostra, em tempo de execução, o fluxo de chamadas dos métodos. Acompanhando o fluxo de chamadas, o primeiro módulo a ser chamado, depois de uma interação do usuário com a interface gráfica, é o módulo *app*. Na sequência esse módulo passa uma mensagem para o módulo *presentation*, que chama uma determinada lógica de negócio contida no módulo *domain* e, caso seja necessário a obtenção de dados para processar esse chamado, continua o fluxo para o módulo *data*.

Independente das chamadas feitas pelas camadas, a resposta final sempre seguirá o fluxo inverso: passando de um módulo para outro, *data, domain, presentation* e chegando

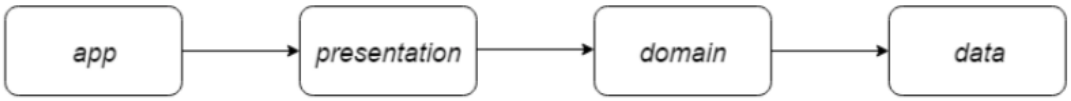

Figura 3.7: Fluxo de dependências entre os módulos em tempo de execução [22].

ao usuário pelo módulo *app*.

Com a inversão de dependência no projeto e em tempo de compilação, a organização muda como visto na Figura 3.8. Com a aplicação desse padrão, o módulo *domain* não depende dos outros módulos. Assim, ele possui, além das classes que lidam com sua lógica de negócio, interfaces para os serviços que ele utiliza. Estes serviços são implementados no módulo data que, consequentemente, depende do *domain*. Assim o mecanismo de injeção fornece todas as referências as classes concretas do serviço.

A Figura 3.8 apresenta um diagrama de classes utilizadas no *RockDroid*, em que pode-se ver as dependências necessárias para processamento dos casos de uso que estão armazenados no módulo *domain*. Na hora da execução, quando é instanciado, recebe em seu construtor a instância concreta da classe provedora através da injeção de dependências.

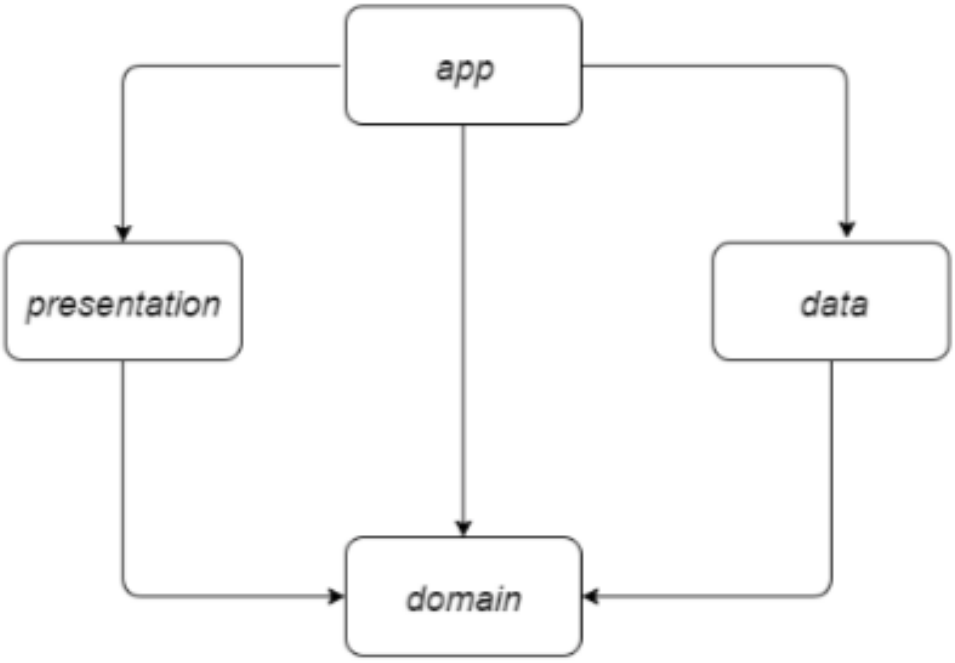

Figura 3.8: Dependências entre os módulos em tempo de compilação [22].

Injeção de dependências

Observando a Figura 3.8 pode-se encontrar a dependência do módulo *app* com o módulo *data*, o que em princípio parece em desacordo com o padrão MVP a ser seguido. Pelo padrão MVP, nenhuma classe *View* deve utilizar os serviços do domínio data diretamente.

O desacordo com o padrão MVP, porém, não existe. A responsabilidade do módulo principal *app* é de promover os mecanismos necessários para a injeção de dependência entre os módulos para existir a conexão dos mesmo em tempo de execução.

Conhecendo todas as classes do sistema, o *app* cria as instâncias e repassa por parâmetro, conectando todos os componentes as implementações dos serviços que forem usados.

3.6 Desenvolvimento

O aplicativo *RockDroid* teve foco na qualidade do código e a forma como foi organizado facilitou a criação das novas funcionalidades propostas nesse projeto, sem comprometer outras partes do projeto.

3.6.1 Softwares Utilizados

Os softwares de apoio ao desenvolvimento utilizados nesse projeto se mantiveram próximos aos usados na primeira versão do *RockDroid*. A IDE *Android Studio* foi utilizada para codificação do aplicativo. O *Git* para o controle de versão, mas dessa vez com auxílio do *BitBucket* como hospedeiro do repositório do aplicativo [1]. O *BitBucket* é uma solução Git para equipes profissionais, criando uma visualização do código em nuvem e a possibilidade de criar comentários nos códigos de forma grátis para equipes de até cinco membros. Como interface gráfica de auxílio ao uso do *Git* foi utilizado o *SourceTree* [4]. O *SourceTree* é um programa que pode ser instalado no computador que possibilita uma interface gráfica para facilitar ações antes feitas por códigos no terminal do sistema operacional.

3.6.2 Desenvolvimento da Funcionalidade

O primeiro passo do desenvolvimento foi entender a estrutura do projeto. O projeto tem um estrutura em camadas com vários módulos e, em princípio, houve uma dificuldade de se compreender o funcionamento e a lógica de cada camada. Passada essa fase inicial, foi possível entender a estrutura já explicada na seção anterior dividida em: módulo *domain* com a lógica de negócio, módulo *data* com as classes *Android* para comunicação entre as

fontes de dados, módulo *presentation* que contém os *Presenter* usados pelo padrão MVP e, por fim, o módulo *app* que contém as classes de interface com o usuário e mecanismo de injeção de dependência.

Além disso, estudou-se o modelo de dados aplicado no banco de dados local *SQLite*, como cada módulo funciona para obter os dados armazenados nesse banco e como foi estabelecida a sincronização de dados entre o aplicativo e o servidor Web que contém o banco de dados central.

Com a análise de todo o ambiente do projeto, iniciou o desenvolvimento dividido em três etapas. Primeiro, criou uma rede *ad hoc* na qual os aplicativos devem fazer parte para poder transferir dados de um aparelho móvel para outro. Seguindo a documentação disponibilizada pelo Google, uma conexão é criada entre os aparelhos por meio do protocolo *Wifi Direct* [7]. Por meio do uso do protocolo, um aparelho torna-se o dono do grupo Wifi e os outros aparelhos se tornam clientes desse grupo, o que faz com que os aparelhos saibam apenas o IP do dono do grupo, facilitando o envio de dados para o dono do grupo, porém dificultando o envio de dados para os outros clientes do grupo.

Para poder sincronizar um projeto salvo no banco de dados local do *RockDroid*, foi necessário criar um serviço que é chamado no início da conexão *Wifi Direct*. Esse serviço cria no aparelho um *socket*, um ponto final de um fluxo de comunicação entre processos através de uma rede de computadores, que funciona como um servidor local esperando requisições dos clientes. Inicialmente, um cliente envia para o servidor do aparelho, dono do grupo, um pedido para que esse armazene seu *Internet Protocol* (IP) e saiba com quem ele pretende se comunicar.

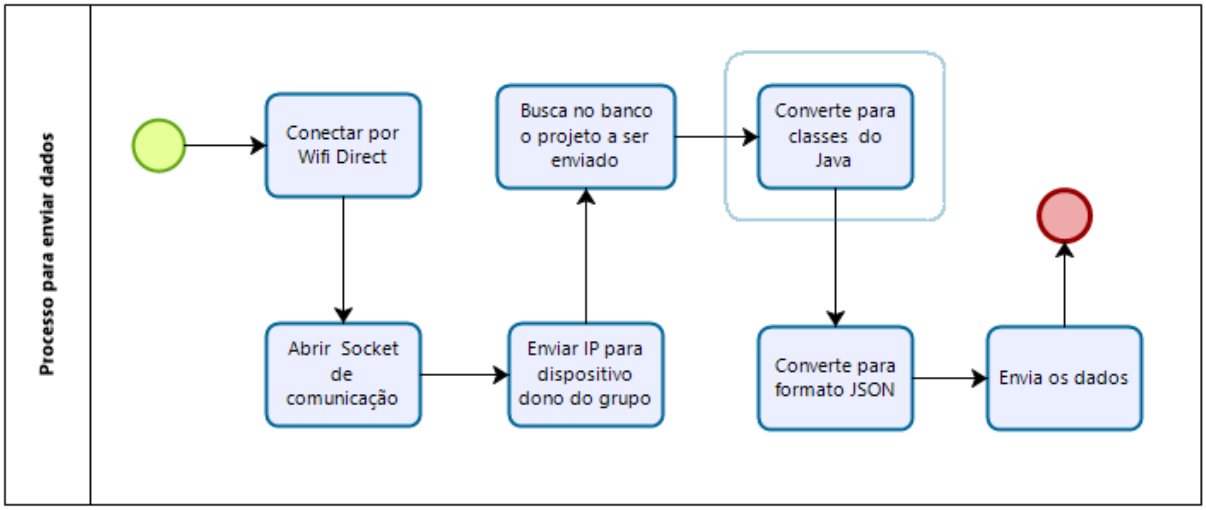

Figura 3.9: Processo de envio de dados.

Concluída a conexão entre os aparelhos e a criação do servidor no dispositivo, pode-se

passar a segunda etapa e dar importância ao envio dos dados entre os aparelhos. Na interface gráfica, o usuário escolhe um projeto para ser enviado e, depois da fase de conexão completa, consulta-se no banco de dados local os dados que pertencem aquele projeto para seu envio, o processo de envio é visto na Figura 3.9. Como não é possível enviar as classes do Java, existe a necessidade de *Serializar* os dados presentes nesse projeto em um formato que possa ser facilmente enviado por meio de requisições Web. Entre os formatos mais utilizados para requisições, XML e JSON, o JSON foi escolhido por ser um formato mais rápido e por utilizar menos recursos do dispositivo [28]. O JSON foi projetado para ser uma linguagem de troca de dados legível e fácil de ser processada e usada pelos computadores [29].

Para auxiliar nessa tarefa uma biblioteca disponibilizada pelo Google chamada *Gson* foi usada [2]. Essa biblioteca permite que seja passada a classe Java e o objeto contendo os dados que se deseja *serializar* como parâmetros de uma função, que *serializa* o objeto para o formato JSON sem prejudicar a estrutura da classe. Com o objeto JSON criado, o cliente envia para o dispositivo receptor o dado do projeto no formato de texto. Quando recebido, o dispositivo móvel transforma o texto recebido em um objeto JSON usando a biblioteca *Gson* para *desserializar* os dados e transformar novamente em uma classe Java, que é facilmente utilizada pelo *Android*.

A última etapa deste processo foi salvar o projeto recebido, mantendo como usuário do projeto o usuário que iniciou a criação daquele dado e modificando a interface gráfica criada anteriormente que mostrava apenas os projetos do seu próprio usuário. Agora a interface gráfica deve mostrar os projetos de todos os usuários que enviaram seus dados para aquele dispositivo móvel, o processo de receber os dados pode ser visto na Figura 3.10.

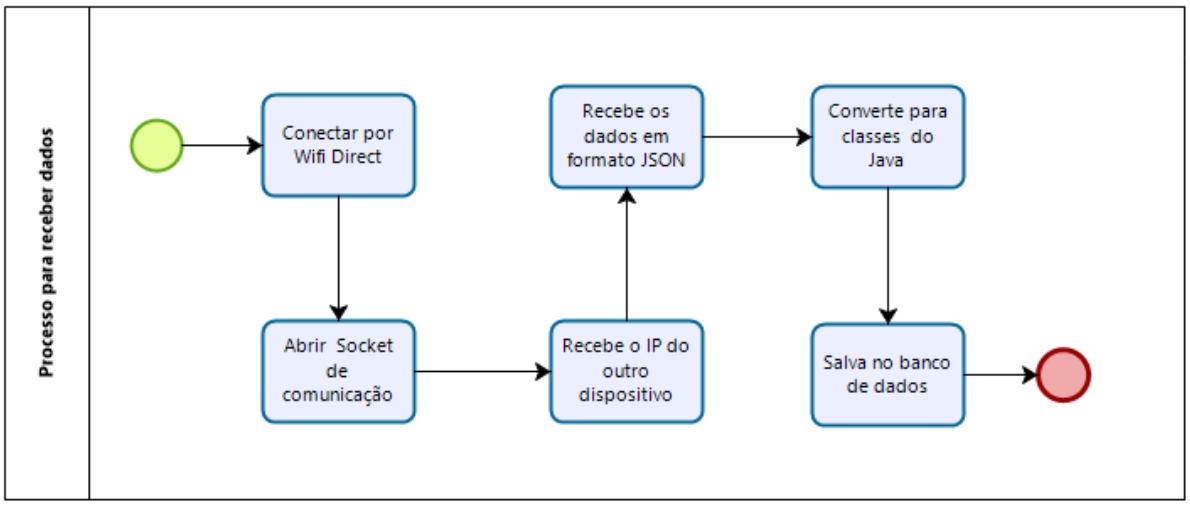

Figura 3.10: Processo para receber dados.

Com a última etapa concluída, o usuário pode optar por sincronizar os dados que ele recebeu de outro dispositivo móvel, ou que ele modificou durante o uso do aplicativo com o servidor central usando a Internet.

3.6.3 Sincronização

Com a atualização do aplicativo, a sincronização pode acontecer de duas formas distintas. Primeiramente, entre diferentes dispositivos móveis que usam o aplicativo por *Wifi Direct*. Posteriormente, entre o aplicativo de um usuário e o servidor central que armazena todos os dados sincronizados.

Para a sincronização funcionar de forma correta, existe um algoritmo que se baseia em algumas premissas. A mais importante delas é um identificador único e universal que permite que cada registro seja diferente, e não possa ser erroneamente removido ou atualizado em uma sincronização com identificadores iguais. Outras premissas definidas são o controle de atualização nos registros e o controle de deleções nos registros.

Quando um registro é gerado, um identificador único e universal é criado para esse registro. Isso acontece para evitar falhas no protocolo de sincronização, caso contrário durante uma sincronização entre o servidor e o aplicativo, o servidor seria obrigado a gerar um identificador para registro em seu banco de dados central e esse deveria ser salvo também no registro já gerado no aplicativo. Isso causaria um problema em que existiria dois identificadores para o mesmo registro, um deles sendo usado apenas pelo aplicativo e outro para sincronização com o servidor.

Um problema parecido aconteceria com a sincronização entre aplicativos de usuários diferentes. Se a sincronização entre aplicativos fosse feita antes de uma sincronização com o servidor, não teria um identificador no banco local que referenciasse o registro no banco de dados do servidor, o que geraria problemas de duplicidade de dados caso os dois aplicativos sincronizarem com o servidor central.

Para solucionar esse problema, é gerado um identificador único. O aplicativo se baseia em três valores distintos para gerar o identificador. O primeiro é um identificador do dispositivo, gerado pelo fabricante do aparelho, presente em todos os aparelhos *Android*. O identificador do dispositivo é, então, concatenado com com a data e hora do aparelho, transformado em milissegundos no momento de criação do registro. Por fim, esses valores são concatenados com uma sequência de caracteres hexadecimais de 64 bits, gerado aleatoriamente pelo aplicativo. Essa sequência de caracteres, que foi gerada pela concatenação dos três valores, pode ser considerada única e ser utilizada como identificador para os registros, tanto no aplicativo como no servidor central da Figura 3.11.

Com esse identificador único, pode-se fazer a sincronização usando *Wifi Direct*. O usuário de um aplicativo escolhe qual projeto registrado no seu banco de dados deseja

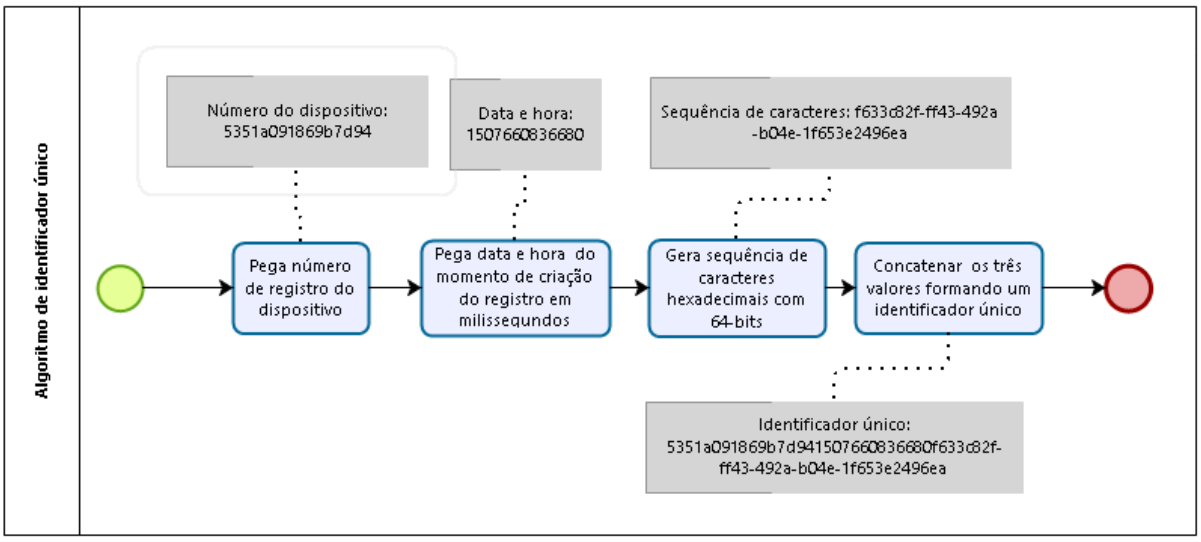

Figura 3.11: Algoritmo de criação do identificador único.

enviar e para qual outro usuário. Como o identificador é único para esse registro, verifica-se na hora que se recebe o registro se esse já existe no banco de dados local, caso não exista é criado um novo registro no banco de dados, sempre mantendo o identificador único do registro gerado pelo dispositivo. Caso o registro já exista apenas atualiza-se com os novos dados recebidos na sincronização. Na Figura 3.12 pode-se ver esse algoritmo de sincronização.

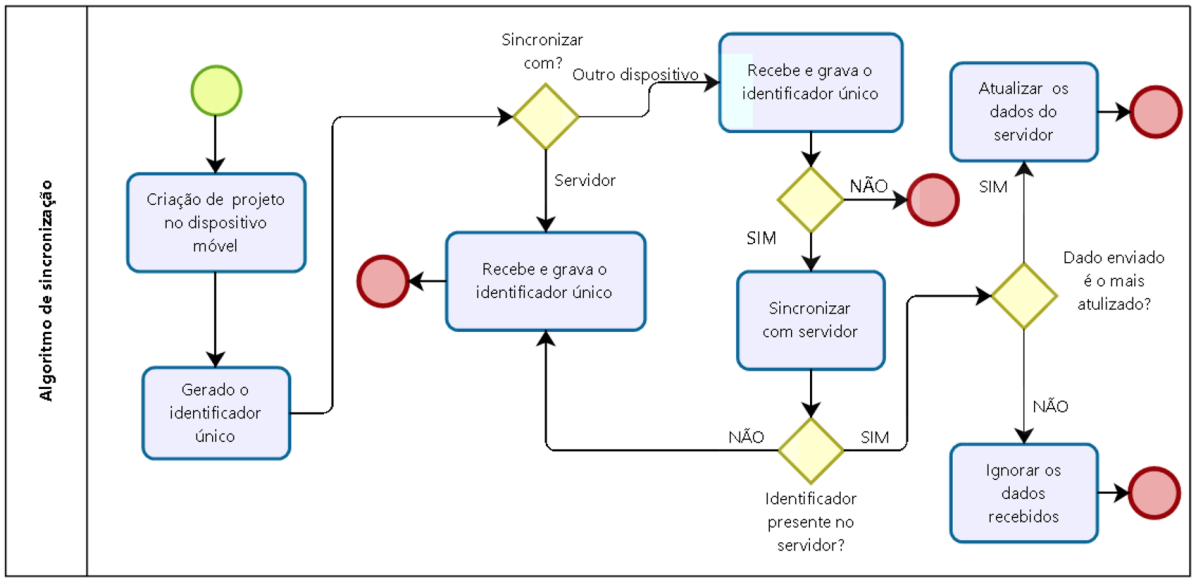

Figura 3.12: Algoritmo de sincronização com identificador único.

Para realizar o controle de atualizações, o registro possui um campo que identifica o

estado atual do registro no dispositivo móvel, indicando se foi sincronizado ou não com o servidor de banco de dados central. Quando um registro é gerado, esse campo fica com o estado atual para *falso*, informando que o dado não foi sincronizado. Quando esse dado é sincronizado com o servidor central, esse campo passa seu estado pra *verdadeiro* representando que a sincronização foi completa. Quando o usuário realiza qualquer alteração no dado, esse campo volta a seu estado inicial como *falso* e a sincronização pode ser feita novamente. Na Figura 3.13 pode ser visto o modelo de dados presente no banco de dados do dispositivo móvel. É possível perceber por esse modelo que em várias tabelas tem-se o campo *is in sync* utilizado para mostrar se o dado foi sincronizado com o servidor central.

Essa forma de sincronização funciona para o caso de um usuário ser o único que pode sincronizar aquele dado. Ao incluir o sincronismo entre os aplicativos, porém, esse mecanismo de sincronização com o banco de dados pode fazer com que dados de um aparelho que ainda não foi sincronizado acabem sendo substituídos por dados de um aparelho com informações mais atualizadas, que foram sincronizados primeiro. Esse problema pode ser exemplificado na Figura 3.14, no qual se houver dois usuários com o mesmo projeto sincronizado vão sincronizar com o servidor central em horários diferentes, criando dados desatualizados no servidor central.

Para resolver esse problema, algumas modificações foram implementadas e são vistas no modelo de dados da Figura 3.13, que foi criado um campo *updated at* e inicialmente sempre que um dado for atualizado, deve ser salvo sua data de atualização neste campo. Assim, quando um dado for enviado para o servidor central, compara-se sua data de atualização com a data de atualização salva no servidor. Se essa data for posterior a data no servidor, então esse dado pode ser atualizado. Dessa maneira, impede-se que dados desatualizados sobrescrevam dados mais recentes.

A sincronização entre os dados é unidirecional - os dados são enviados apenas do aplicativo para o servidor - assim o algoritmo de sincronização fica simples e existe apenas no cliente. Com a solução proposta de atualizar apenas os dados mais recentes, fica resolvida a maioria dos problemas, porém um novo problema é encontrado quando um projeto está sendo sincronizado entre vários usuários. Se um usuário enviar seu projeto para outro usuário e excluir o projeto enviado, o usuário que recebeu esse projeto nunca saberá que aquele projeto foi excluído e ficará com ele em sua lista de projetos até que seja manualmente excluído. Como projeto futuro é interessante que o *RockDroid* tenha uma sincronização bidirecional para resolver esse tipo de problema.

Para evitar que a sincronização crie registros que contenham chaves estrangeiras inválidas, as quais apontam para registros que ainda não existem, é necessário que a sincronização tanto entre aplicativos como entre o banco central ocorra numa ordem predefinida da hierarquia de classes. Com isso, os dados gerados nunca sofrerão erros de falta de

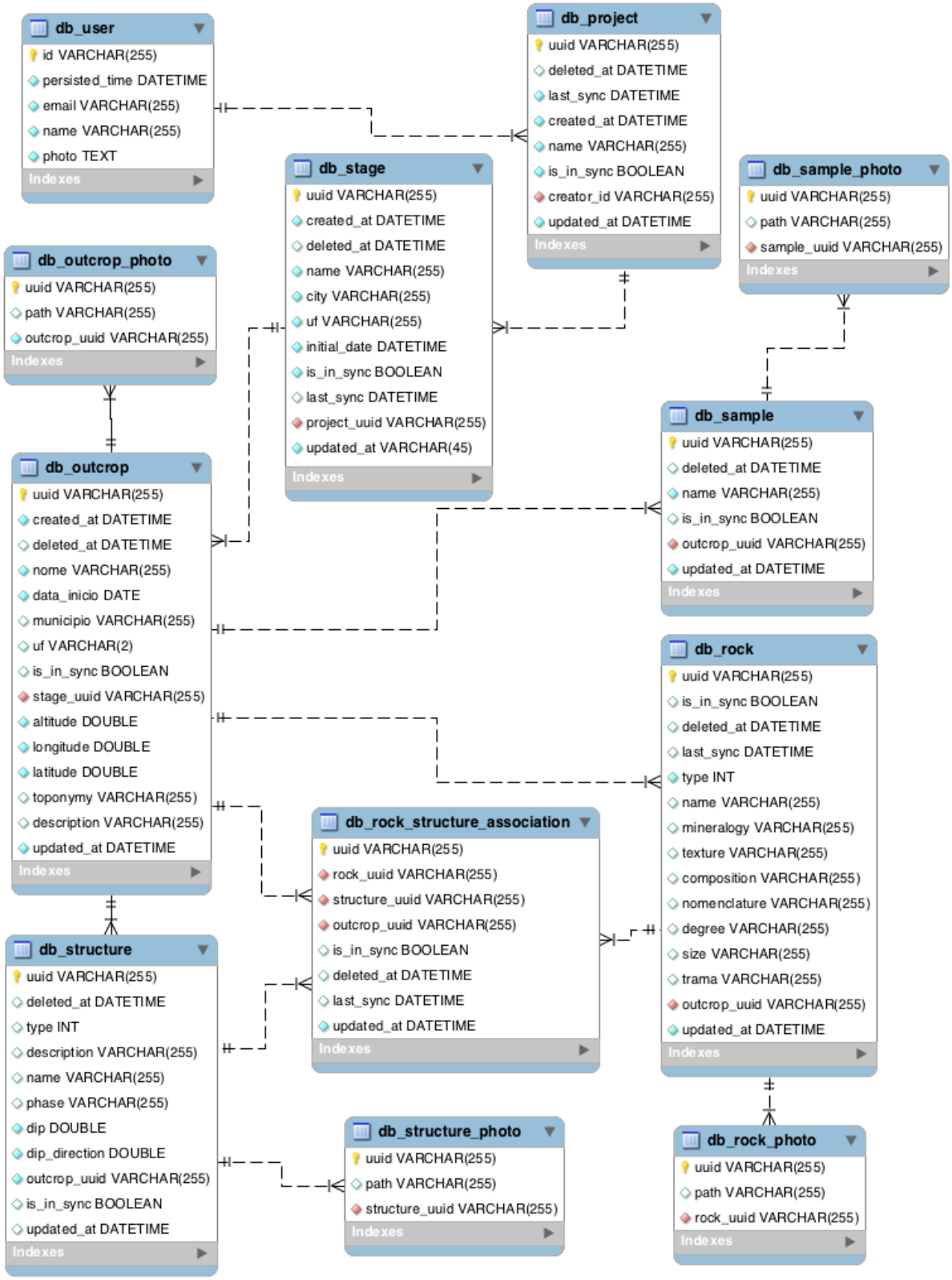

Figura 3.13: Modelo de dados do banco local.

Figura 3.14: Sincronismo entre aplicativos sem tratamento de atualizações.

chave estrangeira, e a sincronização funcionará de forma correta.

3.6.4 Desenvolvimento do Serviço Web

A infraestrutura existente foi temporariamente substituída por um serviço de nuvem chamado *Heroku* [3], que suporta a linguagem *Ruby* usada e auxilia no *deploy* da aplicação, facilitando assim o uso e atualização da aplicação durante seu desenvolvimento e testes.

Com o uso do servidor, algumas modificações foram feitas para solucionar alguns problemas. Um problema resolvido foi o de sincronização do mesmo dado entre os vários aparelhos. Esse problema causava substituição de um dado atualizado por um desatualizado em alguns casos de uso, garantindo sempre a sincronia do dado mais atualizado.

Uma lógica foi então desenvolvida no servidor para verificar qual é o dado mais recente enviado por um usuário. Quando um usuário pede a sincronização de um dado, é então enviado com este dado a data da última atualização no aplicativo. Caso a data de atualização seja menor que a data presente no servidor, o algoritmo ignora aquele dado. Assim, dados antigos não devem substituir dados recentes que já foram sincronizados por outro usuário no banco de dados central como pode ser visto na Figura 3.15.

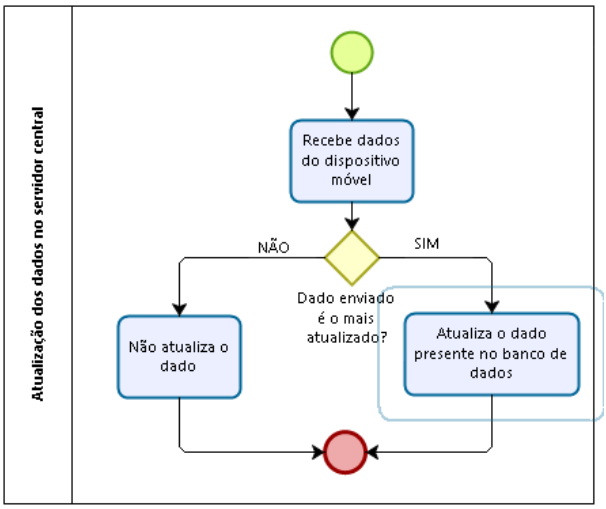

Figura 3.15: Algoritmo de atualização dos dados no servidor.

3.7 Telas do RockDroid

Ao finalizar o desenvolvimento da nova funcionalidade, foi inciada a fase de testes. Devido a pequenas mudanças que ocorreram no código, era necessário testar novamente a aplicação para garantir o funcionamento correto. Nesta seção são apresentadas as principais telas da nova funcionalidade, uma vez que as outras telas já foram apresentadas no trabalho inicial do *RockDroid*.

Listagem de projetos

Uma vez logado, o usuário será redirecionado para a tela de projetos como mostrado na Figura 3.16. Nessa tela, um ícone mostra se o projeto já foi sincronizado ou não. Além disso, cada projeto tem um botão que abre uma lista de opções de operações que podem ser executadas com o projeto. Entre as opções já existentes, foi adicianada a opção de 'Enviar para outro celular'. Essa é a opção responsável por redirecionar para a tela de sincronização, como visto na Figura 3.17.

Menu

Com o usuário logado, uma barra lateral contendo várias opções pode ser aberta. Esse menu possui uma nova opção 'Receber de outro celular' como na Figura 3.18. Essa opção deve ser aberta sempre que o usuário desejar receber uma projeto de outro usuário. Essa tela redireciona para a tela de sincronização como na Figura 3.19.

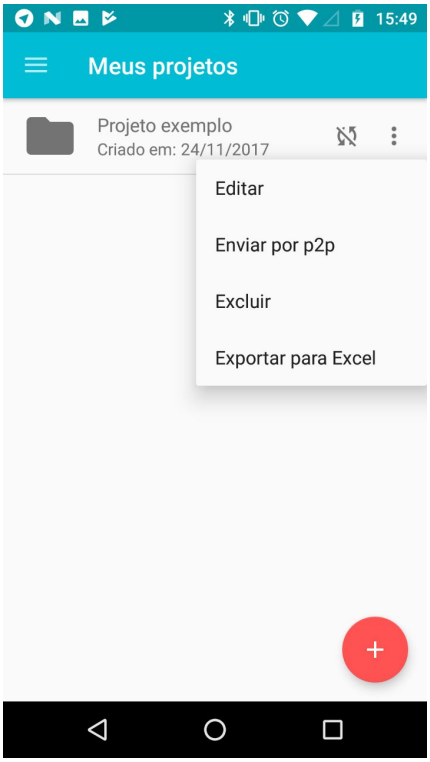

Figura 3.16: Listagem de projetos e menu.

Sincronização por *Wifi Direct*

Esta tela é dinâmica e funciona tanto para quem vai receber os dados enviados, quanto para quem está enviando os dados. Nela é exibida a lista de telefones próximos e algumas informações sobre como realizar a transferência. Clicando em um item da lista, um pedido para se conectar a um telefone por *Wifi Direct* aparece e recebe-se a mensagem 'Conectado', como visto na Figura 3.20. Com a conexão finalizada, clica-se em 'Envia' e completa-se o processo de envio, como visto na Figura 3.21.

Figura 3.17: Tela de envio de projeto.

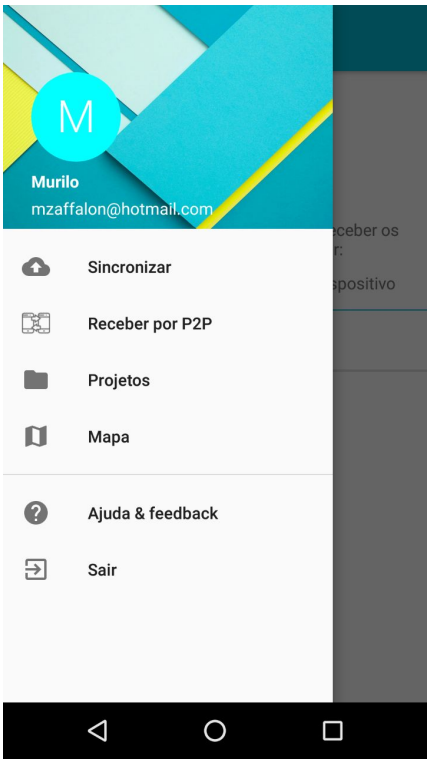

Figura 3.18: Tela de menu lateral.

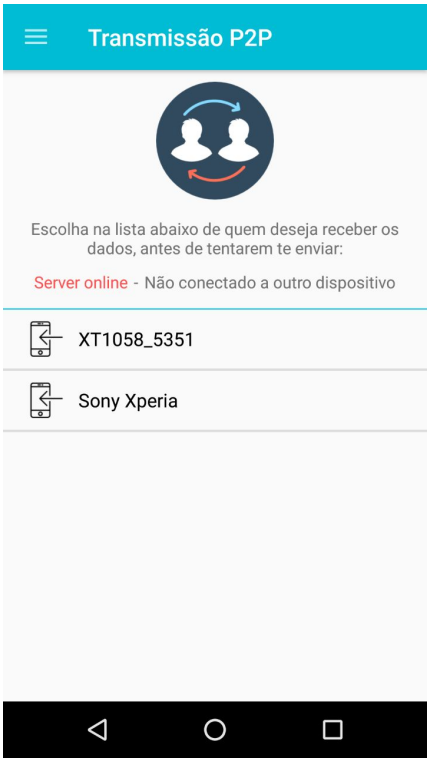

Figura 3.19: Tela de receber de outro celular.

Figura 3.20: Tela de mensagem conectado.

Figura 3.21: Tela de projeto enviado.

Capítulo 4

Testes e Validação

Neste capítulo são destacadas as metodologias de testes do aplicativo e de validação de sua usabilidade. Os testes que foram feitos são testes de conexão e de sincronização com diferentes dispositivos móveis e versões do sistema operacional Android.

4.1 Testes

Os testes de conexão foram implementados para demonstrar o comportamento esperado quando há uma limitação na conectividade por *Wifi Direct*, durante a utilização do aplicativo. Os testes de sincronização, por sua vez, apresentam os detalhes esperados no processo de sincronização de dados.

4.1.1 Ambiente Computacional de Execução de Testes

Para realizar os testes propostos, inicialmente, foram utilizados dois dispositivos móveis:

- Motorola Moto X (XT1098)
  - Sistema operacional: Android 5.1 (Lollipop)
  - Memória RAM: 2GB
  - Memória interna: 16GB
  - Processador: Dual-core 1.7 GHz Krait 300
- Sony Xperia
  - Sistema operacional: Android 4.3 (Jelly Bean)
  - Memória RAM: 1GB
  - Memória interna: 8GB

– Processador: 1.7 GHz Dual Core

Para hospedar o servidor central durante os testes, foi usado o serviço de nuvem da *Heroku* no seu pacote grátis [26]. *Heroku* remove todo o trabalho de instalação de softwares, manutenção, monitoramento, configurações de *deployment*, dentre outros.

4.1.2 Testes de Conexão

Os testes de conexão têm como objetivo verificar o comportamento da aplicação quando funcionalidades que exigem conexão entre os dispositivos por *Wifi Direct* está ou não estabelecida. Os seguintes testes foram realizados:

- Teste 1: Enviar projeto por *Wifi Direct* com aceitação de outro dispositivo;
- Teste 2: Enviar projeto por *Wifi Direct* sem conectar a outro dispositivo;
- Teste 3: Enviar projeto por *Wifi Direct* sem aceitação de outro dispositivo;
- Teste 4: Sincronizar projetos recebidos com o servidor central;
- Teste 5: Atualizar um projeto em dois dispositivos e sincronizar em ordem inversa;

Teste 1: Enviar projeto por *Wifi Direct* com aceitação de outro dispositivo

Para estes testes foi necessário realizar, primeiramente, o *login* com contas diferentes em dois dispositivos móveis. Um desses dispositivos, então, criou um projeto e adicionou algumas informações, enquanto o outro dispositivo ficou sem projetos. O dispositivo criador escolheu um projeto e clicou em "Enviar para outro celular". Assim uma nova tela é aberta, no qual é preciso escolher com qual aparelho se deseja tentar a conexão, como visto na Figura 4.1.

Enquanto isso, o outro dispositivo abriu o menu 'Receber de outro celular' e escolheu o dispositivo que pretende se conectar. Com a conexão estabelecida, o projeto é enviado de um aparelho ao outro. Como foi finalizado com sucesso, o usuário foi redirecionado para a tela de projetos. Recebido o projeto, o usuário foi capaz de sincronizar com o servidor central, clicando em 'Sincronizar' no menu lateral, como é possível observar nas sequência da Figura 4.2.

Teste 2: Enviar projeto por *Wifi Direct* sem conectar a outro dispositivo

Os mesmos passos do Teste 1 são seguidos, mas o segundo dispositivo nunca iniciou a tela de "Receber de outro celular". Neste caso, o aplicativo que desejou enviar o projeto ficou com a lista vazia, impedindo o envio do projeto.

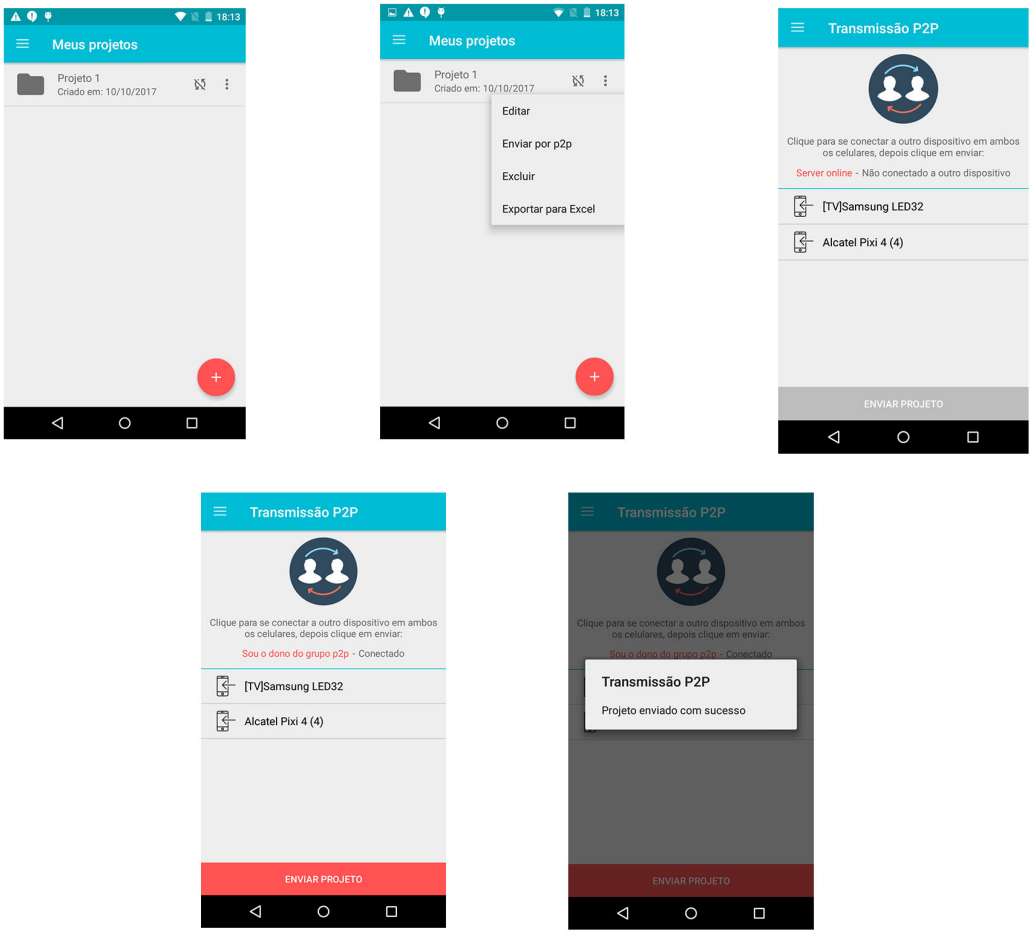

Figura 4.1: Sequência de telas de envio.

Teste 3: Enviar projeto por *Wifi Direct* sem aceitação de outro dispositivo

Quando o projeto vai ser enviado pelo dono do grupo no *Wifi Direct*, esse é o membro que faz o roteamento entre os dispositivos na rede. Ele não possui o IP de nenhum outro membro na rede, apenas o seu próprio. Por isso, ao tentar enviar um projeto para um outro dispositivo que não escolheu de quem deseja receber, a aplicação não permite o envio para o único IP conhecido, deixando o botão desabilitado. Isso torna a transferência impossível até que o outro usuário escolha o aparelho na lista.

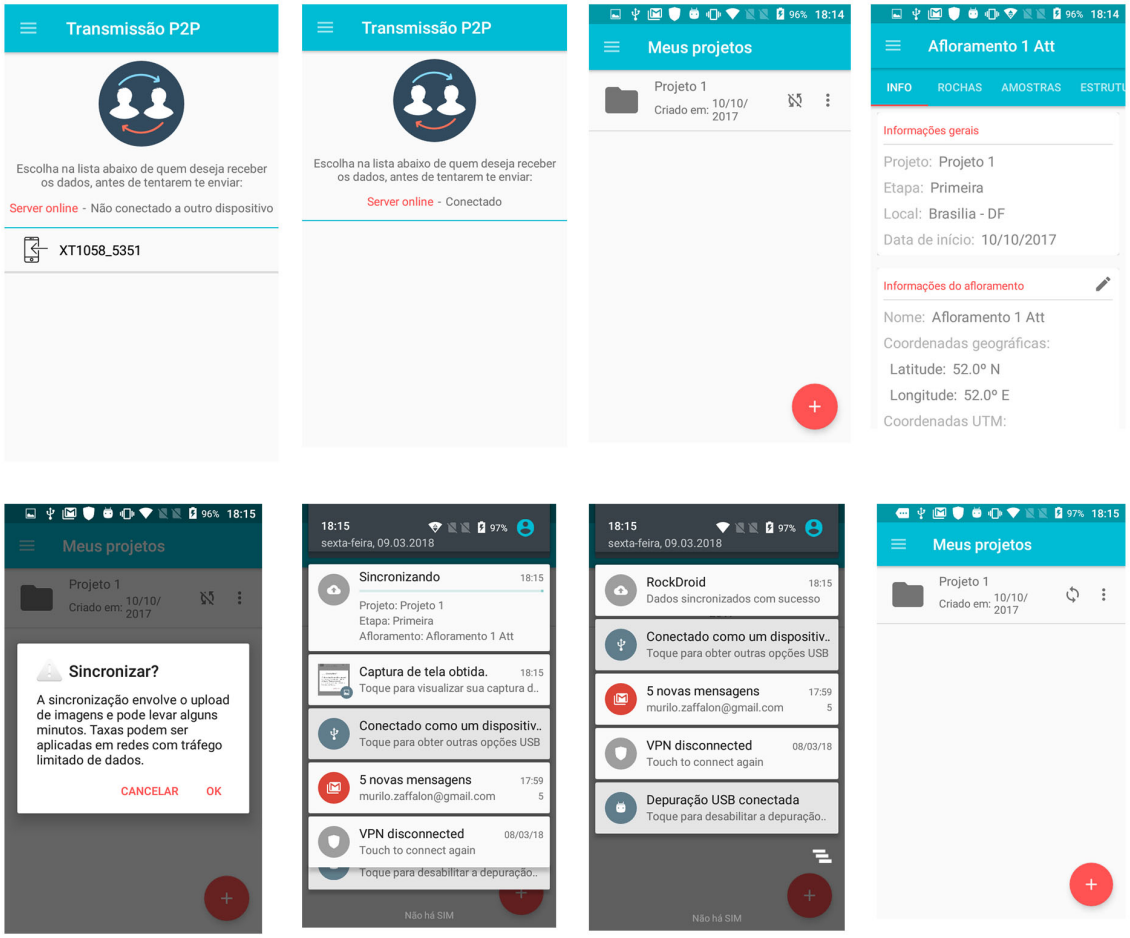

Figura 4.2: Sequência de telas de recebimento.

Teste 4: Sincronizar projetos recebidos com o servidor central

Este teste acontece depois do Teste 1. Quando um usuário recebe um projeto, consegue ver o projeto em sua lista de projetos. O usuário que recebeu esse projeto, então, abre o menu e clica em "Sincronizar". Uma mensagem de confirmação de sincronização aparece, e no servidor web pode-se verificar que os dados foram atualizados com sucesso.

Teste 5: Atualizar um projeto em dois dispositivos e sincronizar em ordem inversa

O Teste 5 é uma variação do Teste 4, em que alguns erros poderiam ocorrer caso não fossem devidamente tratados. Um usuário envia o dado para outro e, logo após esse envio, atualiza o dado. O outro usuário, então, recebe esse dado e não realiza a sincronização com o servidor central. Com isso, o usuário que enviou o dado e o atualizou decide fazer a sincronização desse dado com o servidor web. O dado desse usuário, então, pode ser considerado o mais atualizado e recente.

O destinatário do dado resolve, então, sincronizar, porém possui uma versão menos atual desses dados. Caso fossem salvos, esses dados atrapalhariam a sequencia cronológica. Com essa segunda sincronia em andamento, o servidor central verifica as datas de atualização dos dados e não permite a atualização de dados antigos.

4.2 Validação

Finalizado os testes mencionados anteriormente, algumas simulações de uso do aplicativo foram feitas. Como o projeto foi finalizado antes de uma saída de campo por parte da equipe do Instituto de Geociências da Universidade de Brasília (IG/UnB), não foi possível uma validação da funcionalidade com os pesquisadores do IG/UnB.

Para validar o projeto, uma equipe de três pessoas foi formada e viajou para o Município de Alto Paraíso dentro da Chapada dos Veadeiros, onde a validação do aplicativo foi feita. Para os testes, foi escolhido a Cachoeira das Loquinhas que possui várias quedas durante seu percurso e duas trilhas diferentes, a visualização por satélite pode ser vista na Figura 4.3.

No início do percurso, o líder da expedição criou um projeto e enviou para os dois outros membros do grupo, os dispositivos utilizados são Galaxy S5, LG X Power Dual e Motorola Moto X. O líder do projeto, então, ficou no inicio das trilhas, enquanto cada um dos outros membros seguiu por uma das trilhas, como pode ser visto na Figura 4.4.

Cada membro fez registros no aplicativo durante o caminho. Depois de determinado tempo, os membros voltaram ao ponto inicial e cada um deles enviou ao líder da expedição o seu projeto atualizado. Na Figura 4.5 pode-se ver todos os pontos de registro feito pelos pesquisadores. Verificado que não ocorreu nenhuma perda de dado, o líder da expedição sincronizou todos os dados com o servidor central.

Com o final da expedição, pode-se perceber que o sincronismo de dados entre os dispositivos móveis e a sincronia dos dados no servidor central facilitam a posterior análise e visualização dos dados, já que esses dados estão agregados em um único projeto, o que ajuda na obtenção rápida e sem replicação das informações presentes no projeto.

Figura 4.3: Localização da Cachoeira das Loquinhas.

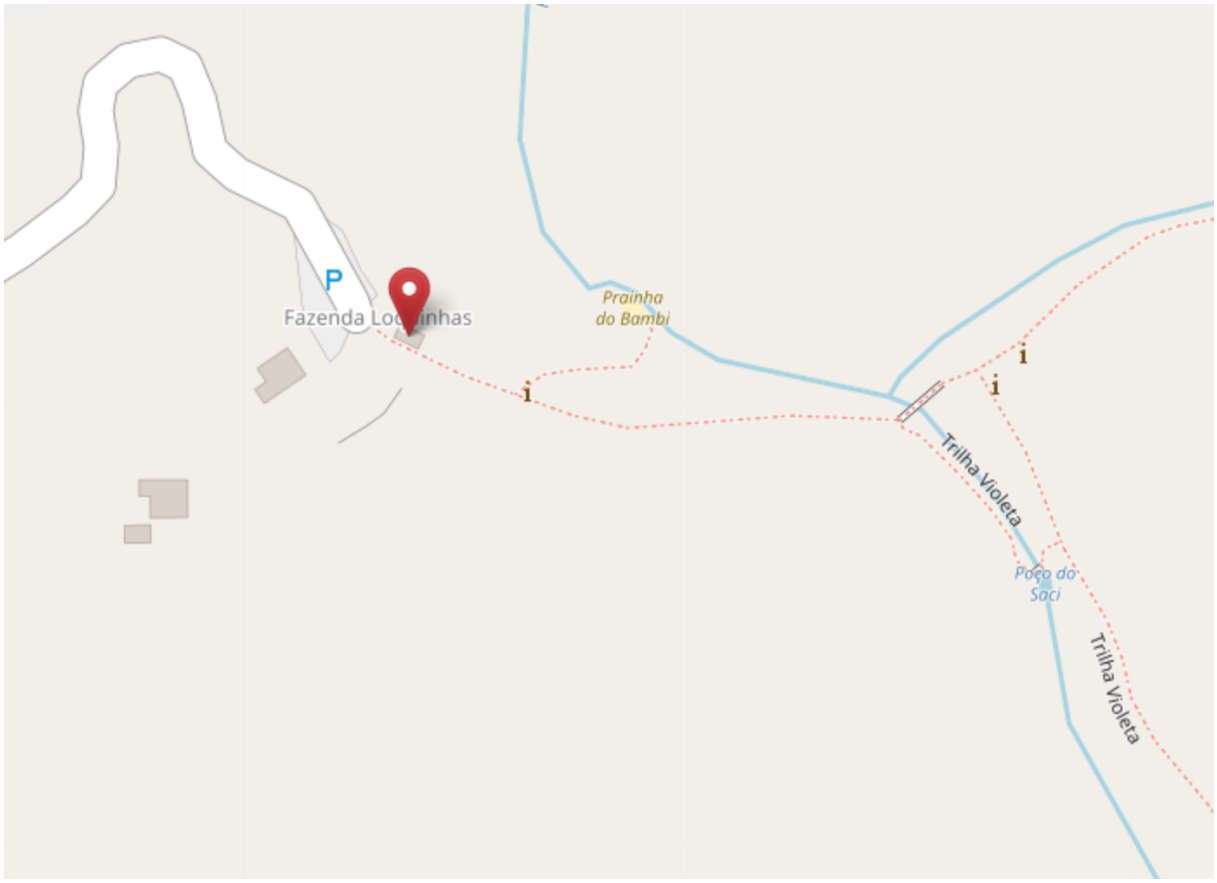

Figura 4.4: Posição do líder da equipe.

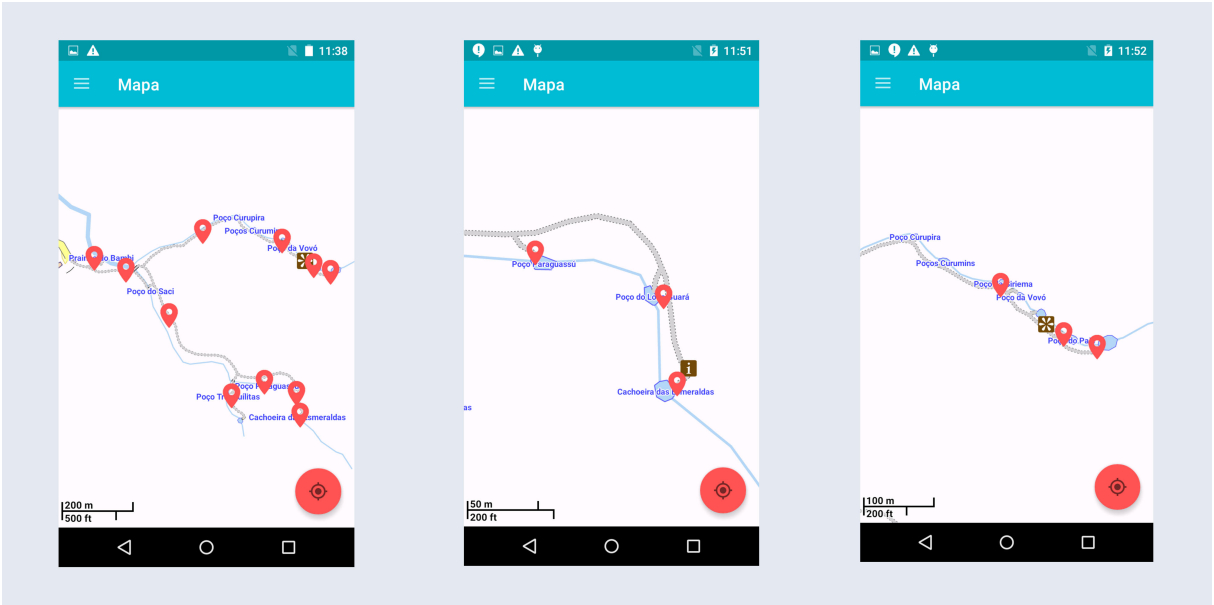

Figura 4.5: Pontos de registro mostrados no dispositivo móvel do líder.

Capítulo 5

Conclusões

O *RockDroid* vem para promover uma solução eficiente para os geólogos, criando uma interface gráfica nos dispositivos móveis que padroniza a coleta de dados e a sincronização destes entre os usuários para análise dos mesmos, tirando a necessidade de anotações em papel e trazendo toda a facilidade do mundo digital. A união das diferentes ferramentas disponibilizadas no aplicativo traz vários benefícios aos pesquisadores, facilitando o trabalho em campo.

O prévio desenvolvimento focado na conformidade com a arquitetura projetada criou uma base bem estruturada para o desenvolvimento deste projeto, facilitando a criação da nova funcionalidade seguindo toda a arquitetura proposta, e alterações no projeto que foram necessárias durante o desenvolvimento.

A construção em paralelo com esse projeto de um novo servidor *Web RESTful* facilitou a análise dos dados, já que foi implementado uma visualização dos dados contidos no banco do servidor. O novo servidor impediu que uma nova versão do *RockDroid* fosse atualizada na plataforma de aplicativos *Google Play*. A expectativa é que, com o fim do desenvolvimento do servidor Web, uma URL confiável seja decidida para apontar para o servidor e, com isso, a atualização do aplicativo seja possível.

A sincronização de dados entre aplicativos foi o foco fundamental deste projeto e foi desenvolvida para funcionar independente de ambiente e conexões com a Internet. E isso possibilita um melhor e mais eficiente trabalho dividido em campo. Além disso, as sincronizações com o servidor foram facilitadas quando os dados são transmitidos para um usuário que possua acesso a Internet. Nessa implementação, impedir que um dado desatualizado de um usuário sobrescreva um dado atualizado de outro usuário foi um dos principais desafios.

Com o desenvolvimento deste projeto, pode-se concluir que a nova funcionalidade para o *RockDroid* satisfez os objetivos propostos. Todos os passos de desenvolvimentos e

arquitetura expressos na primeira versão do *RockDroid* foram mantidos, com a criação de novas interfaces gráficas implementadas para auxiliar os usuários do aplicativo.

Durante o desenvolvimento deste projeto, a necessidade de uma nova funcionalidade ficou cada vez mais evidente. Quando sincronizado os dados dos dispositivos móveis com o banco de dados central, pode-se ter um dado desatualizado em certo dispositivo. Isso acontece porque a sincronização é unidirecional e apenas o servidor central recebe os dados. Para uma futura atualização, uma importante funcionalidade a ser implementada seria a sincronização bidirecional. Com isso, em situações de sincronização, o servidor receberia os dados atualizados de um dispositivo, mas caso esse dispositivo estivesse desatualizado, o servidor enviaria os dados atualizados para esse dispositivo, mantendo assim os dados atualizados em todas as fontes.

Além disso, um grande diferencial para implementação futura seria um painel para visualização e administração dos dados adicionados. Este, porém, já encontra-se em desenvolvimento, e vai acrescentar ainda mais facilidades e funcionalidades para os usuários que utilizam o aplicativo *RockDroid*.

Um outro trabalho futuro proposto é a verificação da segurança do pesquisador em campo, algo que ficaria muito complexo para o contexto do aplicativo devido, principalmente, à falta de Internet na maior parte das áreas de coleta. Uma versão reduzida dessa funcionalidade, porém, poderia acontecer em locais com Internet e uso da interface Web, em que seria exibida a posição exata de cada usuário em campo, podendo ajudá-los em caso de emergência.

Referências

[1] *BitBucket the git solution for professional teams.* `https://bitbucket.org/product`. Accessed: 2017-12-20. 27

[2] *Google gson.* `https://github.com/google/gson`. Accessed: 2017-12-28. 29

[3] *Heroku cloud application plataform.* `https://www.heroku.com`. Accessed: 2017-01-12. 34

[4] *Sourcetree free git gui for mac and windows.* `https://www.sourcetreeapp.com`. Accessed: 2017-12-20. 27

[5] *Sqlite (2013), author=htttp://www.sqlite.org/, year=2013, publisher=SQLite.* 7

[6] Alliance, Wi Fi: *Wi-fi certified wi-fi direct: personal, portable wi-fi technology white paper*, 2010. 8, 9, 10

[7] Asadi, Arash e Vincenzo Mancuso: *Wifi direct and lte d2d in action.* Em *Wireless Days (WD), 2013 IFIP*, páginas 1–8. IEEE, 2013. 28

[8] Association, GSM *et al.*: *The mobile economy 2016.* URL: https://www.gsmaintelligence.com/research/?file=97928efe09cdba2864cdcf1ad1a2f58c&download[a 2016-11-30][WebCite Cache], 2015. 1

[9] Basagni, Stefano, Marco Conti, Silvia Giordano e Ivan Stojmenovic: *Mobile ad hoc networking.* John Wiley & Sons, 2004. 2

[10] Connolly, Thomas M e Carolyn E Begg: *Database systems: a practical approach to design, implementation, and management.* Pearson Education, 2005. 5

[11] Crespo, Arturo e Hector Garcia-Molina: *Semantic overlay networks for p2p systems.* Em *International Workshop on Agents and P2P Computing*, páginas 1–13. Springer, 2004. 7

[12] Elmasri, Ramez: *Fundamentals of database systems.* Pearson Education India, 2008. 7, 8

[13] Elmasri, Ramez, Shamkant B Navathe, Marília Guimarães Pinheiro, Claudio Cesar Canhette, Glenda Cristina Valim Melo, Claudia Vicci Amadeu e Rinaldo de Oliveira Morais: *Sistemas de banco de dados.* 2005. 2, 5

[14] Ferreira, João Eduardo e Marcelo Finger: *Controle de concorrência e distribuição de dados: a teoria clássica, suas limitações e extensões modernas.* IME-USP, 2000. 6

[15] Figueiredo, Carlos MS e Eduardo Nakamura: *Computação móvel: Novas oportunidades e novos desafios*. T&C Amazônia, 1(2), 2003. 1

[16] Hipp, Richard: *Sql for the iot*. 2015. 7

[17] Ito, Giani Carla, Maurício Ferreira e Nilson Sant'Ana: *Computação móvel: Aspectos de gerenciamento de dados*. Instituto Nacional de Pesquisas espaciais–INPE, 2003. 4, 5, 6

[18] Korth, Henry F: *Sistema de banco de dados*. Makron Books/McGraw-Hill, 1994. 4

[19] Latiff, LA, A Ali Ahmed, N Fisal e SA Arifin: *Directional Routing Protocol in Wireless Mobile Ad Hoc Network*. INTECH Open Access Publisher, 2010. ix, 8, 9

[20] Liyan, Chen: *Application research of using design pattern to improve layered architecture*. Em *Control, Automation and Systems Engineering, 2009. CASE 2009. IITA International Conference on*, páginas 303–306. IEEE, 2009. 21

[21] Luz, José Wilker Pereira e Luís Carlos Costa Fonseca: *Educonnect: uma ferramenta de apoio à aprendizagem colaborativa para dispositivos móveis em redes manet*. Em *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 24, página 164, 2013. ix, 11, 12, 13

[22] Macêdo, Bernardo Augusto Pereira de e Renata Cristina Machado Nunes: *Rockdroid - uma arquitetura para coleta de dados geológicos*. 2016. ix, 2, 22, 25, 26

[23] Magalhães, Julia e Maristela Holanda: *Eiko: A social mobile network for manet*. Em *Information Systems and Technologies (CISTI), 2011 6th Iberian Conference on*, páginas 1–5. IEEE, 2011. ix, 11, 12

[24] Mansouri, Najme, Gholam Hosein Dastghaibyfard e Ehsan Mansouri: *Combination of data replication and scheduling algorithm for improving data availability in data grids*. Journal of Network and Computer Applications, 36(2):711–722, 2013. 6

[25] Mateus, Geraldo Robson e Antonio Alfredo Ferreira Loureiro: *Introdução à computação móvel*. DCC/IM, COPPE/UFRJ, 1998. 4

[26] Middleton, Neil e Richard Schneeman: *Heroku: Up and Running: Effortless Application Deployment and Scaling*. " O'Reilly Media, Inc.", 2013. 41

[27] Norman, Donald A e Stephen W Draper: *User centered system design*. New Perspectives on Human-Computer Interaction, L. Erlbaum Associates Inc., Hillsdale, NJ, 1986. ix, 17

[28] Nurseitov, Nurzhan, Michael Paulson, Randall Reynolds e Clemente Izurieta: *Comparison of json and xml data interchange formats: a case study.* Caine, 9:157–162, 2009. 29

[29] Org, Json: *Json*. Url: http://www.json.org, 2018. 29

[30] Potel, Mike: *Mvp: Model-view-presenter the taligent programming model for c++ and java*. Taligent Inc, página 20, 1996. 20

[31] Pritchett, Dan: *Base: An acid alternative*. Queue, 6(3):48–55, 2008. 5

[32] Rogers, Yvonne, Helen Sharp e Jenny Preece: *Interaction design: beyond human-computer interaction*. 2011. 18

[33] Schantz, Douglas, Paulo Silva, Rodrigo Antunes e Regis Rodolfo Schuch: *Sqlite para dispositivos móveis*. I Seminário de Pesquisa Científica e Tecnológica, 1(1), 2017. 6

[34] SparkPlug: *Wifidirect*. `https://play.google.com/store/apps/details?id=anuj.wifidirect&hl=pt_BR`, 2018. [Online; accessed 19-July-2018]. 12

[35] Vogel, Lars: *Android sqlite database and contentprovider-tutorial*. Java, Eclipse, Android and Web programming tutorials, 8, 2010. 5, 23

[36] Vogel, Lars: *Android sqlite database and content provider - tutorial*, 2016. `http://www.vogella.com/articles/AndroidSQLite/article.html`, acesso em 2016-10-13. 7

[37] Wright, Maury: *Wi-fi direct adds peer-to-peer capabilities to ubiquitous wireless network technology*. Techzone: Wireless Solutions, Date Unknown, páginas 1–3. ix, 11

[38] Xavier, Jonas, Marina Muzzi, Edilson Camargo, Rodrigo Caetano e Fernando Matos: *Estudo da evolução da telefonia móvel no brasil*. X Encontro de Iniciação Científica e VI Encontro de Pós-Graduação–Universidade do Vale do Paraíba. São José dos Campos-SP, 2006. 6