MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

ACOPLAMENTO DE APLICATIVO EM DISPOSITIVOS MÓVEIS PARA VISTORIA DE CAMPO COM SISTEMAS DE MONITORAMENTO E ALERTA A DESASTRES NATURAIS

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA
(PIBIC/CNPq/INPE)

Junio Luiz Sendreto dos Santos (FATEC / SJC, Bolsista PIBIC/CNPq)
E-mail: junio.sendreto@hotmail.com

Eymar Silva Sampaio Lopes (DPI/OBT/INPE, Orientador)
E-mail: eymar@dpi.inpe.br

COLABORADORES

Dr. Edzon Eduardo Basseto Junior (FATEC/SJC)

Junho de 2016

AGRADECIMENTOS

Eu agradeço a Deus, minha família, meu orientador Dr. Eymar Silva Sampaio Lopes e ao CNPq por ter me proporcionado a bolsa de iniciação científica e assim realizar esse projeto.

RESUMO

A Defesa Civil foi criada no Brasil após acontecimentos que geraram grandes estragos, como fortes chuvas que ocorreram em Caraguatatuba (1967), onde muitas vidas foram perdidas pela falta de órgãos públicos e composição com comunidades, que trabalhassem de forma mais rápida. Com todos esses acontecimentos foi visto a necessidade de ter preparação, para que seja feita a prevenção contra esses acontecimentos emergências. Os deslizamentos de terra estão entre os desastres naturais que se tornou foco da Defesa Civil por ser o fenômeno responsável pelo maior número de óbitos no Brasil. Os deslizamentos envolvem solo, rocha e/ou vegetação ao longo da vertente sob a ação direta da gravidade, onde a água da chuva é o agente deflagrador. O Instituto Geológico juntamente com o Instituto de Pesquisas Tecnológica elaborou estudos e assim criado o Plano Preventivo da Defesa Civil que se baseia em três parâmetros Índice Pluviométricos, Previsão Meteorológica e Vistorias em Campo. Atualmente essas vistorias realizadas em campo são feitas manualmente em formulário de papel. O objetivo desse trabalho é criar uma aplicação para dispositivos móveis, que controlará o fluxo de trabalho da Defesa Civil em suas vistorias, a fim de evitar ou mitigar deslizamentos de terra envolvendo pessoas e principalmente agilizar a tomada de decisão no caso da retirada de pessoas de áreas de risco, logo acabando com as vistorias feitas através de formulários em papel. Com essa aplicação o controle de dados ficará mais consistente, pois o fluxo de vistoria será armazenado em um banco de dados local no dispositivo móvel, onde o usuário terá uma conta e ao acessa-la poderá realizar vistorias.

Palavras-chave: Deslizamento. Dispositivos móveis. Defesa Civil. Android.

ABSTRACT

The Civil Defense was created in Brazil after events generated that Great damage, as heavy rains occurred in Caraguatatuba (1967), where they have lost many lives for lack of public agencies and composition with communities, that they worked faster. With all events it has seen the necessity to have preparation, for there is prevention with emergency events. Landslides are among the disaster that became focus of the Civil Defense, because it is phenomenon responsible increased number of deaths in Brazil. landslides involve soil, rock and / or vegetation Along shed hiccup a Direct Action of Gravity, Where the rain water and the triggering agent. The Geological Institute and the Institute for Technological Research and Studies created the Civil Defense Plan Preventive That is based on Tres Parameters Index Pluviometric, Weather Forecast and Surveys Field. Currently these surveys field are made manually in paper form. The aim of this work is to create a application for mobile devices, that control the civil defense Workflow in your surveys, one to avoid or mitigate landslides involving people and especially streamline decision making in the case of the people of withdrawal risk areas, therefore ending the made surveys through forms on paper. With this application the data control will be more consistent because the inspection flow will be stored in a local database on the mobile device where the user has an account and access it can carry out surveys.

Keywords: Landslide. Mobile devices. Civil defense. Android..

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS E ABREVIATURAS

IG	Instituto Geológico
IPT	Instituto de Pesquisas Tecnológicas
PPDC	Plano Preventivo da Defesa Civil
IDE	Integrated Development Environment
JDK	Java SE Development Kit
ORMLite	Object Relational Mapping
DDL	Data Definition Language
XML	Extensible Markup Language

SUMÁRIO

Conteúdo

1 INTRODUÇÃO

A Defesa Civil no Brasil (Defesa Civil, 2016) foi criada após acontecimentos que geraram grandes danos às comunidades, como fortes chuvas que ocorreram em Caraguatatuba (1967), incêndio nos edifícios Andraus (1972) e Joelma (1874), onde muitas vidas foram perdidas pela falta de órgãos públicos e composição com comunidades, que trabalhassem de forma mais rápida. Com todos esses acontecimentos foi visto a necessidade de ter preparação, para que seja feita a prevenção contra esses acontecimentos emergências.

Movimentos gravitacionais de massa como os deslizamentos de terra ocorrem continuamente em diferentes escalas de tempo e espaço, como parte da evolução natural das encostas do relevo da Serra do Mar (Guidicini & Nieble, 1984, IPT, 1988; Wolle, 1988; Wolle & Carvalho, 1989; Fernandes & Amaral, 2003), pois sob condições de clima tropical em relevo escarpado, como é o litoral paulista, tais movimentos ocorrem de modo localizado e generalizado, geralmente decorrentes de eventos pluviométricos intensos. Segundo Cerri (2001), tal tipo de movimento está dentre os riscos ambientais naturais relacionados ao meio físico, geológico e de origem exógena.

Os deslizamentos de terra estão entre os desastres naturais que se tornou foco da Defesa Civil (Tominaga, Santoro, & Amaral, 2009) por ser o fenômeno responsável pelo maior número de óbitos no Brasil. Os deslizamentos envolvem solo, rocha e/ou vegetação ao longo da vertente sob a ação direta da gravidade, onde a água da chuva é o agente deflagrador. A figura 1 mostra a quantidade óbitos decorrente dos deslizamentos nos últimos anos.

Em 1988, o governo do estado de São Paulo ficou inquieto com a série de deslizamentos, em Petrópolis com 171 mortes, Rio de Janeiro 53 mortes e no litoral de São Paulo com 17 (Cubatão, Santos e Ubatuba). Com isso o governo determinou que o IG (Instituto Geológico) e IPT (Instituto de Pesquisas Tecnológicas) fizessem estudos, que produziram mapeamento dos problemas e

proposta de solução. Um dos resultados desses estudos foi a criação do Plano Preventivo da Defesa Civil - PPDC, (Macedo, Santoro, & Araújo).

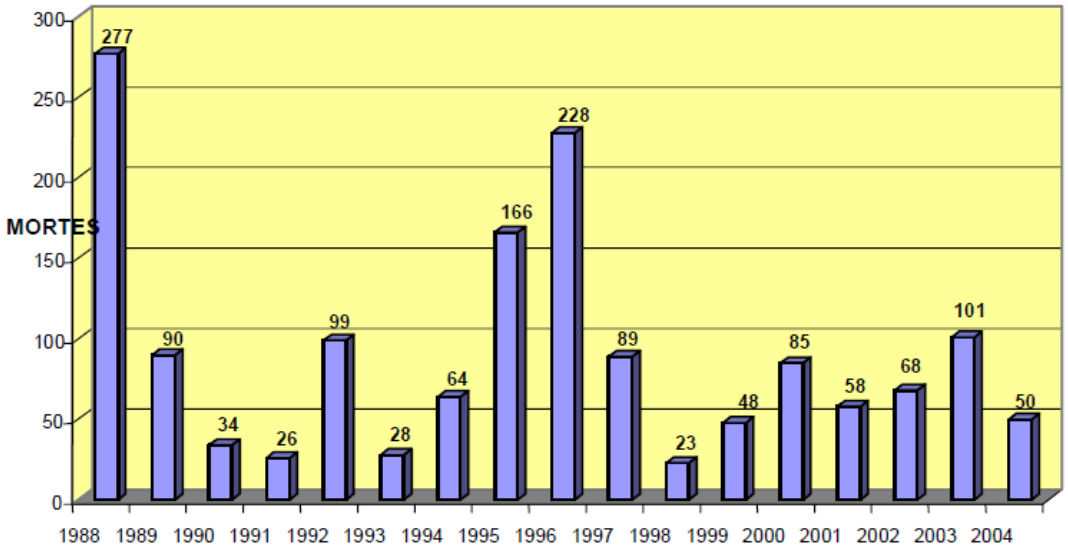

Figura 12 - Gráfico com o número de mortos no Brasil vítimas de escorregamentos (deslizamentos), a partir de notícias de jornal (Fonte: PT, 2004).

O PPDC se fundamenta em formas antecipadas em relação a ação de movimento em massa, assim ela dispõe de 3 parâmetros: Índice Pluviométricos, Previsão Meteorológica e Vistorias em Campo. Esse plano se estrutura em 4 níveis (Observação, Atenção, Alerta e Alerta Máximo), onde ele nos constata uma probabilidade maior ou menor de ocorrer escorregamentos (Macedo, Santoro, & Araújo).

Cada nível tem a sua tem sua ação a ser tomada, uns dos níveis em destaque é Atenção, pois esse estágio indica que o solo atingiu o limite de milímetros que foi estudado pelo IG/IPT (Macedo, Santoro, & Araújo), assim uma equipe da Defesa Civil vai até o local em atenção e realiza uma vistoria, porém essa vistoria é feita manualmente, ou seja, o técnico da Defesa Civil vai até o local com uma ficha em papel e a preenche com as informações do local. Após o término dessa vistoria o técnico analisa e com seu conhecimento dita se aumenta, deixa estável ou diminui o grau de risco da área.

Por outro lado, os serviços Web Geoespaciais têm sido utilizados para facilitar o acesso a dados espaciais, porém a utilização desse meio não é mais voltada para aplicações desenvolvidas para computadores pessoais. Os dispositivos móveis têm sido cada vez mais empregados pelo seu baixo custo, e uma vez que muitos usuários têm acesso a cada vez mais a uma grande quantidade de aplicativos que permitem o compartilhamento de conteúdo, realização de análises e consultas em tempo real, segundo uma pesquisa feita pela Fundação Getulio Vargas de São Paulo, se um usuário tiver uma quantidade de dinheiro para comprar ou um smartphone ou um computador, a chance dele comprar o smartphone é bem maior.

Com o advento da possibilidade de substituição de processos manuais por informatizados, é interessante para o cenário atual de vistorias em campos sobre movimentos em massa, utilizar o informatizado em detrimento do manual, ainda que esse modo manual permitam os trabalhos, um aplicativo para dispositivos móveis traria robustez e facilidade para o processo, alguns dos quais citados abaixo:

- Problemas de inconsistência de dados ou falta de dados – Em formulário o usuário pode fazer o que quiser, ou seja, se ele quiser deixar algum campo sem preencher não há problemas, mas através da aplicação restrições podem ser colocadas para determinados atributos que são de grande importância para coleta, logo não deixando finalizar a vistoria se os dados não estiverem presentes.

- Problemas de rasuras ou inserção de dados – Todo formulário depende da parte escrita ou o check do autor, assim a chances de erros são grandes ou como há partes de checklist pode ocorrer rasuras por inserção equivocada da avaliação. Com o formulário em aplicativo ele dispõe de recursos que o usuário mesmo feito pode ser alterado quantas vezes for desejado.

2 PREPARAÇÃO DO TRABALHO

2.1. OBJETIVO

O objetivo geral desse trabalho é criar uma aplicação feita para dispositivos móveis, que controlará o fluxo de trabalho da Defesa Civil em suas vistorias, a fim de evitar ou mitigar deslizamentos de terra envolvendo pessoas.

2.2. OBJETIVOS ESPECÍFICOS

Para a consecução deste trabalho foram estabelecidos os objetivos específicos:

- Registrar o local da vistoria através da aquisição de uma localização por GPS;
- Registrar os dados de usuário de uma vistoria;
- Registrar os dados de vistorias em diferentes datas;
- Visualizar vistorias realizadas e realizar atualizações se necessário.

2.3. ABORDAGEM METODOLÓGICA

A preparação do trabalho realizada neste trabalho seguiu as seguintes etapas, conforme apresentadas na Figura 2.

1 – Levantamento do Problema de Vistorias pelo PPDC: Para a compreensão desse problema, foi feito revisão da literatura, reunião técnica com IG e participação de curso “Oficina preparatória para operação verão” – com a intenção de compreender todas as etapas de preenchimento do formulário em papel da vistoria.

2 – Definição e Criação do Modelo Lógico: Com base na necessidade do projeto apresentado na reunião e no curso de verão. Com base em todo o procedimento anterior, foi feito uma reunião sobre como seria projetado o banco de dados dessa aplicação, assim foi gerado um modelo lógico para especificar melhor o banco de dados e facilitar no momento da codificação.

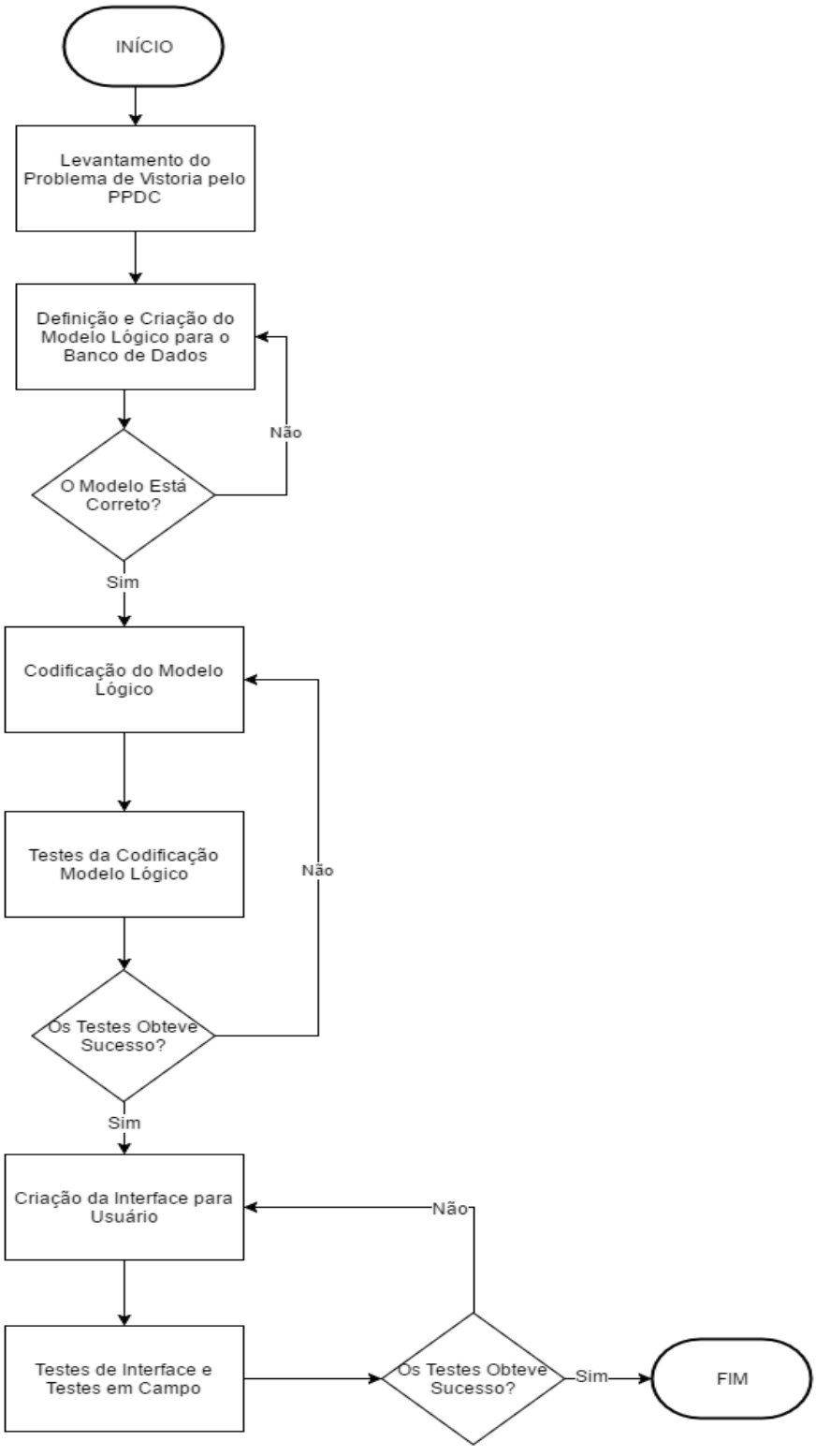

Figura 13 - Fluxograma da preparação do trabalho.

3 – Codificação do Modelo Lógico: A codificação do modelo lógico foi realizada na linguagem de programação JAVA, juntamente com SQLite e o framework ORMlite.

4 – Testes da Codificação do Modelo Lógico: Para que não houvesse problemas na persistência dados foram realizados testes de unidade nos métodos que trabalham com a persistência de dados no banco de dados.

5 – Criação da Interface para Usuário: A criação da interface para o usuário foi feita por XML (Extensible Markup Language).

6 – Testes de Interfaces e Testes em Campo: Os testes foram realizados por intermédio do simulador que o Android Studio dispõe e por dispositivos móveis, para ter uma realidade melhor de como seria feito a vistoria com a aplicação.

3 DESENVOLVIMENTO

Nesse capítulo será mostrado a ferramentas utilizadas e como foi feito o desenvolvimento.

3.1. FERRAMENTAS

Ubuntu: Sistema Operacional, onde foi preparado todo o ambiente para desenvolvimento da aplicação, versão 14.04LTS.

Java Devepment Kit: Conjunto de ferramenta para desenvolvimento com a linguagem de programação Java, versão 1.7

Android: Sistema Operacional para executar a aplicação.

Android Studio: IDE (Integrated Development Environment), versão 2.1.2

SQLite: Banco de dados da aplicação, versão 3.13

ORMLite: O ORMLite (Object Relational Mapping) foi utilizado para persistência de objetos feitos em Java para banco de dados, versão 4.48

Junit: Para validação de persistência de dados no banco teste unitário foram realizados através do Junit, versão 14.12.

3.1. LEVANTAMENTO DE REQUISITOS

3.1.1. CASOS DE USO

Os casos de usos estão divididos em criação de usuário, acesso a aplicação através do cadastro feito, criação de vistorias, salvar vistorias, listar vistorias e recuperar dados de qualquer vistoria realizada.

3.2.2. DIAGRAMA DE CASOS DE USO

A figura 3 a seguir apresenta o diagrama de casos de uso do sistema.

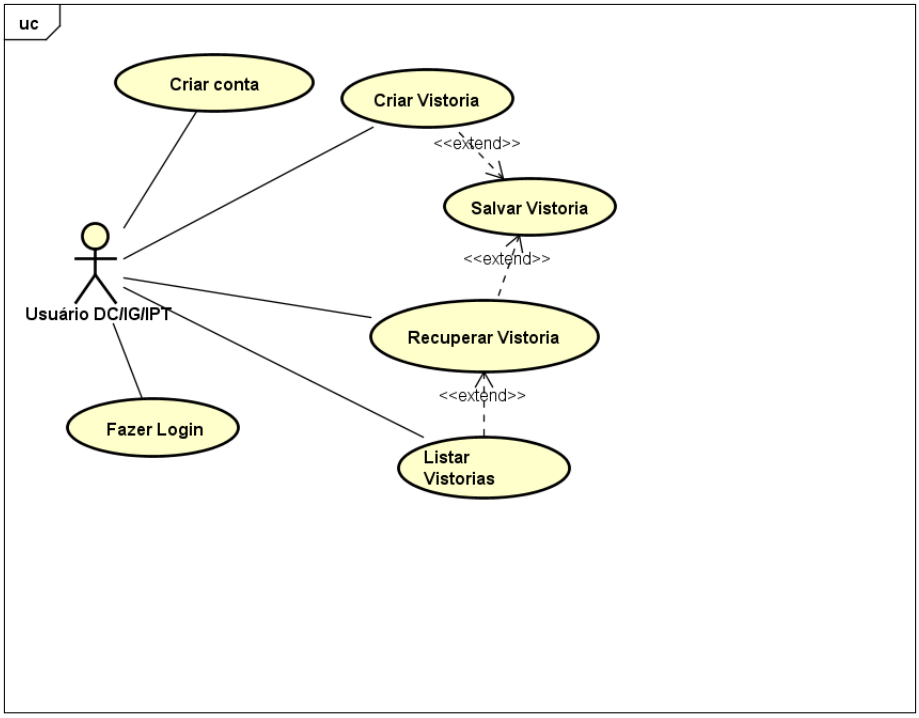

Figura 14 - Diagrama de casos de uso

3.2.3. DOCUMENTAÇÃO DOS CASOS DE USO

A seguir é apresentado os quadros de 1 a 6 da documentação do diagrama de casos de uso.

Quadro 13 - Caso de Uso: Criar Conta

Nome do Caso de Uso	Criar Conta
Ator Principal	Usuário DC/IG/IPT
Resumo	Esse caso de uso descreve as fases realizadas pelo usuário em seu cadastro.
Pré-Condições	
Fluxo Principal	
Ações do autor	Ações do sistema
1 – Acessar a tela de login.	5 – Validar campos digitados pelo usuário
2 – Clicar no ícone que referencia a	6 – Salvar o usuário no banco de

tela de cadastro.	dados.
3 – Informar dados pessoais.	
4 – Clicar em salvar.	
Restrições/Validações	É necessário que o usuário forneça seu nome, um login único, senha com no mínimo 5 dígitos, confirmar sua senha, digitar CPF corretamente e e-mail.
Fluxo de Exceção – Informações incorretas	
Ações do ator	Ações do sistema
	1 – Mostrar mensagem informando que há campos obrigatórios ou inválidos.

Quadro 14 - Caso de Uso: Fazer Login

Nome do Caso de Uso	Fazer Login
Ator Principal	Usuário DC/IG/IPT
Resumo	Esse caso de uso descreve as fases realizadas pelo usuário para utilizar os recursos da aplicação.
Pré-Condições	Ter feito a criação de usuário
Fluxo Principal	
Ações do autor	Ações do sistema
1 – Informar login e senha.	3 – Verificar se existe o usuário no banco de dados.
2 – Clicar em entrar.	4 – Levar o usuário para tela inicial.
Restrições/Validações	1 – O usuário deve fornecer login e senha corretamente.
Fluxo de Exceção – Informações incorretas	
Ações do ator	Ações do sistema
	1 – Mostrar mensagem informando que o login ou senha estão incorretos.

Quadro 15 - Caso de Uso: Criar Vistoria

Nome do Caso de Uso	Criar Vistoria
Ator Principal	Usuário DC/IG/IPT
Resumo	Esse caso de uso descreve as fases realizadas pelo usuário para criar uma vistoria.
Pré-Condições	Ter feito login.
Fluxo Principal	
Ações do autor	Ações do sistema
1 – Informar latitude e longitude.	3 – Levar o usuário para tela de vistoria
2 – Clicar em criar nova vistoria.	
Restrições/Validações	1 – O usuário deve fornecer latitude e longitude.
Fluxo de Exceção – Informações incorretas	
Ações do ator	Ações do sistema
	1 – Mostrar mensagem informando que o usuário precisa informa a latitude e longitude.

Quadro 16 - Caso de Uso: Salvar Vistoria

Nome do Caso de Uso	Salvar Vistoria
Ator Principal	Usuário DC/IG/IPT
Resumo	Esse caso de uso descreve as fases realizadas pelo usuário para salvar uma vistoria.
Pré-Condições	Ter feito login e informado latitude e longitude corretamente.
Restrições/Validações	O usuário pode ou não salvar a vistoria.
Fluxo Principal	
Ações do autor	Ações do sistema
1 – Informar como se encontra o local de acordo com o formulário.	5 – Verificar se existe o usuário no banco de dados.
2 – Clicar em salvar vistoria.	6 – Levar o usuário para tela inicial, após a confirmação do armazenamento da vistoria.
3 – Cancelar o armazenamento da vistoria	7 – Salvar a vistoria no banco de dados de acordo com o relacionamento entre as tabelas.
4 – Confirmar o armazenamento da vistoria	

Quadro 17 - Caso de Uso: Listar Vistorias

Nome do Caso de Uso	Listar Vistorias
Ator Principal	Usuário DC/IG/IPT
Resumo	Esse caso de uso descreve as fases realizadas pelo usuário para listar as vistorias realizadas.
Pré-Condições	Ter feito login.
Fluxo Principal	
Ações do autor	Ações do sistema
1 – Clicar em listar vistorias.	2 – Levar o usuário para tela de lista de vistorias.
	2 – Buscar todas as vistorias realizada no banco de dado.
	3 – Listar todas as vistorias buscadas no banco de dado, informando somente o autor, data, bairro município.

Quadro 18 - Caso de Uso: Recuperar Vistoria

Nome do Caso de Uso	Recuperar Vistoria
Ator Principal	Usuário DC/IG/IPT
Resumo	Esse caso de uso descreve as fases realizadas pelo usuário para recuperar vistorias.
Pré-Condições	Ter feito login.
Restrições/Validações	O usuário pode ou não recuperar uma vistoria.
Fluxo Principal	
Ações do autor	Ações do sistema
1 – Estar na tela inicial	5 – Levar usuário para tela de lista de vistorias.
2 – Clicar em lista vistorias.	6 – Buscar no banco de dados a vistoria selecionada pelo usuário.
3 – Fazer longo clique em cima das informações da vistoria que está na lista.	7 – Levar o usuário para tela de vistoria com todos os valores colocados no formulário da vistoria escolhida pelo usuário.

3.2. REQUISITOS

3.3.1. REQUISITOS FUNCIONAIS

A seguir é apresentado o quadro 7 dos requisitos funcionais do sistema.

Quadro 19 - Requisitos Funcionais

Requisito 1	O sistema possuir a funcionalidade de cadastro de usuário.
Requisito 2	O sistema deverá ter autenticação de usuário para permitir o acesso as demais funcionalidades.
Requisito 3	O sistema deverá permitir que o usuário crie vistorias.
Requisito 4	O sistema deverá a permitir busca no banco de dados de todas as vistorias realizadas.
Requisito 5	O sistema deverá manter a sessão do usuário durante todo o tempo de execução da aplicação.
Requisito 6	O sistema deverá permitir ao usuário o aproveitamento de dados das vistorias já feitas para criação de novas vistorias.
Requisito 7	O sistema deverá possuir a funcionalidade de cancelamento do armazenamento da vistoria.
Requisito 8	A aplicação deve ser feita para dispositivos móveis.
Requisito 9	O sistema deve apresentar um formulário digital semelhante ao formulário de papel.

3.3.2 REQUISITOS NÃO FUNCIONAIS

A seguir é apresentado o quadro 8 dos requisitos não funcionais do sistema.

Quadro 20 - Requisitos não funcionais

Requisito 1	O sistema deverá validar todos os campos de cadastro.
Requisito 2	O sistema deverá apresentar ao usuário uma notificação se o cadastro ocorreu com sucesso.
Requisito 3	O sistema deverá apresentar uma notificação após o usuário clicar em salvar vistoria.

3.4. MODELO DE DADOS

3.4.1 MODELO LÓGICO

A figura 4 apresenta o modelo lógico que foi criado para o sistema.

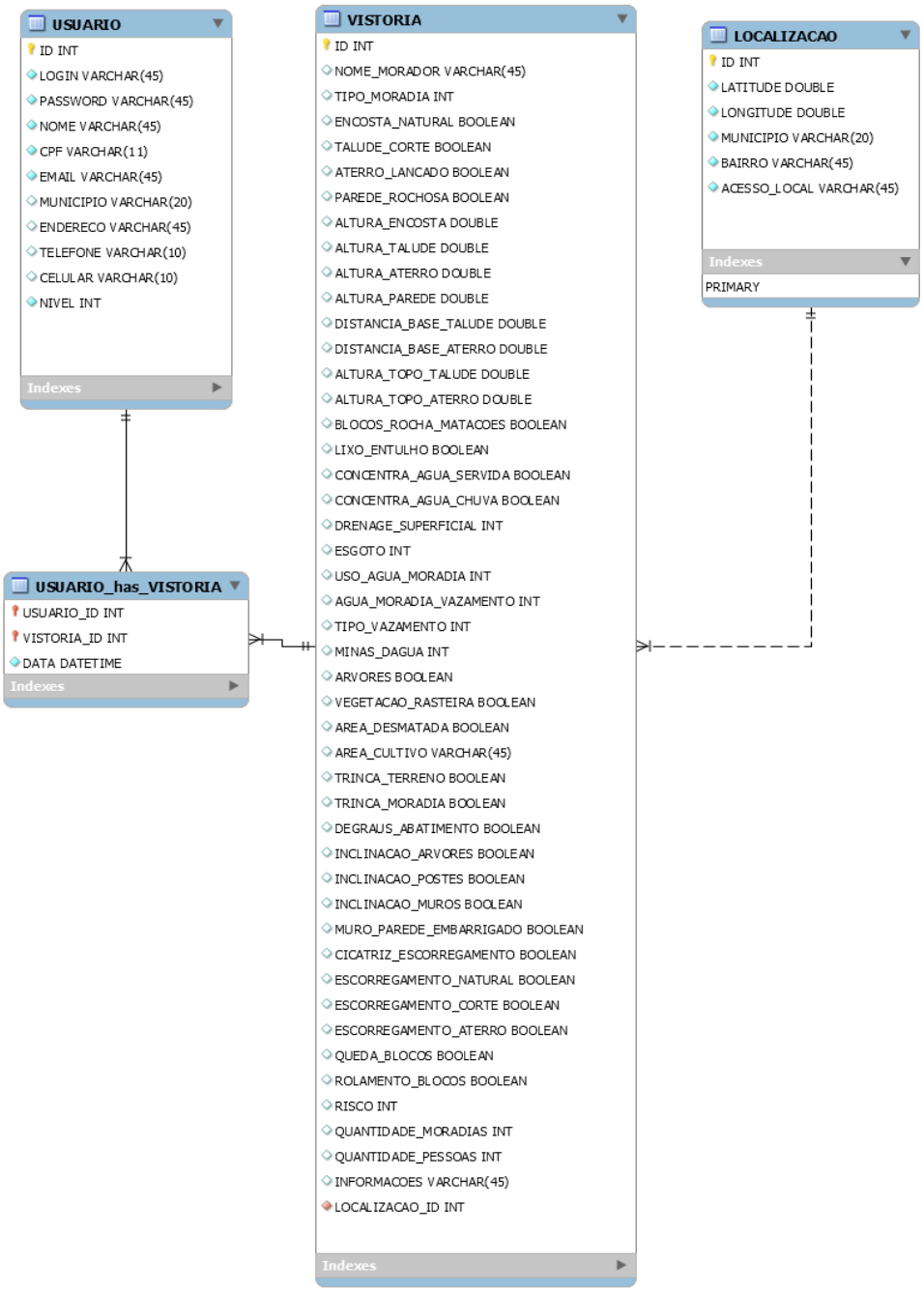

Figura 15 - Modelo Lógico do sistema

3.4.2. DICIONÁRIO DE DADOS

A seguir é apresentado as tabelas de 9 a 12 do dicionário de dados.

Quadro 21 - Dicionário de Dados da tabela "usuario".

Entidade: usuário				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
ID	Determinante	Numérico		Chave primária
NOME	Simples	Texto	45	Nome do usuário
LOGIN	Determinante	Texto	45	Apelido do usuário para acesso a aplicação
PASSWORD	Determinante	Texto	45	Senha do usuário para acesso a aplicação
CPF	Determinante	Texto	11	Cadastro de Pessoas Físicas
EMAIL	Simples	Texto	45	
MUNICÍPIO	Simples	Texto	20	Cidade em que a usuário mora
ENDERECO	Composto	Texto	45	Local onde o usuário mora
TELEFONE	Simples	Texto	10	
CELULAR	Simples	Texto	10	
NIVEL	Simples	Numérico		Identificação de tipo de usuário

Quadro 22 - Dicionário de Dados da tabela "usuario_vistoria".

Entidade: usuario_vistoria				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
ID	Determinante	Numérico		Chave primária
USUARIO_ID	Determinante	Numérico		Chave estrangeira da entidade USUARIO
VISTORIA_ID	Determinante	Numérico		Chave estrangeira da entidade Vistoria
DATA	Composto	Data		Data do registro

Quadro 23 - Dicionário de Dados da tabela "vistoria".

Entidade: vistoria				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
ID	Determinante	Numérico		Chave primária
LOCALIZACAO_ID	Determinante	Numérico		Chave Estrangeira da entidade Localização

Entidade: vistoria				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
NOME_MORADOR	Simples	Texto		Nome do morador do local vistoriado
TIPO_MORADIA	Simples	Texto	45	
ENCOSTA_NATURAL	Composto	Booleano		
TALUDE_CORTE	Composto	Booleano		
ATERRO_LANCADO	Composto	Booleano		
PAREDE_ROCHOSA	Composto	Booleano		
ALTURA_ENCOSTA	Simples	Numérico		Altura Da Encosta Natural
ALTURA_TALUDE	Simples	Numérico		Altura do Talude de Corte
ALTURA_ATERRO	Simples	Numérico		Altura do Aterro Lançado
ALTURA_ATERRO	Simples	Numérico		Altura de Presença de Parede Rochosa
DISTANCIA_BASE_TALUDE	Simples	Numérico		Distância das moradias do Talude de Corte
DISTANCIA_BASE_ATERRO	Simples	Numérico		Distância das moradias do Aterro Lançado
ALTURA_TOPO_TALUDE	Simples	Numérico		Topo do Talude de Corte
ALTURA_TOPO_ATERRO	Simples	Numérico		Topo do Aterro Lançado
BLOCOS_ROCHA_MATACOES	Simples	Booleano		Presença de blocos de rocha e matacões
LIXO_ENTULHO	Simples	Booleano		Presença de Lixo ou Entulho
CONCENTRA_AGUA_CHUVA	Simples	Booleano		Concentração de água de chuva em superfície (enxurrada)
LANCAMENTO_AGUA_SERVIDA	Simples	Booleano		Lançamento de água servida em superfície (a céu aberto ou no quintal)
DRENAGE_SUPERFICIAL	Simples	Numérico		Sistema de drenagem superficial
ESGOTO	Simples	Numérico		Local que o esgoto vai
USO_AGUA_MORADI	Simples	Numérico		Local que fornece

Entidade: vistoria				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
A				água para moradia
TIPO_VAZAMENTO	Simples	Numérico		Presença de vazamento na tubulação
MINAS_DAGUA	Simples	Numérico		Minas d'água no talude
ARVORES	Simples	Booleano		Presença de árvores
VEGETACAO_RASTEIRA	Simples	Booleano		Presença de vegetação Rasteira
AREA_DESMATADA	Simples	Booleano		Presença de área desmatada
AREA_CULTIVO	Simples	Texto	20	Presença de área de cultivo
TRINCA_TERRENO	Simples	Booleano		Presença de trincas no terreno
TRINCA_MORADIA	Simples	Booleano		Presença de trincas na moradia
DEGRAUS_ABATIMENTO	Simples	Booleano		Presença de Degraus de abatimento
INCLINACAO_ARVORES	Simples	Booleano		Presença de inclinação de árvores
INCLINACAO_POSTES	Simples	Booleano		Presença de inclinação de postes
INCLINACAO_MUROS	Simples	Booleano		Presença de inclinação de muros
MURO_PAREDE_EMBARRIGADO	Simples	Booleano		Muros ou parede embarrigados
CICATRIZ_ESCORREGAMENTO	Simples	Booleano		Presença de Escorregamento próximo à moradia
ESCORREGAMENTO_NATURAL	Simples	Booleano		Presença de escorregamento no talude de natural
ESCORREGAMENTO_CORTEA	Simples	Booleano		Presença de escorregamento no talude de corte
ESCORREGAMENTO_ATERRO	Simples	Booleano		Presença de escorregamento no aterro
QUEDA_BLOCOS	Simples	Booleano		
ROLAMENTO_BLOC	Simples	Booleano		

Entidade: vistoria				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
OS				
RISCO	Simples	Numérico		Validação do risco que se encontra a moradia
QUANTIDADE_MORADIAS	Simples	Numérico		Quantidade de moradia que estão em risco
QUANTIDADE_PESSOAS	Simples	Numérico		Quantidade de pessoas que deverão ser removidas
INFORMACOES	Simples	Texto	45	Informações adicionais sobre o local

Quadro 24 - Dicionário de Dados da tabela "localizacao".

Entidade: localização				
Atributo	**Classe**	**Domínio**	**Tamanho**	**Descrição**
ID	Determinante	Numérico		
LATITUDE	Simples	Numérico		
LONGITUDE	Simples	Numérico		
MUNICIPIO	Simples	Texto	20	
BAIRRO	Simples	Texto	45	
ACESSO_LOCAL	Simples	Texto	45	Condições de acesso ao local

3.5. Arquitetura

Para realização desse trabalho projeto foi utilizado o padrão de projeto MVC (Model, View, Controler).

A figura 5 apresenta a estrutura do sistema desenvolvido na aplicação.

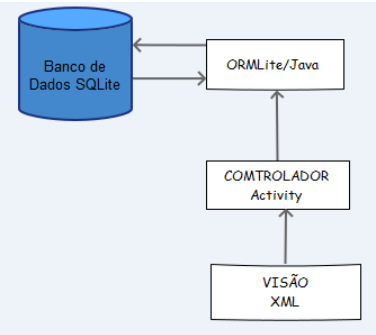

Figura 5 - Estrutura do Sistema

3.5.2. MAPEAMENTO DO BANCO DE DADOS

Trabalhando com aplicativos para android há maneiras de criar tabelas no banco:

- Forma Padrão: Esse modo é feito por codificação DDL (Data Definition Language) puro, como segue a Figura 6. Dessa maneira todo comando é feito em String em Java e na classe feita para criação de banco de dados essa String é passada por um método e assim é criado as tabelas.

```
01 String CREATE_USUARIO_TABLE = "CREATE TABLE " +
02 Usuario.TABLE_NAME + "(" +
03          Usuario.COL_ID + " INTEGER PRIMARY KEY
04 AUTOINCREMENT, " +
05          Usuario.COL_NOME + " TEXT NOT NULL, " +
06          Usuario.COL_CPF + " TEXT NOT NULL, " +
07          Usuario.COL_LOGIN + " TEXT NOT NULL," +
09          Usuario.COL_PASSWORD + " TEXT NOT NULL, " +
10          Usuario.COL_EMAIL + " TEXT NOT NULL, " +
11          Usuario.COL_MUNICIPIO + " TEXT, " +
12          Usuario.COL_ENDERECO + " TEXT, " +
13          Usuario.COL_TELEFONE + " TEXT, " +
14          Usuario.COL_CELULAR + " TEXT, " +
15          Usuario.COL_NIVEL + " INTEGER NOT NULL " + ");";
16 db.execSQL(CREATE_USUARIO_TABLE);
17
```

Figura 6 - Forma padrão do DDL com Java

- Framework ORMLite: Esse modo dispõe de notações para classes e seus atributos, onde essas notações referenciam eles como tabelas ou colunas no banco. Por intermédio dessas notações é substituído e toda codificação DDL, como segue Figura 7.

```
01 @DatabaseTable (tableName = "usuario")
02 public class Usuario{
03          @DatabaseField (columnName = "id", generatedId =
04 true)
    private Long id;
05
    @DatabaseField (columnName = "nome", canBeNull = false)
06  private String nome;
07
    @DatabaseField (columnName = "login", canBeNull = false)
09  private String login;
10
    @DatabaseField (columnName = "password", canBeNull =
11 false)
12  private String password;
13
    @DatabaseField (columnName = "cpf", canBeNull = false)
14  private String cpf;
15
    @DatabaseField (columnName = "email", canBeNull = false)
16  private String email;
17
    @DatabaseField (columnName = "municipio", canBeNull =
18 true)
19  private String municipio;
20  @DatabaseField (columnName = "endereco", canBeNull = true)
21  private String endereco;
22  @DatabaseField (columnName = "telefone", canBeNull = true)
23  private String telefone;
24  @DatabaseField (columnName = "celular", canBeNull = true)
24  private String celular;
25  @DatabaseField (columnName = "nivel", canBeNull = false)
26  private Integer nivel;
}
```

Figura 7 - DDL criada pelo framework ORMLite

As duas formas de mapeamento foram realizadas, porém através do framework ORMLite foi visto uma grande vantagem de se trabalhar, pois não necessita a utilização de DDL puro e há melhorias como segue no trabalho.

3.5.3. PERSISTÊNCIA DE DADOS

Na consecução dos métodos de persistência de dados no banco de dados SQLite o framework ORMLite dispõe de alguns métodos CRUD (Create, Read, Update e Delete) genéricos que facilita o reaproveitamento de código.

Com isso uma classe com implementação desses métodos foi feita (AbstractDaoImp.class), Figura 8. Cada método possui sua necessidade para que chegue ao objetivo, que em casos precisa da classe ou do objeto da classe.

```
01 public abstract class AbstractDaoImpl {
02     protected DataBase dataBase;
03     private Context context;
04 
05     public AbstractDaoImpl(Context context) {
06         this.context = context;
07     }
09 
10     public void openBD() {
11         try {
12             if (dataBase == null || !(dataBase.isOpen())) {
13                 dataBase = new DataBase(context);
14             }
15         }catch (Exception e){
16             Log.d("--------", "ERRO OpenBD(ABSTRACTDAO) " +
17 e.getMessage());
18         }
19     }
20 
21     public Object save(Class classe, Object object) {
22         try {
23             openBD();
```

```
24              dataBase.getDao(classe).create(object);
24              return object;
25          } catch (Exception e) {
26              Log.d("--------",  "ERRO  SAVE(ABSTRACTDAO)  "  +
27  e.getMessage());
28          } finally {
29              dataBase.close();
30          }
31          return null;
32      }
33
34      public Boolean update(Class classe, Object object) {
35          try {
36              openBD();
37              dataBase.getDao(classe).update(object);
38              return true;
39          } catch (SQLException e) {
40              Log.d("------", "ERRO UPDATE(ABSDAO)");
41          } finally {
42              dataBase.close();
43          }
44          return false;
45      }
46
47      public Boolean delete(Class classe, Object object) {
48          Integer validarDelete = 0;
49          try {
50              openBD();
51              validarDelete                                    =
52  dataBase.getDao(classe).delete(object);
53          } catch (java.sql.SQLException e) {
54              e.printStackTrace();
55              Log.d("------", "ERRO DELETE(ABSDAO)");
56          } finally {
57              dataBase.close();
58          }
59          if(validarDelete == 1){
```

```
60              return true;
61          }else {
62              return false;
63          }
64      }
65      public List<Class> listAll(Class classe) {
66          List<Class> lista = null;
67          try {
68              openBD();
69              lista =  dataBase.getDao(classe).queryForAll();
70              return lista;
71          } catch (SQLException e) {
72              Log.d("------", "ERRO listAll(ABSDAO)");
73          } finally {
74              dataBase.close();
75          }
76          return null;
77      }
78      public Object findById(Class classe, Long id) {
79          Object classeId = null;
80          try {
81              openBD();
82              Dao<Class, Integer> dao = dataBase.getDao(classe);
83              classeId                                              =
84  dao.queryForId(Integer.valueOf(id.toString()));
85              return classeId;
86          } catch (SQLException e) {
87              Log.d("------", "ERRO findById(ABSDAO)");
88          }finally {
89              dataBase.close();
90          }
91          return null;
92      }
93  }
94
```

Figura 8 - Classe AbstractDaoImpl

Em análise através do reaproveitamento de código pode-se observar a Figura 8 é feito herança da classe abstrata (Figura 7). Essa forma de trabalhar faz com que a classe abstrata seja útil para qualquer outra classe que se torna tabela no banco de dados pelo mapeamento.

Ressaltado a figura 9, os dois métodos que foram gerados nela, são auxiliares em validação de usuário, quando o método (findByLoginAndPassword) faz uma querry no banco de dados em busca de um usuário de acordo com o login e o password fornecido, para realizar o acesso do usuário as demais funcionalidades da aplicação, enquanto que o método (findByLogin) faz uma querry no banco de dados em busca de logins semelhantes aquele texto informado na hora do cadastro de usuário, assim não deixando que dois usuários tenham “logins” idênticos.

```
01 public class UsuarioDaoImpl extends AbstractDaoImpl{
02     public UsuarioDaoImpl(Context context) {
03         super(context);
04     }
05     public Usuario findByLoginAndPassword(String login, String
06 password){
07         List<Usuario> usuarioList = null;
09         try {
10             openBD();
11             QueryBuilder<Usuario,   Object>   queryBuilder   =
12 (QueryBuilder<Usuario,                                Object>)
13 dataBase.getDao(Usuario.class).queryBuilder();
14             Where<Usuario,        Object>        where        =
15 queryBuilder.where();
16             where.eq(Usuario.COL_LOGIN,
17 login).and().eq(Usuario.COL_PASSWORD, password);
18             PreparedQuery<Usuario>       preparedQuery       =
19 queryBuilder.prepare();
20             usuarioList                                      =
21 dataBase.getDao(Usuario.class).query(preparedQuery);
22         } catch (java.sql.SQLException e) {
```

```
23              e.printStackTrace();
24              Log.d("----------",     "Problema     com     método
24  findByLoginAndPassword");
25          }
26          if(usuarioList.isEmpty() == true){
27              return null;
28          }else{
29              return usuarioList.get(0);
30          }
31      }
32      public    Usuario    findByLogin(String    login)    throws
33  java.sql.SQLException {
34          List<Usuario> usuarioList;
35          openBD();
36          QueryBuilder<Usuario,     Object>     queryBuilder     =
37  (QueryBuilder<Usuario,                                  Object>)
38  dataBase.getDao(Usuario.class).queryBuilder();
39          Where<Usuario, Object> where =  queryBuilder.where();
40          where.eq(Usuario.COL_LOGIN, login);
41          PreparedQuery<Usuario>          preparedQuery          =
42  queryBuilder.prepare();
43          usuarioList                                             =
44  dataBase.getDao(Usuario.class).query(preparedQuery);
45          if(usuarioList.isEmpty() == true){
46              return null;
47          }else{
48              return usuarioList.get(0);
49          }
50      }
51  }
```

Figura 9 - Classe UsuarioDaiImpl

Caso não houvesse a utilização do ORMLite, cada implementação do CRUD deveria ter sua especificação com cada persistência das tabelas. A Figura 10 nos mostra um pedaço do código realizado da classe (UsuarioDaoImpl.class). Sem a utilização desse framework todo esse código deveria ser feito para

persistência de dados de cada tabela no banco, logo deixando de existir a classe (AbstractDaoImp.class) e a herança.

```
01 public class UsuarioDaoImpl implements UsuarioDao{
02     private SQLiteDatabase db;
03     private DataBase banco;
04     private Cursor cursor;
05     private ContentValues values;
06     private Usuario usuario;
07 
08     public UsuarioDaoImpl(Context context){
09         banco = new DataBase(context);
10     }
11 
12     @Override
13     public Usuario save(Usuario usuario) {
14         try {
15             db = banco.getWritableDatabase();
16             if(db != null){
17                 ContentValues values =  new ContentValues();
18                 values.put(usuario.COL_ID, usuario.getId());
19                 values.put(usuario.COL_NOME,
20 usuario.getNome());
21                 values.put(usuario.COL_CPF, usuario.getCpf());
22                 values.put(usuario.COL_LOGIN,
23 usuario.getLogin());
24                 values.put(usuario.COL_PASSWORD,
25 usuario.getPassword());
26                 values.put(usuario.COL_EMAIL,
27 usuario.getEmail());
28                 values.put(usuario.COL_MUNICIPIO,
29 usuario.getMunicipio());
30                 values.put(usuario.COL_ENDERECO,
31 usuario.getEndereco());
32                 values.put(usuario.COL_TELEFONE,
32 usuario.getTelefone());
33                 values.put(usuario.COL_CELULAR,
```

```
34  usuario.getCelular());
35                  values.put(usuario.COL_NIVEL,
36  usuario.getNivel());
37
38                  if(db.insert(Usuario.TABLE_NAME, null, values)
39  != -1){
40                      db.close();
41                      return usuario;
42                  }
43              }
44              return null;
45
46          }catch (Exception e){
47              return null;
48          }
49      }
60  }
51
```

Figura 10 - Parte da Classe UsuarioDaoImpl sem framework ORMLite

3.6. TESTE UNITÁRIO

Para a validação dos métodos criados relacionado a persistência de dados, testes unitários da classe abstrata foram realizados pela persistência de dados do usuário, com framework JUnit Figura 11.

```
01  public class UsuarioDaoImplTest extends ConfigBDTestCase{
02     public void testSave(){
03          Long Id;
04          UsuarioDaoImpl          usuarioDao          =          new
05  UsuarioDaoImpl(getContext());
06          Usuario     actualUsuario     =     new     Usuario("Junio",
07  "00000000000",   "junio",   "12345",   "junio@hotmail.com",   "SJC",
08  "Uberaba", "11111111", "222222222", 1);
09          usuarioDao.save(Usuario.class, actualUsuario);
10          Id = actualUsuario.getId();
```

```
11          Usuario expectedUsuario = (Usuario)
12  usuarioDao.findById(Usuario.class, Id);
13          assertEquals(actualUsuario, expectedUsuario);
14          usuarioDao.delete(Usuario.class, expectedUsuario);
15
16      }
17
18      public void testUpdate() throws Exception {
19          UsuarioDaoImpl usuarioDao = new
20  UsuarioDaoImpl(getContext());
21          Usuario normalUsuario = new Usuario("Junio",
22  "00000000000", "junio", "12345", "junio@hotmail.com", "SJC",
23  "Uberaba", "11111111", "222222222", 1);
24          usuarioDao.save(Usuario.class, normalUsuario);
25          Usuario actualUsuario = (Usuario)
26  usuarioDao.findById(Usuario.class, normalUsuario.getId());
27
28          Usuario newFieldsUsuario = new Usuario("jaxson", "12345",
29  "jaxson@jaxson", "SJC", "avenida", "111222", "222111");
30
31          usuarioDao.update(Usuario.class, actualUsuario);
32          Usuario expectedUsuario = (Usuario)
32  usuarioDao.findById(Usuario.class, actualUsuario.getId());
33          assertEquals(actualUsuario, expectedUsuario);
34          usuarioDao.delete(Usuario.class, actualUsuario);
35      }
36
37      public void testDelete(){
38          Long Id;
39          UsuarioDaoImpl usuarioDao = new
40  UsuarioDaoImpl(getContext());
41          Usuario actualUsuario = new Usuario("Junio",
42  "00000000000", "junio", "12345", "junio@hotmail.com", "SJC",
43  "Uberaba", "11111111", "222222222", 1);
44          usuarioDao.save(Usuario.class, actualUsuario);
45          Id = actualUsuario.getId();
46          usuarioDao.delete(Usuario.class, actualUsuario);
```

```
47          Usuario expectedUsuario = (Usuario)
48  usuarioDao.findById(Usuario.class, Id);
49          assertEquals(true, expectedUsuario == null);
50      }
51
52      public void testListAll(){
53          UsuarioDaoImpl usuarioDao = new
54  UsuarioDaoImpl(getContext());
55          Usuario usuario1 = new Usuario("Junio", "00000000000",
56  "junio", "12345", "junio@hotmail.com", "SJC", "Uberaba",
57  "11111111", "222222222", 1);
58          Usuario usuario2 = new Usuario("Jarvan", "00000000000",
59  "Jarvan", "12345", "jarvan@hotmail.com", "SJC", "Guaraciaba",
60  "11111111", "222222222", 1);
61          Usuario usuario3 = new Usuario("Fiora", "00000000000",
62  "Fiora", "12345", "fiora@hotmail.com", "SJC", "Caparaó",
63  "11111111", "222222222", 1);
64          List<Usuario> listFixa = new ArrayList<Usuario>();
65          listFixa.add(usuario1);
66          listFixa.add(usuario2);
67          listFixa.add(usuario3);
68          usuarioDao.save(Usuario.class, usuario1);
69          usuarioDao.save(Usuario.class, usuario2);
70          usuarioDao.save(Usuario.class, usuario3);
71          List<Usuario> listBD = (ArrayList)
72  usuarioDao.listAll(Usuario.class);
73
74          for(int i = 0; i < listFixa.size(); i++){
75
76  assertEquals(listBD.get(i).getNome().equals(listFixa.get(i).getNom
77  e()), true);
78          }
79
80          usuarioDao.delete(Usuario.class, usuario1);
81          usuarioDao.delete(Usuario.class, usuario2);
82          usuarioDao.delete(Usuario.class, usuario3);
83      }
```

```
84
85      public void testFindById(){
86          UsuarioDaoImpl usuarioDao = new
87      UsuarioDaoImpl(getContext());
88          Usuario actualUsuario = new Usuario("Junio",
89      "00000000000", "junio", "12345", "junio@hotmail.com", "SJC",
90      "Uberaba", "11111111", "222222222", 1);
91          usuarioDao.save(Usuario.class, actualUsuario);
92          Usuario expectedUsuario = (Usuario)
93      usuarioDao.findById(Usuario.class, actualUsuario.getId());
94
95          assertEquals(actualUsuario.getId(),
96      expectedUsuario.getId());
97
98          usuarioDao.delete(Usuario.class, actualUsuario);
99      }
100
101     public void testFindByLoginAndPassword(){
102         UsuarioDaoImpl usuarioDao = new
103     UsuarioDaoImpl(getContext());
104         Usuario actualUsuario = new Usuario("Junio",
105     "00000000000", "junio", "12345", "junio@hotmail.com", "SJC",
106     "Uberaba", "11111111", "222222222", 1);
107         usuarioDao.save(Usuario.class, actualUsuario);
108
109         Usuario expectedUsuario =
110     usuarioDao.findByLoginAndPassword(actualUsuario.getLogin(),
111     actualUsuario.getPassword());
112
113         assertEquals(actualUsuario.getLogin(),
114     expectedUsuario.getLogin());
115         usuarioDao.delete(Usuario.class, actualUsuario);
116
117     }
118     public void testFindByLogin() throws SQLException {
119         UsuarioDaoImpl usuarioDao = new
120     UsuarioDaoImpl(getContext());
```

```
121         Usuario actualUsuario = new Usuario("Junio",
122 "00000000000", "junio", "12345", "junio@hotmail.com", "SJC",
123 "Uberaba", "11111111", "222222222", 1);
124         usuarioDao.save(Usuario.class, actualUsuario);
125
126         Usuario expectedUsuario =
127 usuarioDao.findByLogin(actualUsuario.getLogin());
128         assertEquals(actualUsuario.getLogin(),
    expectedUsuario.getLogin());
        usuarioDao.delete(Usuario.class, actualUsuario);
    }
}
```

Figura 11 - Classe de teste UsuarioDaoImplTest

A Figura 12 representa a mensagem de que o teste feito com todos os métodos do usuário foi bem-sucedido.

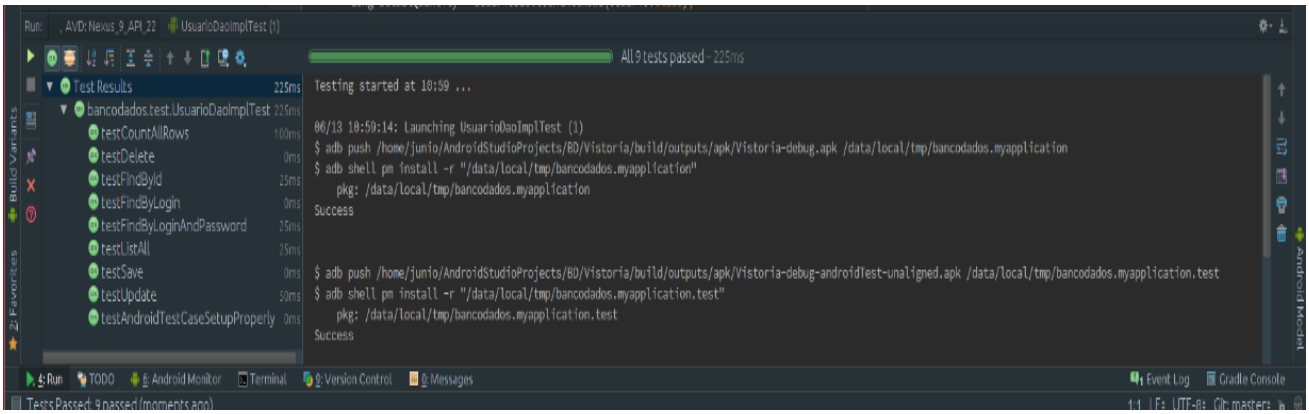

Figura 16 - Resultado do Teste Unitário da Classe UsuarioDaoImplTest

4 RESULTADOS E CONCLUSÕES

As telas que foram criadas para atender cada requisito levantado foram feitas em XML. A Quadro 13 a seguir apresenta os resultados conforme os requisitos do sistema.

Quadro 13 – Solução proposta para cada requisito levantado

Requisito 1	Para atender essa funcionalidade uma tela de cadastro foi criada (Figura 14).
Requisito 2	Para atender essa funcionalidade uma tela de login para o usuário foi feita e quando o usuário solicitar entrar com o usuário e a senha o sistema faz consultar no bando de dado pelo login e password fornecido, caso o usuário informe os dados incorretamente, uma mensagem informando sobre o erro aparecerá, figura numeração.
Requisito 3	Para atender essa funcionalidade uma tela foi criada com todas as informações do formulário em papel (Figura 16).
Requisito 4	Para atender essa funcionalidade uma foi criada uma tela, onde o sistema faz uma querry que retorna todos os registros de vistorias feita no banco de dado, assim exibindo parte dessa lista para usuário (Figura 18).
Requisito 5	Para atender essa funcionalidade foi criada uma variável estática, da classe Usuario, e ao autentificar o usuário através de uma querry buscando seu login e senha, as informações dele são para esse objeto. O propósito da se utilização de uma variável estática se dá pelo motivo em que você tem acesso a essa variável sem ter que cria um objeto de uma classe.
Requisito 6	Para atender essa funcionalidade foi criada uma tela com uma lista de todas as vistorias que estão no banco e através dela pode-se dar um longo “click” sobre o item e automaticamente o sistema leva o usuário para tela de vistoria com todos os valores colocados na vistoria.
Requisito 7	Para atender essa funcionalidade foi feito um pop-up com duas opções para usuário “Salvar” ou “Cancelar”, quando o usuário aciona o botão “Salvar Vistoria”.
Requisito 8	Para atender essa funcionalidade a codificação do sistema foi feita em Java e na estrutura de criação de aplicações para dispositivos móveis.
Requisito 9	Para atender essa funcionalidade uma foi criada uma tela com todas as informações do formulário em papel (Figura 16). Como o formulário é baseado em passo, para melhor visualização para o usuário foram escondidos todos os componentes em seu título, mas quando o usuário dá um “click” todos os componentes aparecem e se o fizer novamente o sistema volta a esconder (Figura 17).

As figuras de 13 a 18 apresentam as interfaces criadas para atender os requisitos do sistema.

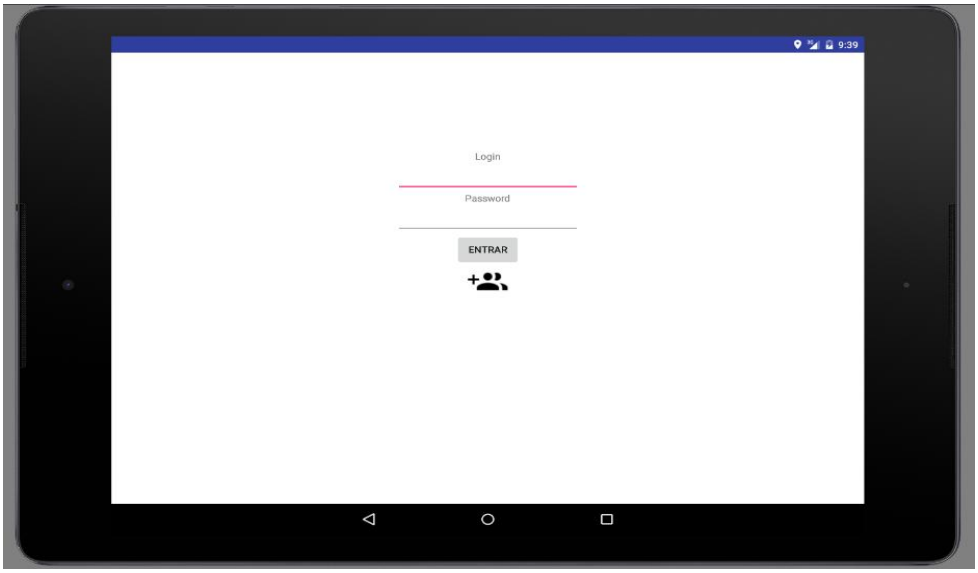

Figura 17 - Tela de login

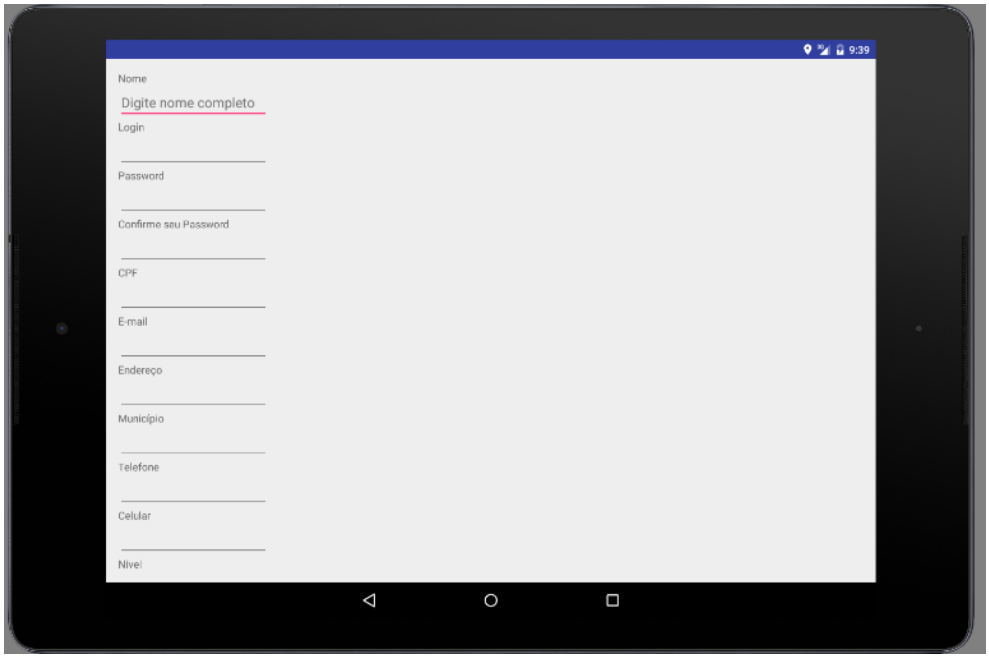

Figura 18 - Tela de cadastro

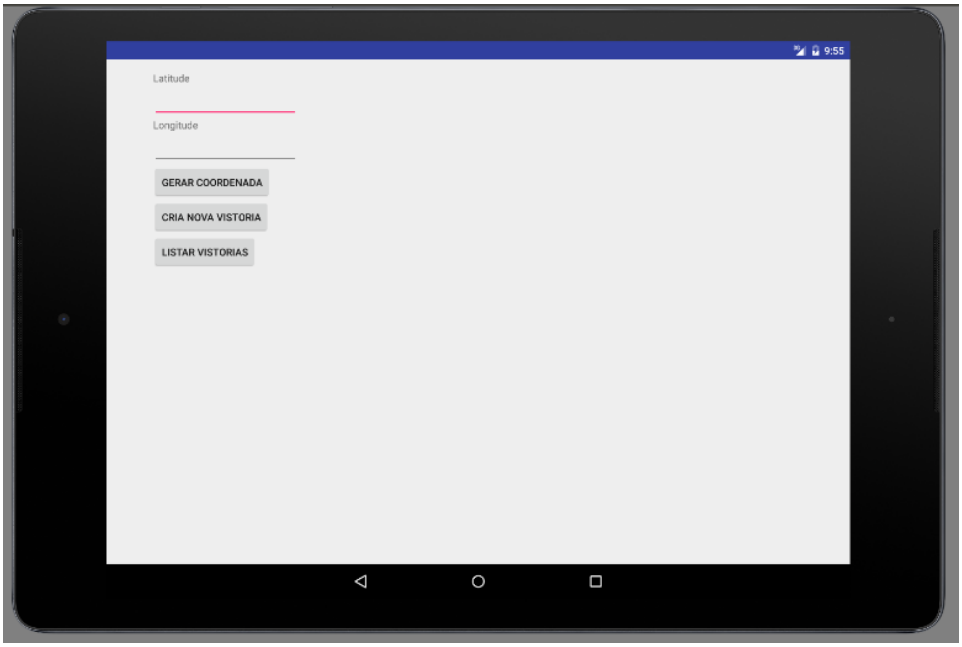

Figura 19 - Tela principal após usuário entra com login e password

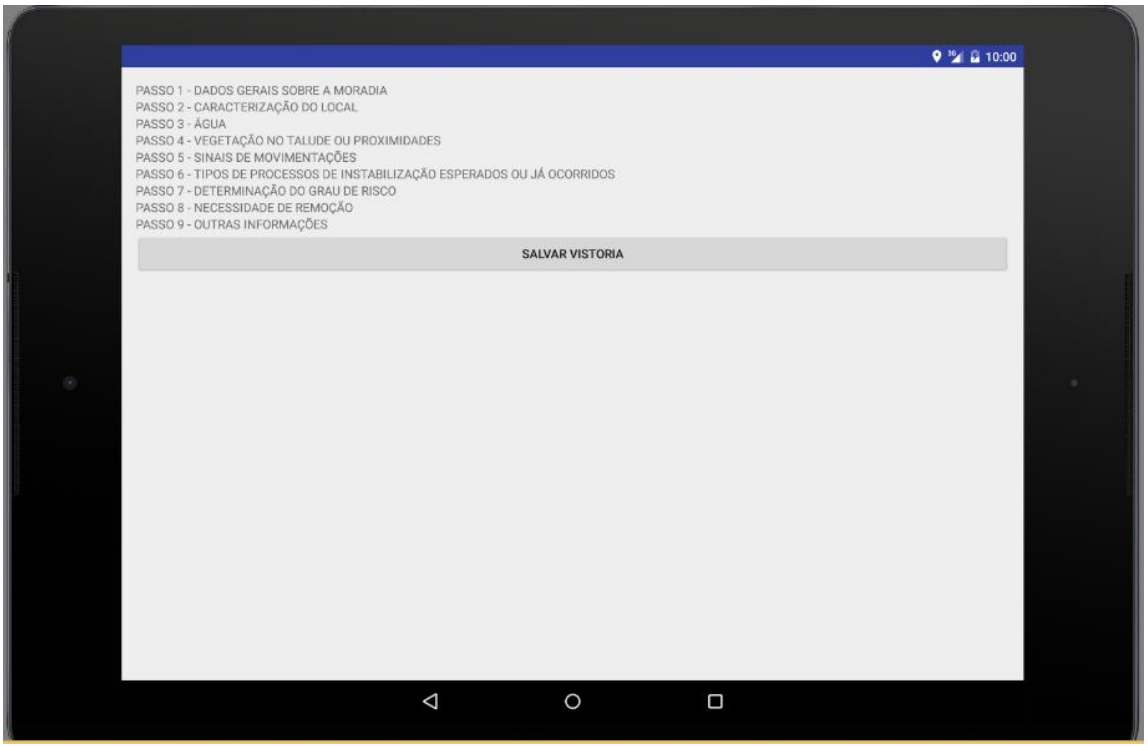

Figura 20 - Tela da vistoria

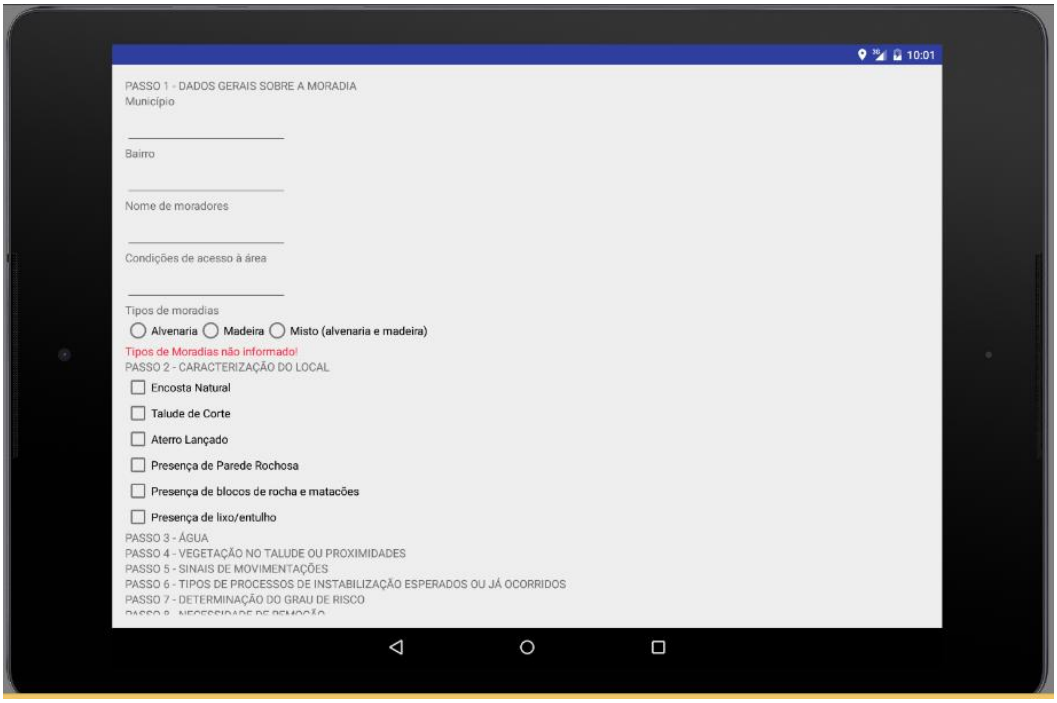

Figura 21 - Tela vistoria após o usuário ter feito um "click" sobre um dos passos

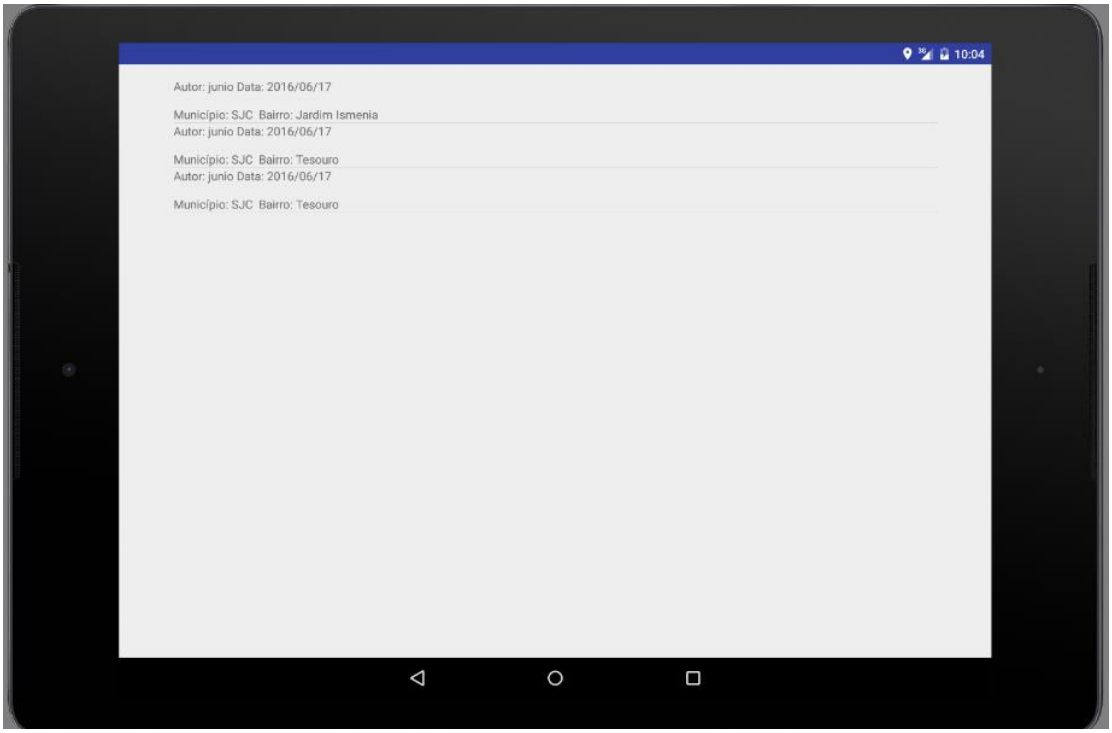

Figura 22 - Tela com uma lista de vistorias

Em análise com o cenário atual (realização de vistorias em campo com formulário em papel) e com aplicação que substituiria esse cenário pôde se concluir que por intermédio da aplicação o técnico que realizará a vistoria em campo tenha mais mobilidade na execução do trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

CERRI, L.E.S. Subsídios para a seleção de alternativas de medidas de prevenção de acidentes geológicos. Rio Claro, 2001. 78p. Tese (Livre-Docência) – Instituto de Geociências e Ciências Exatas, Universidade Estadual Paulista.

Defesa Civil. (25 de Março de 2016). Coordenadoria Estadual de Defesa Civil. Fonte: Defesa Civil do Estado de São Paulo: http://www.defesacivil.sp.gov.br/

FERNANDES, N. F.; AMARAL, C. P. Movimentos de massa: uma abordagem geológico-geomorfológica. In: GUERRA, A. J. T.; CUNHA, S. B. Geomorfologia e meio ambiente. 4.ed. Rio de Janeiro: Bertrand Brasil, 2003. Cap. 3, p. 123-194.

Fundação Getulio Vargas de São Paulo (FGV-SP). Disponível em: http://tecnologia.ig.com.br/2016-04-15/crescimento-de-smartphones-no-pais-e-de-9-em-relacao-ao-ano-passado.html. Acesso em: 21 de maio de 2016

Geológico, I. (25 de Março de 2016). Secretaria do Meio Ambiente. Fonte: Sistema Ambiental Paulista: http://igeologico.sp.gov.br/

GUIDICINI, G.; NIEBLE, C. M. Estabilidade de taludes naturais e de escavação. 2.ed. São Paulo. Edgard Blücher; Ed. da Universidade de São Paulo, 1984. 194p.

INSTITUTO DE PESQUISAS TECNOLÓGICAS DO ESTADO DE SÃO PAULO (IPT) Programa Serra do Mar: Estudo geotécnico dos principais mecanismos de instabilização na Serra do Mar. São Paulo, 1988. Relatório n 25957.

INSTITUTO DE PESQUISAS TECNOLÓGICAS DO ESTADO DE SÃO PAULO (IPT) Banco de dados de mortes por escorregamentos. Formato digital. 2014.

Macedo, E. S., Santoro, J., & Araújo, R. E. (s.d.). PLANO PREVENTIVO DE DEFESA CIVIL (PPDC) PARA DESLIZAMENTOS, ESTADO DE SÃO PAULO, BRASIL.

Tominaga, L. K., Santoro, J., & Amaral, R. (2009). DESASTRES NATURAIS: Conhecer para previnir (1ª ed.). São Paulo.

WOLLE, C.M. Análise dos escorregamentos translacionais numa região da Serra do Mar no contexto de uma classificação de mecanismos de instabilização de encostas. São Paulo, 1988. 394p. Tese (Doutorado em Engenharia) – Escola Politécnica da USP.

WOLLE, C.M. & CARVALHO, C. S. Deslizamentos em encostas na Serra do Mar - Brasil. Solos e Rochas, v. 12, p. 27-36, 1989.