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

Plataforma Web de Visualização de Dados do Sistema RockDroid

Victor Zaffalon Marra

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia de Computação

Orientadora
Prof.ª Dr.ª Maristela Terto de Holanda

Brasília
2018

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

Coordenador: Prof. Dr. José Edil Guimarães

Banca examinadora composta por:

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora) — CIC/UnB
Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo — CIC/UnB
Prof. Dr. Marcio Victorino — CIC/UnB

CIP — Catalogação Internacional na Publicação

Marra, Victor Zaffalon.

Plataforma Web de Visualização de Dados do Sistema RockDroid / Victor Zaffalon Marra. Brasília : UnB, 2018.
171 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2018.

1. RockDroid, 2. Visualização de dados geologicos, 3. Banco de dados, 4. Sistema Web

CDU 004.4

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

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

Plataforma Web de Visualização de Dados do Sistema RockDroid

Victor Zaffalon Marra

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia de 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. Marcio Victorino
CIC/UnB

Prof. Dr. José Edil Guimarães
Coordenador do Bacharelado em Engenharia de Computação

Brasília, 28 de Novembro de 2018

Dedicatória

Dedico este trabalho a minha família, especialmente meus irmãos e meus avós que me auxiliaram e me deram apoio durante o curso, também dedico aos meus amigos que vivenciaram comigo essa experiência única que é a graduação e aos professores que me ajudaram a construir o conhecimento necessário a elaboração deste trabalho.

Agradecimentos

Agradeço a minha orientadora pelo auxilio na elaboração deste trabalho, a Universidade de Brasília pela oportunidade de cursar Engenharia de Computação, ao departamento de ciência da computação, e a todos os professores da UnB que me forneceram conhecimento ao longa desta trajetória.

Resumo

Este trabalho apresenta o desenvolvimento de uma Plataforma Web que complementa o aplicativo RockDroid de coleta de dados geológicos. A plataforma é uma ferramenta que permite que o andamento da pesquisa de campo seja acompanhado por um profissional responsável durante o processo de coleta. A plataforma permite visualizar e editar os dados obtidos por meio do aplicativo, além de mostrar, com o auxilio de gráficos e tabelas, os dados de forma a facilitar o trabalho de análise e acompanhamento dos profissionais. A plataforma de visualização e de análise dos dados, e o processo de desenvolvimento da mesma encontra-se descrito neste documento.

Palavras-chave: RockDroid, Visualização de dados geologicos, Banco de dados, Sistema Web

Abstract

This document presents the development of a Web System that complements the RockDroid Mobile Application for geological data gathering. This platform is a tool that allows the monitoring of a geological group field research by a responsible professional. The Web System will allow the visualization and edition of the data gathered by the Mobile Application, besides it shows, with the help of graphs and data tables, the data in a way that will facilitate the job of the professional monitoring the field research. The data visualization and data analysis platform and the process of building it will be fully described in this document.

Keywords: RockDroid, Visualizing geological data, Database, geology, Web System

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

Os dispositivos móveis desenvolvidos com a tecnologia atual possuem hardware e funcionalidades que os tornam úteis no contexto acadêmico e no desenvolvimento de pesquisas. As tecnologias desenvolvidas para ambientes de computação móvel, em conjunto com o crescente número de aparelhos de uso pessoal, permitem que usuários tenham acesso a uma rede de compartilhamento de dados independente de sua localização física.

Os recursos presentes nos aparelhos móveis permitem a obtenção de dados de pesquisa por meio de fotos, aúdios, textos e localização geográfica. A localização pode ser obtida com o auxílio de ferramentas como o *Global Positioning System*(GPS), a rede de telefonia celular ou uma conexão com a Internet.

Esse ambiente de computação móvel permite que pesquisadores gravem essas diversas fontes de dados e as envie a outras máquinas na rede, permitindo que equipes trabalhem em conjunto trocando dados e informações relevantes durante o processo de pesquisa em campo.

Assim partir dessa idéia surgiu o aplicativo RockDroid que propós atender a necessidade de geólogos durantes suas pesquisas de campo, facilitando a coleta, o armazenamento e a sincronização dos dados usando seus aparelhos celulares e acessando uma rede de computação móvel para o compartilhamento dos dados.

O aplicativo foi desenvolvido com a funcionalidade de salvar os dados obtidos mesmo em situações de insistência de conexão com a Internet, Posteriormente, foi desenvolvida a funcionalidade de trocar os dados entre os aparelhos celulares distintos por meio da conexão por uma rede *Ad hoc*. Por último, neste processo foi desenvolvido e implementado um algoritmo que permite enviar dados a uma base de dados central por meio de um modelo de tranferências de dados chamado de REST(*Representational State Transfer*), sem prejudicar a integridade e a consistência dessa base.

Contudo, Para complementar a estrutura desse ecossistema faz-se necessário a criação de um banco de dados central, que sincronize os dados com qualquer outro aparelho, utilizando para isso o algóritmo de sincronização.

Assim sendo, os profissionais da geólogia indentificaram como necessário o desenvolvimento de uma plataforma Web que permita visualizar, editar, adicionar e remover os dados do banco central de forma que os pesquisadores possam acompanhar o andamento da pesquisa de campo, além de obter por meio de gráficos, tabelas e filtros estatísticas

que auxiliam a análise dos dados obtidos. Esse acompanhamento deve ser realizado pelo coordenador do projeto.

1.2 Objetivo

Este trabalho tem como objetivo desenvolver um SIG Web que complemente o sistema de coletas de dados geográficos RockDroid. A plataforma é responsável por permitir a visualização dos dados obtidos pelo aplicativo, e sincronizados com o banco de dados.

1.2.1 Objetivos Específicos

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

- Desenvolver um serviço Web responsável por acessar, receber, salvar e sincronizar dados obtidos pelo aplicativo RockDroid com o banco de dados;
- Criar um painel de administração de que permita ao usuário, com autorização, a habilidade de alterar os dados e os objetos relacionados dentro do banco de dados do sistema;
- Desenvolver um Sistema Web que acesse esse banco de dados e mostre os dados de maneira relevante aos profissionais da geologia de forma a garantir que um responsável acompanhe a coleta de dados durante uma saída de campo.

1.3 Estrutura do Trabalho

Este documento é composto pelos seguintes capítulos:

- Capítulo 2 - Fundamentação Teórica: Apresentação dos conceitos necessários para o entendimento da trabalho;
- Capítulo 3 - Arquitetura do Sistema Web: Apresentação da arquitetura geral do sistema RockDroid antes do desenvolvimento do projeto, em conjunto com a apresentação da arquitetura que se deseja obter com a conclusão deste trabalho;
- Capítulo 4 - Desenvolvimento do sistema: O processo de desenvolvimento do software e a aplicação das tecnologias apresentadas, descrição das rotas de acesso as interfaces dos sistema, descrição do fluxo de telas e detalhamento das telas desenvolvidas;
- Capítulo 5 - Conclusão: Apresentação das conclusões e dos trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Para iniciar o desenvolvimento do projeto é necessário o entendimento do que é um SIG web, e dos modelos e protocolos que cercam o desenvolvimento de aplicações Web. Assim na seção 2.2 serão descritos os protocolos e os modelos usados no processo de desenvolvimento. Para obtenção de dados referentes aos protocolos foram consultados RFCs, documentos técnicos acerca de informações sobre o funcionamento da Internet que definem alguns conceitos sobre redes de computadores, protocolos, programas e procedimentos.

2.1 Sistema de Informação Geográfica

2.1.1 Introdução

Um SIG (Sistema de Informação Geográfica) consiste em um sistema que realiza tratamentos computacionais sobre dados geográficos, esse tipo de sistema deve possuir a habilidade de auxiliar profissionais que executam seu trabalho com auxílio de mapas. Para isso, é necessário o armazenamento em conjunto com dados convencionais, de diferentes tipos de dados geográficos [12].

Para representação de dados em um SIG é comum o uso de mapas temáticos, esse tipo de mapa é usado para representar dados focados em apenas um tema dentre os diversos existentes no ramo da geografia e geologia [12].

Em um SIG é necessário a representação de dados espaciais, seguindo um modelo vetorial, esses tipos de dados podem ser representados por pontos, linhas ou polígonos. A maioria dos mapas de um SIG são construídos por uma matriz, logo, a representação de linhas e polígonos é feita por um conjunto de pontos e posições da matriz. Como consequência quanto maior for a matriz que representa uma área, menor será a área que uma posição da matriz representa e maior será a resolução espacial e a precisão do mapa [15].

Um SIG pode agrupar e tratar um conjunto de dados geográficos para uma ampla gama de aplicações, e para a resolução de diferentes tipos de problemas. No fim o SIG é, principalmente, usado como base para alcançar três objetivos de pesquisa [10]:

- Produção de mapas;
- Criação de um banco de dados geográficos, com a capacidade de armazenar e recuperar informações espaciais;

- Dar suporte a profissionais da geociência para a análise espacial de fenômenos.

2.1.2 Arquitetura Integrada

Um sistema para ser considerado um SIG deve conter uma arquitetura composta por [21] [24]:

- Interface com o usuário;
- Banco de dados geográfico;
- Módulo de entrada e integração de dados;
- Módulo de consulta e análise espacial dos dados;
- Módulo de visualização e plotagem dos dados;
- Módulo de gerência de dados espaciais.

Os componentes citados podem ser implementados seguindo diferentes modelos. O modelo usado neste trabalho é chamado de arquitetura integrada, essa arquitetura armazena tanto dados geográficos como dados convencionais em um mesmo banco de dados. Isso permite a utilização de um SGBD (Sistema de Gerenciamento de Banco de Dados) para manipular objetos, gerenciar transações e realizar consultas, tanto para dados geográficos como para dados comuns. Os outros modelos se baseiam em armazenar os dados geográficos em arquivos fora do ambiente do SGBD, o que inibiria o uso das ferramentas do SGBD [21].

2.1.3 Conversão de Dados Geográficos

Para se obter informações geográficas úteis para o desenvolvimento de uma pesquisa a partir de dados geográficos faz-se necessário seguir um modelo de ciclo de extração e utilização de informações definidos pelas etapas [12]:

- Aquisição de dados;
- Modelagem e integração dos dados em uma única base de dados;
- Armazenamento dos dados em banco de dados geográfico;
- Consulta, manipulação e análise dos dados;
- Utilização das informações obtidas para tomada de decisão.

Um SIG deve auxiliar o trabalho nas etapas intermediárias, nas quais a maior parte dos processos pode ser automatizados, com o objetivo de que o pesquisador dedique seu tempo na utilização das informações obtidas para analisar os dados processados, apresentar resultados e tomar decisões futuras, ao invés de desperdiçar tempo com trabalhos manuais.

2.1.4 SIG Web

Um SIG Web é definido como qualquer SIG que utilize tecnologias da Web para sua operação. O SIG Web permite que pesquisadores utilizem um SIG a partir de qualquer cliente que contenha um navegador web, seja ele um celular, ou uma aplicação *desktop* [26].

O uso de tecnologias Web permite que dados sejam enviados para um servidor remoto, e sejam retornados e visualizados, por exemplo, em um celular após o processamento ter sido realizado, sem a necessidade de grande uso computacional do cliente utilizando o sistema [20].

Para representação de dados geográficos em um SIG Web existem alguns tipos de mapa *Online*, dentre eles existe o mapa colaborativo no qual qualquer cliente pode contribuir para a criação e o melhoramento dos dados existentes. Nesse caso que os dados enviados por usuários passa por uma verificação e controle de qualidade, eles são adicionados ao mapa. Dentre os mapas colaborativos *onlines* existentes o OpenStreetMap (http://www.openstreetmap.org) foi selecionado para uso neste projeto [1] [8].

A facilidade do acesso remoto por qualquer dispositivo com acesso a Internet, e a facilidade de difusão de dados provida pela web, vem em conjunto com alguns problemas que se não resolvidos atrapalham o uso de um SIG web [8]:

- Muitos mapas da Web, incluindo mapas colaborativos, não apresentam grande precisão dos dados e a qualidade dos mapas pode não ser boa;
- A renderização de mapas na Web requer uma largura de banda significativa, sendo assim, se o cliente não possuir uma boa conexão a rede, os dados no mapa demoram a carregar ou ate mesmo não carregam por completo;
- A criação de mapas na Web ainda é uma tarefa complexa que depende da integração de vários módulos e do uso de diversas ferramentas;
- Direitos autorais relativos ao uso indevido dos dados disponibilizados na Web.

O OpenStreetMap é um mapa que busca solucionar esses problemas apresentando qualidade e precisão de dados boas, que não necessita de uma largura de banda tão alta para renderização do mapa em navegadores Web e que opera sob a licença ODbL(*Open Data Commons Open Database*) que permite que qualquer um possa utilizar os dados desde que a autoria seja atribuída ao OpenStreetMap e aos seus contribuidores [1].

O modelo mais simples e mais comum de um SIG Web é divido em três camadas [15] [20]:

- Interface;
- Servidor de aplicação;
- Camada de banco de dados.

Esse modelo é subdivido em Arquitetura *thin client* e *thick client*, na *thin client* praticamente todo o processamento dos dados é feito no servidor da aplicação, enquanto na *thick client* parte do processamento dos dados é realizado no cliente, ou seja no web *browser* acessando o SIG Web [15].

Para implementação de um SIG Web com a Arquitetura *thick client*, que segue um modelo de SIG integrado é necessário entender como funcionam os modelos de transferência de dados mais comuns a aplicações Web. Para isso, nas seções a seguir são apresentados o funcionamento do modelo REST e da arquitetura cliente/servidor que foram utilizadas neste projeto [20].

2.2 REST

A arquitetura REST (*Representational state transfer*) define uma série de princípios que devem ser seguidos ao desenvolver um serviço Web, de maneira que ele atue como um serviço que possa ser usado por uma grande quantidade de clientes, mesmo com características distintas e escritos em diferentes linguagens, por meio da transferência de dados pelo protocolo HTTP. É o modelo de serviço predominante na Internet desbancando modelos antigos como o SOAP e o modelo de *design* baseado no WSDL [22].

A arquitetura REST em sua forma mais concreta segue quatro princípios básicos [22]:

- Métodos HTTP explícitos;
- Independência de estado;
- Exposição de URIs seguindo a estrutura de diretórios;
- Transferência de dados por XML, JSON ou ambos.

Nas seções a seguir serão explicados por que esses princípios são necessários no *design* de um arquitetura REST.

2.2.1 Métodos HTTP Explícitos

O uso de métodos HTTP explícitos seguindo o protocolo definido pela RFC 2616 [11] é um ponto chave para manter a estrutura desse modelo. Esse principio básico do REST estabelece um mapeamento de um-para-um entre as ações de criar, ler, atualizar e deletar (CRUD) dados de um banco de dados e entre ações do método HTTP. Seguindo esse mapeamento tem-se que:

- Para criar um elemento no servidor, use uma requisição do tipo POST;
- Para buscar um elemento, use uma requisição do tipo GET;
- Para alterar os atributos ou estado de um elemento, use uma requisição do tipo PUT;
- Para remover ou deletar um elemento, use uma requisição do tipo DELETE.

Não seguir esse mapeamento a risca é considerada uma falha crítica no desenvolvimento de uma arquitetura REST, em sistemas desenvolvidos incorretamente a arquitetura possui comportamentos inesperados durante seu uso. Requisições HTTP do tipo GET podem, por exemplo, serem usadas para criar elementos no servidor ou até mesmo alterar atributos de um determinado recurso, quando de maneira correta deveriam ser utilizadas estritamente para a obtenção de recursos [22].

Essa quebra do padrão imposto para a arquitetura inválida a sua característica de simplicidade. Causando como consequência, por exemplo, que clientes como ferramentas de buscas e *Web Crawlers* possam acabar modificando dados no servidor, fazendo requisições GET quando na verdade possuem o objetivo de apenas receberem recursos do serviço.

2.2.2 Independência de estado

Serviços REST são construídos para serem escaláveis e, consequentemente, manterem sua performance com o aumento do número de clientes, seu objetivo é diminuir ao máximo o tempo de resposta de uma chamada a um serviço do sistema Web [22].

Para atingir esse objetivo, o REST é construído de forma que todo o dado necessário para que a requisição seja completada está incluso no corpo, nos cabeçalhos e nos parâmetros da requisição HTTP. Dessa maneira, tem-se que toda requisição é independente, e não necessita que variáveis de estado sejam guardadas no servidor, como consequência, para toda requisição, o servidor possui todos os dados necessários para gerar uma resposta sem necessidade de esperar a verificação de um estado passado [22].

Assim, Um exemplo prático da diferença entre um processo dependente de estado e um processo feito com a arquitetura REST é analisado a seguir. A Figura 2.1 exemplifica o processo de busca de dados de uma página em um ambiente multi-paginado, sendo uma página nesse contexto definido como um conjunto de dados do servidor subdivididos em sequência por um *index*. Após a busca da primeira página o serviço guarda qual foi a página acessada anteriormente, e o estado atual de qual deve ser a próxima página buscada. Considerando um ambiente com vários clientes fazendo requisições, salvar o estado, para cada cliente, causa um grande *overhead* no sistema e desperdício de recursos. Além do *overhead* o Serviço web está suscetível a problemas de sincronização que devem ser resolvidos, e aumentam ainda mais o *overhead* do sistema [22].

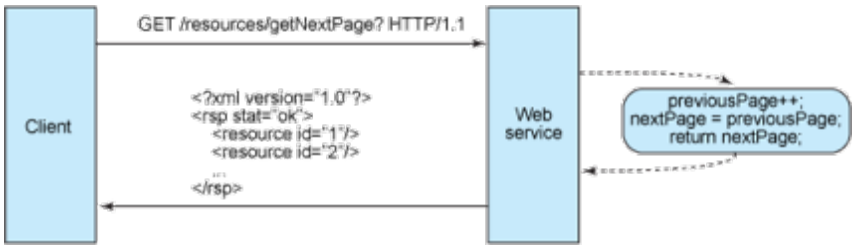


Figura 2.1: Serviço web sem independência de estado [17].

Na Figura 2.2 é possível ver o caso em que o Serviço web não armazena o estado da página anterior e o contador de qual a próxima página a ser buscada pelo cliente. O cliente se torna o responsável por passar como parâmetro da requisição o número da página a ser buscada, como consequência, ele se torna o responsável por manter o estado [22].

Esse modelo acaba com o *overhead* de sincronização e de gasto de recursos por parte do serviço web. Isso torna o serviço web mais próximo de um serviço que pode ser utilizado para qualquer tipo de cliente, apenas respondendo aos parâmetros da entrada com uma saída, sem criar nenhum vinculo desnecessário com o cliente [22].

Figura 2.2: Serviço web com independência de estado [17].

Em resumo, para criar um servidor independente de estado é necessária a implementação das seguintes funcionalidades [22]:

No Servidor:

- Respostas para requisições que incluem links necessários para acessar outros recursos da aplicação permitindo o acesso lógico a elementos relacionados. Por padrão um exemplo prático é a busca de um objeto pai que apresenta um ou múltiplos objetos filhos no modelo de orientação a objetos. O serviço Web deve retornar o id de acesso ou a rota para o(s) objeto(s) filho(s) naquele servidor [22];
- Incluir na resposta do serviço Web se o objeto pode ser colocado em cache ou não reduzindo o número de requisições para recursos duplicados e ate mesmo eliminando requisições de recursos armazenados em cache completamente. O servidor faz isso incluindo os Headers de Cache-Control e Last-Modified na cabeçalho das respostas HTTP [22].

No Cliente:

- O Cliente deve usar as informações de *Cache-Control* na resposta das requisições para decidir se uma copia local do dado deve ser criada ou não. O cliente deve também ler os dados de *Last-Modified* e enviar na resposta a data que o recurso foi modificado novamente quando alterado. Uma requisição HTTP de codigo 304 voltada pelo servidor indica que a copia local do cliente pode ser usada [22];
- O cliente é responsável por enviar todos os dados necessários para que o servidor responda imediatamente sem precisar armazenar ou buscar variáveis de estado. O cliente no geral não deve assumir que o serviço Web guarde qualquer informação, contexto ou estado entre requisições a não ser para algumas raras exceções como o caso do armazenamento sessões de login no servidor.

2.3 Arquitetura Cliente/Servidor

Na comunicação entre um par de processos da arquitetura cliente/servidor tem-se que [17] [27]:

- **Cliente:** o processo que inicializa a comunicação e manda um sinal para identificar o outro processo no início da sessão de comunicação.

- **Servidor:** o processo que espera receber o contato no inicio de uma sessão.

No contexto de sistemas Web tem-se que essa arquitetura opera com um servidor Web que busca estar sempre ligado, com endereço de IP fixo conhecido e que fornece objetos Web para possivelmente milhões de *Web Browsers* diferentes. Deve ser possível acessar e utilizar um serviço Web de qualquer lugar do planeta apenas utilizando um dispositivo que consiga, de alguma forma, encontrar e acessar o endereço no qual o serviço está hospedado. Nessa arquitetura clientes não comunicam-se diretamente, é necessário que o servidor faça uma intermediação [17].

O cliente e o servidor nessa arquitetura não compartilham memória, logo torna-se inválida a utilização de métodos de comunicação comuns a sistemas operacionais (filas de mensagens, semáforos, monitores, etc.) que necessitam o acesso ao mesmo contexto de dados. A troca de mensagens pela rede é o método utilizado para estabelecer essa comunicação entre cliente e servidor [27].

A troca de mensagens é feita por meio de um protocolo, protocolo nesse contexto pode ser definido como um conjunto de regras e passos, que consiga garantir que as mensagens trocados sejam corretamente escritas e lidas, tornando possível a comunicação entre os dois lados [18]. Uma arquitetura cliente/servidor deve levar em conta o canal da comunicação, visto que possíveis problemas no canal, como falta de banda e baixa velocidade de tráfego de informações, são problemas comuns que precisam ser considerados [18].

No contexto Web o processo cliente e o processo servidor estão fisicamente distantes, para se estabelecer uma conexão, o processo envia uma mensagem pela rede por uma interface de software denominada *Socket* [17]. O processo deve assumir que existe uma infraestrutura de transporte que permita o envio da mensagem do remetente ao destinatário com segurança. Logo, essa arquitetura precisa se apoiar fortemente em protocolos, tanto da camada de redes, transporte e aplicação para seu funcionamento correto [18].

Os processos remetentes e alvo são identificados pelo endereço de IP e uma porta associada ao processo. No geral, a porta 80 de acesso em um computador identifica um servidor Web. Existem diversas aplicações que utilizam esse modelo tais como: Web, FTP(File Transfer Protocol) e Telnet [17]. Por entrar na categoria Web o projeto desenvolvido também segue essa arquitetura.

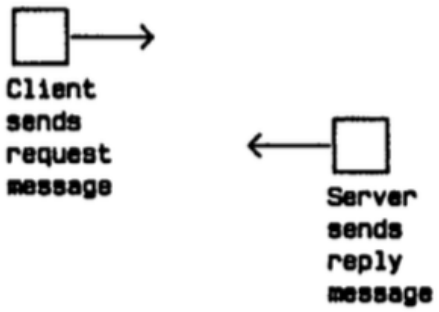


Figura 2.3: Interação cliente servidor [17].

A Figura 2.3 mostra a comunicação iniciada pelo cliente, que faz um pedido para o servidor. O servidor recebe o pedido, processa-o e devolve uma mensagem como resposta a

requisição. Essas operações(primitivas) que seguem o modelo de pedido e de resposta são definidas como confiáveis ou não-confiáveis, bloqueadas ou não-bloqueadas, bufferizadas ou não-bufferizadas que são descritas a aseguir [17] [27]:

- **Confiáveis ou não-confiáveis:** Primitivas confiáveis apresentam garantia de entrega da mensagem. Normalmente, é utilizado um *acknowledgement* (ou ACK), que indica que uma mensagem foi reconhecida ou recebida com sucesso. O ACK deve ser enviado pelo receptor;
- **Bloqueadas ou não-bloqueadas:** Primitiva bloqueadas fazem com que um dos lados da comunicação fique com a sua execução suspensa até o momento em que a resposta da solicitação é recebida;
- **Bufferizado ou não-bufferizado:** Primitivas bufferizadas armazenam mensagens enviadas que não foram atendidas, enquanto primitivas não-bufferizadas descartam essas mensagens, e as retransmitem posteriormente.

Capítulo 3

Arquitetura do Sistema Web

Neste capitulo é apresentado a arquitetura do sistema RockDroid e a nova arquitetura que foi obtida após o desenvolvimento do projeto. Além disso é descrito as ferramentas escolhidas para o desenvolvimento, suas características e as motivações para suas escolhas. Com o conjunto das ferramentas é definido a arquitetura do funcionamento sistema.

3.1 Arquitetura do Rockdroid

3.1.1 Arquitetura original

Na arquitetura original com *smarthphones* do RockDroid tem-se um aplicativo desenvolvido para o sistema operacional Android que coleta os dados obtidos durante a saída de campo e grava por meio de textos e fotos as informações em um banco de dados local sem necessidade do uso da internet [9].

Na segunda etapa do projeto RockDroid o aplicativo foi desenvolvido para se comunicar sem uma rede wi-fi com o mesmo aplicativo em outros dispositivos, permitindo a troca de dados de projetos obtidos durante uma saída de campo entre vários dispositivos por meio de uma rede Ad hoc. Cada um desses dispositivos contém o seu próprio banco de dados local para salvar informações mesmo em situações de conexão com a internet inexistente [29].

Na Figura 3.1 é apresentado a estrutura original do sistema. Vários aplicativos executando na plataforma Android conseguem se comunicar para troca de dados dos projetos salvos em um banco de dados SQLite no próprio celular.

Nessa estrutura os dados obtidos por todos os pesquisadores não possuem um banco de dados central com uma interface que consegue receber e salvar todos os dados permanentemente. Caso ocorra algum problema com o celular, dados necessários para a pesquisa, gravados durante a saída de campo, serão perdidos.

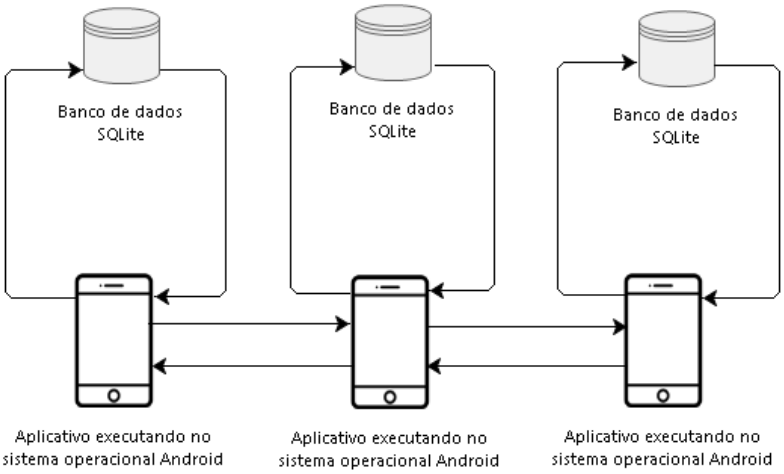


Figura 3.1: Estrutura do Rockdroid antes do Projeto.

A estrutura não apresenta um mecanismo, seja ele feito por uma aplicação mobile, Web ou Desktop, que agregue todos esses dados e apresente de maneira inteligente, por meio de gráficos ou tabelas, os dados obtidos. Implementando essa funcionalidade é permitido que alunos e os professores envolvidos no processo de pesquisa consigam fazer uma análise detalhada dos dados obtidos [9].

3.1.2 Arquitetura Proposta neste Trabalho

Com o objetivo de tornar o sistema completo é criado neste projeto uma interface para receber os dados obtidos por meio da sincronização entre aplicativo e banco de dados central. O algoritimo de sincronização já esta implementado pelo aplicativo, portanto, cabe a este projeto o trabalho de criar uma interface Web para receber as requisições HTTP enviadas pelo algoritimo.

Assim a construção da arquitetura pode ser dividida em duas etapas:

- Na primeira etapa do desenvolvimento é feita a implementação da interface Web responsável pelo processo de receber e atualizar os dados enviados por meio do algoritimo de sincronização, além da criação de um painel de administração do banco de dados;
- Na segunda etapa um visualizador web é criado que permita listar, excluir e atualizar dados de qualquer elemento da estrutura do projeto, além de mostrar por meio de gráficos, listas e filtros relatórios úteis a tomada de decisão por meio do responsável por acompanhar a saída de campo.

Os elementos do painel administrativo foram desenvolvidos com o suporte de *frameworks* Web especializados em desenvolvimento de interfaces. Assim o objetivo da segunda etapa é além do funcionamento, obter uma boa experiência e interface de usuário estimulando e incentivando o uso do sistema por parte dos pesquisadores.

Na Figura 3.2 é ilustrada a estrutura objetivo do projeto. Seguindo o diagrama, é possível ver que além da comunicação entre os aplicativos tem-se o processo de inserção

de dados no banco central, e a comunicação entre banco de dados e a interface web de visualização.

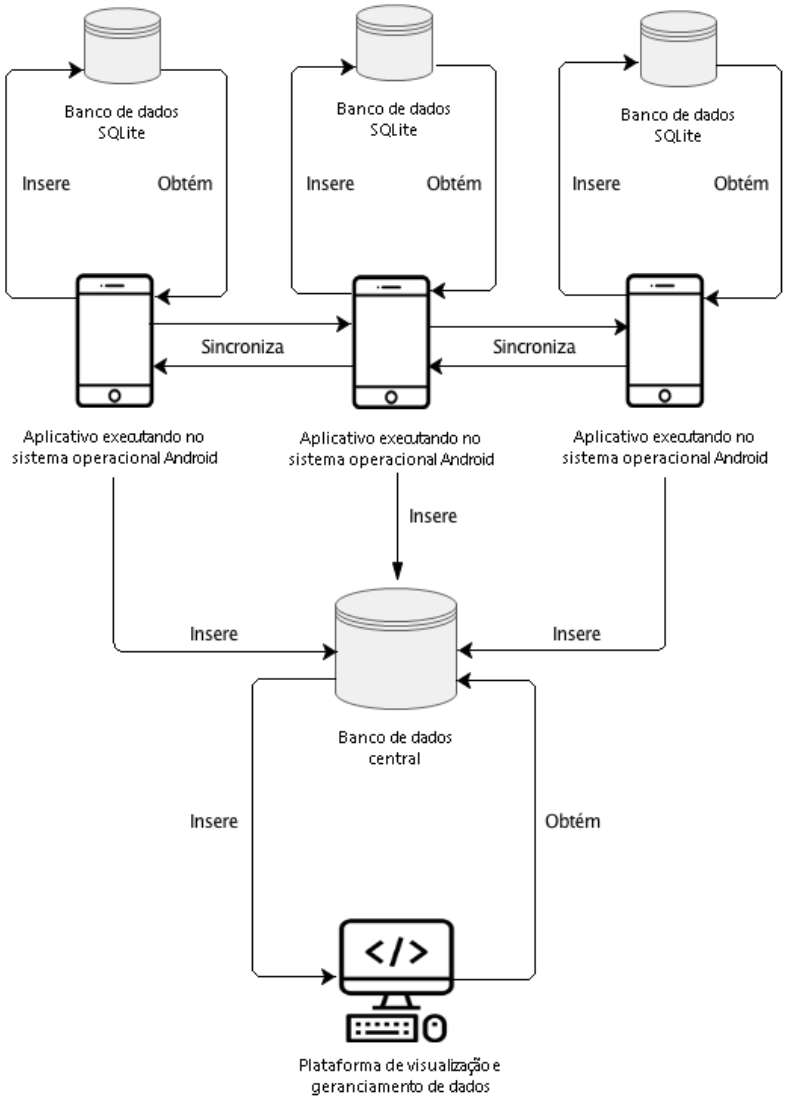


Figura 3.2: Estrutura objetivo do projeto.

3.2 Arquitetura do Back-end

3.2.1 Introdução

O Back-End é o elemento com a maior parte das regras de negócio da aplicação, sendo responsável por qualquer ação referente ao bancos de dados, também gera URIs de acesso a recursos referentes aos objetos obtidos a partir do acesso ao banco. O *Back-End* é o responsável por cuidar do processo de receber e responder as requisições HTTP de acesso ao banco de dados, ele é quem deve implementar as lógicas de autenticação para acesso aos recursos da aplicação e garantir a segurança do acesso aos dados do sistemas.

Além disso, deve ser responsável por guardar os documentos da aplicação como páginas Webs que devem ser retornadas, imagens, arquivos, configurações, senhas criptografadas. A maior parte da lógica da aplicação deve ser feita no *Back-End* visto que ele é criado para aguentar o processamento da aplicação com maior nível de complexidade algorítmica.

O servidor executando, o *Back-End* da aplicação deve conter especificações mais potentes do que uma máquina executando um servidor responsável pelo *Front-End* da aplicação.

No desenvolvimento do sistema Web deste projeto foi utilizada uma ferramenta de auxílio ao desenvolvimento de aplicações Web chamada de *Ruby on Rails*[16]. O RoR é um *framework open-source* construído utilizando a linguagem Ruby que segue o padrão MVC (*Model-View-Controller*) de desenvolvimento [23].

O *framework* foi construído com o objetivo de usar esse padrão para trabalhar com a maioria dos banco de dados relacionais existentes, e aplicados em sistemas Web como o Mysql, MS SQL Server, IBM DB2 e PostgreSQL [5].

O *framework* também possui uma das maiores comunidades de desenvolvimento existentes, sendo até mesmo utilizados por empresas como Twitter, Github e Basecamp [16].

A linguagem Ruby possui as características de ser interpretada, fracamente tipada, *open-source*, orientada a objetos com sistema de gerência de memória e recursos abstraídos do programador. Ela foi criada com o objetivo de ser simples, produtiva e elegante assim como o próprio *framework Rails* [28].

O Rails é baseado em algumas doutrinas e convenções criadas com o intuito de tornar o processo de desenvolvimento mais prazeroso simples e produtivo possível. A lista das principais propriedades são apresentadas a seguir [16]:

- Otimizado para a felicidade do programador;
- Convenção sobre configuração;
- O programador é obrigado a usar as ferramentas do Rails;
- Qualidade de código;
- Não seguir um único paradigma;
- Valorizar sistemas integrados.

Esses princípios são a base do motivo pelo qual esse *framework* é amplamente usado. O Rails possui ferramentas para realizar os trabalhos comuns ao desenvolvimento de um sistema Web tais como: mapeamento entre objetos e as tabelas do banco de dados, estruturação de arquivos, gerência de email, ferramentas de teste de código, convenções de nomeação de arquivos e variáveis, sistema de migração de dados do banco, construção de APIs RESTfull e configuração de suas URIs de acesso, sistema de deploy para produção [16, 23].

Com essas ferramentas e com o princípio da convenção sobre configuração, o desenvolvimento se torna rápido e com poucos erros, o código se torna menor e fácil de se entender por outros programadores visto que todos são obrigados a seguir as convenções de desenvolvimento e o padrão MVC do Rails [23].

Em suma, o Rails tira o trabalho do programador de gastar uma grande quantidade de tempo configurando o projeto e definindo arquiteturas em troca de uma configuração

e arquitetura padrão que acelera o tempo de desenvolvimento, e mantém um padrão de alta qualidade [5, 23].

3.2.2 Padrão MVC

O Rails utiliza o padrão de desenvolvimento MVC com o objetivo de ampliar a manutenibilidade do código, padronizando a arquitetura que deve ser usada por todos os desenvolvedores que utilizam o *framework* [23]. O MVC é constituído por três camadas:

- *Model* - Modelo de dados;
- *View* - Camada de visualização dos dados;
- *Controller* - Ponte entre *Model* e *View*.

A seguir é descrito o papel de cada uma dessas camadas no desenvolvimento de uma aplicação construída com base no *Ruby On Rails*:

- **Modelo(*Model*)**

  O Model é o responsável pela parte lógica da aplicação no *Ruby on Rails*, manipulando diretamente os dados da aplicação, e mapeando os relacionamentos entre as tabelas do banco de dados relacional. Ele é o responsável pela validação dos dados recebidos antes de uma ação ao banco de dados ser realizada garantindo que apenas dados válidos sejam inseridos no sistemas [13].

  A construção e a execução de consultas de busca, inserção, deleção e atualização no banco também são realizadas pela camada do Model. O mapeamento de relacionamentos entre tabelas de 1 para N, N para N, 1 para 1, e a criação das tabelas necessárias para esse procedimento é implementada a partir da classe pai das ações de Model do *Ruby On Rails* chamada de *Active Record* [5].

- **Visualização(*View*)**

  A camada de visualização deve construir a interface com o usuário, coletar e mostrar os dados do usuário ou cliente a partir de formulários. O conteúdo da *view* pode ser construído com várias ferramentas como HTML, PDF, RSS, JSON, XML [13].

  O *Ruby On Rails* no desenvolvimento deste trabalho foi usado apenas para construir uma estrutura REST, logo a *View* não será um documento em formato HTML e sim um objeto no formato JSON responsável por apresentar os dados da resposta de requisições feitas na API, esse objeto JSON será consumido pelo *Front-End* da aplicação para exibição dos dados.

- **Controlador(*Controller***

  O controlador faz a ligação entre o *Model* e a *View*, quando uma requisição HTTP é enviada para uma URI, configurada no arquivo de configuração de rotas do Rails (routes.rb). O tratamento dessa requisição é iniciado por uma função do *controller* definida para essa rota [5, 13].

O controlador tem o trabalho de acessar os modelos com a lógica de negócio, receber os dados, renderizar a *View* e enviar de volta ao cliente como resposta da requisição HTTP feita [5].

No Rails lógicas de negócio podem ser feitas no controlador, mas o ideal para manter o padrão é manter toda lógica nos modelos e utilizar o *controller* apenas como uma interconexão entre as partes do modelo padrão MVC [23].

3.2.3 Detalhamento do Processo de Requisição

Na Figura 3.3 pode-se ver um processo de maneira simplificada de como uma requisição é tratada no *Ruby On Rails*. O servidor Web recebe uma requisição vinda de um cliente, podendo ser ele um *browser*, um aplicativo ou até mesmo um dispositivo embarcado.

O servidor passa para o sistema de roteamento do Rails que é a URI da requisição, o arquivo de configuração de rota do *rails* é consultado para mapear para qual *controller* e função do *controller* aquela URI deve chamar. Todos esses dados consultados são passados para o *Dispatcher* que realiza o trabalho de instanciar o *controller* e chamar a função solicitada.

Na segunda etapa o *controller* busca os dados requisitados obtidos a partir do *model* e os relaciona com a *View*, e depois chamada a função de renderização para gerar a resposta da requisição que é enviada de volta para o servidor Web responder o cliente.

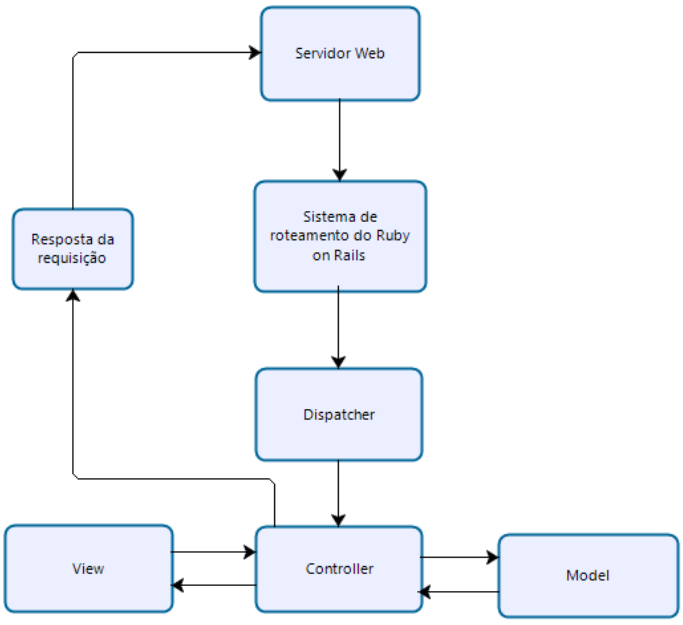


Figura 3.3: Relacionamento entre elementos do Ruby On Rails.

3.2.4 Ferramentas do Rails

A seguir serão apresentadas as ferramentas do ecossistema do *Ruby on Rails* que foram utilizadas durante o desenvolvimente deste trabalho:

- **RubyGems**

O mecanismo de compartilhamento de bibliotecas e de gerenciamento de dependências utilizado pelo *framework Rails* é o mesmo utilizado para qualquer projeto usando a linguagem ruby chamado de RubyGems.

Com o uso do RubyGems as bibliotecas podem ser adicionadas e baixadas para o projeto apenas adicionando uma linha ao arquivo da lista de gemas (gema é o nome dado as bibliotecas do ruby). Existem mais de 100 mil gemas no repósitorio geral do Ruby sendo a maioria *open-source*, testadas e, principalmente, voltadas para solucionar problemas recorrentes de desenvolvimento Web [3].

- **Serializers**

  Serializers é o nome dado a classes do Rails responsáveis por definir qual campos de informação de um determinado objeto devem aparecer na resposta da requisição [23]. Os *serializers* foram muito usados durante o desenvolvimento dessa aplicação para garantir que objetos retornados por querys ao banco de dados mostrem para a api, apenas o campos necessários para aquela requisição, diminuindo a poluição de dados em requisições e garantindo a segurança no retorno dos dados [5, 16].

- **Funções de permit**

  Permit é uma função particular do *Rails* que funciona quase da mesma maneira que os *serializers*, a principal diferença é que o *permit* opera no processo de recebimento de dados em uma requisição ao servidor Web.

  A função de *permit* é usada para garantir que apenas os parâmetros declarados no *permit* possam ser recebidos pela api, todos os outros parâmetros enviados por um cliente são ignorados como forma de garantir a segurança e a integridade da api, evitando também recebimento de parâmetros inadequados e criados com objetivo malicioso [5].

- **Migrations**

  Migrations é a ferramenta criada pelo Rails para administrar e gerar mudanças na estrutura do banco de dados relacional. As *migrations* são arquivos que declaram qualquer tipo de mudança no banco seguindo o formato de uma linha do tempo. Os arquivos de migração são criados pelo *Rails* com um identificador único no formato timestamp com a data de criação da mudança. O comando *rake db:migrate* pode ser usado no terminal do Rails para executar em ordem cronológica as mudanças descritas nos arquivos [5].

  As ações desses arquivos envolvem qualquer mudança existente como criar, deletar, editar e excluir o banco de dados ou qualquer tabela do banco, assim como declarar os campos de cada tabela e definir qualquer atributo de campo como index, estado de unicidade, chave primária e secundária [23].

  A *migration* é a principal ferramenta oferecida pelo *Rails* para garantir que mudanças feita no banco de dados tanto em produção quanto em desenvolvimento e teste possam ser feitas com segurança sem o risco de prejudicar a estrutura do banco ou os dados existentes em um banco já em produção [5, 23].

- **Seeds**

Seeds é a ferramenta do Rails criada para permitir a inicialização de valores no banco de dados, o arquivo *seeds.rb* escrito na linguagem ruby é usado para inicializar valores *default* ou dados estáticos de tabelas do banco como Ids que definem algum tipo de estrutura no banco de dados. Para inicializar o banco de dados com essas informações basta chamar o comando *rake db:seed* no console do Rails e automaticamente todas as ações escritas no arquivo de seeds serão executadas no banco.

3.3 Arquitetura do Front-End

O Front-End é a parte da aplicação Web que envolve tudo que o usuário vê, incluindo o *design* e objetos de visualização, tais como o HTML e o CSS(*Cascade-Style-Sheet*).

O *Front-End* é o que define a parte do cliente da aplicação Web que consome os serviços do servidor Web, com o objetivo de se obter e apresentar dados da melhor forma para o usuário. Na construção e no planejamento de um sistema Web deve-se desenvolver o UX (*User Experience*) em conjunto com o UI (*User Interface*).

A experiência de usuário tem o trabalho de garantir que o uso do sistema seja o mais fácil e intuitivo possível para o usuário, um sistema com uma lógica complexa pode tornar-se inútil quando em combinação com uma experiência de usuário mal planejada.

Para o uso da lógica de programação no *Front-End* é utilizado arquivos em *Javascript*, a linguagem de programação oficial que todos os navegadores Web são programados para entender e interpretar.

No desenvolvimento do *Front-End* da aplicação foi utilizado o *framework* AngularJs, desenvolvido pela Google e que segue a estrutura de SPA (*Single Page Application*) e a arquitetura de projeto MVC (*Model View Controller*).

3.3.1 Single Page Application

Introdução

Uma SPA (*Single Page Application*) é um tipo de aplicação Web construída de maneira que todos os arquivos HTML, CSS e *javascript* necessários para renderizar o *layout* de qualquer página do sistema sejam recebidos pelo servidor, no momento que a página inicial foi requisitada pelo *browser* [19].

3.3.2 Funcionamento

O servidor executando o sistema SPA tem a habilidade de gerar URL para determinados recursos(outras páginas). Quando uma outra página da Web diferente da atual é requisitada, o SPA ao invés de buscar todos os objetos da página no *Back-End*, busca apenas os arquivos necessários para renderizar aquela parte da aplicação solicitada do seu servidor local de *Front-End* ou em cache do *browser* [19].

Um site tradicional faz uma requisição no servidor, recebe e renderiza os arquivos HTML, css e *javascript* recebidos toda vez que o usuário faz alguma ação. Do ponto de vista do usuário isso resulta em uma pausa do uso enquanto o servidor reinicia a página e a renderiza de novo na tela. Para casos em que o servidor está ocupado, a página contêm

muito conteúdo, ou a conexão com a internet esta lenta esse processo pode demorar muito para toda ação atrapalhando completamente a experiência do usuário utilizando o sistema.

Assim o SPA não gasta tempo atualizando e renderizando toda página novamente, apenas a parte que será mudada é renderizada novamente com o novo conteúdo. Por essa característica da página nunca precisar ser atualizada durante sua operação o SPA recebeu seu nome de aplicação de página única [19].

Por o SPA responder como uma aplicação *desktop* o tempo de resposta é minimizado transferindo dados que geralmente ficam no servidor para o cliente rodando no *browser*. O SPA portanto contém dados e a lógica de negócio minima necessária para tomar decisões localmente e rapidamente sem ter que consultar o servidor *Back-End* para todas as ações. Logo o *Back-End* fica apenas com o trabalho de validar dados, autenticar o cliente e salvar dados permanentes relativos ao sistema [19].

Um SPA consegue notificar usuários sobre o estado de uma ação, quando o SPA esta esperando uma ação ele pode dinamicamente renderizar uma barra de progresso enquanto em um website normal o usuário deve praticamente adivinhar se a página esta carregando ou com erro durante os processos de pedido de uma nova página, recebimento de objetos e renderização da pagina novamente [19].

O servidor Web feito em Rails com esse sistema não transfere arquivos *javascript*, css, HTML e nem tem o trabalho de renderizar esses objetos. Com o uso do SPA o servidor Rails tem apenas que retornar respostas no formato JSON com os recursos e resultados das operações de autenticação e acesso ao banco de dados.

Com esse modelo o servidor Rails se torna muito mais rápido e fica menos sobrecarregado com o aumento do uso de usuários pois a quantidade de dados retornados nas requisições feitas retornam uma quantidade de dados muito menor evitando o reenvio desnecessário de páginas.

Por último tem-se que o código da aplicação de SPA fica em um projeto diferente do código da aplicação do servidor *Back-End* e portanto há uma separação lógica entre a lógica de negócios do servidor e a lógica relacionada com o *Front-End* e com a experiência de usuário.

3.3.3 Desvantagens do Uso de SPA

A principal desvantagem é que como os dados necessários para renderizar todas as páginas são recebidos e colocados em cache local tem-se que o tempo de carregamento inicial necessário quando a aplicação é aberta pela primeira vez é maior do que o de um outro *website* que não usa essa estrutura [19].

Como a aplicação SPA atua como um cliente separado do servidor *Back-end* no momento de configurar a estrutura do servidor na maquina alvo deve então ser configurado dois servidores diferentes, um executando a aplicação SPA e o outro executando o servidor *Back-End* com o acesso ao banco de dados da aplicação [19].

O algoritmo de indexação de páginas no google não consegue identificar as sub-páginas construídas a partir de um SPA atrapalhando o rankeamento de sites no estilo SPA de serem encontrados pela busca do google. Esse não é um problema no projeto visto que é um sistema de uso fechado em que apenas responsáveis do departamento de geologia terão acesso.

Motivação

Contudo, as principais motivações para a escolha de uma estrutura de sistema Web em SPA neste projeto são:

- Aumento da performance da aplicação;
- Separação lógica do código e funcionalidades relacionadas ao *Front-End* do *Back-End*;
- Melhorar a experiência do usuário usando o sistema;
- Utilização de *frameworks open-source* em SPA no padrão MVC que auxiliam a construção de uma aplicação robusta.

3.3.4 NodeJs

O Front-End do projeto usa a ferramenta NodeJs como base do servidor da aplicação. O NodeJs é uma máquina virtual em *javascript* construída sobre a ferramenta Chrome's V8 Javascript. Ele é orientado a eventos, com um modelo de não bloqueio de entrada/saída que o torna leve e eficiente [2].

O NPM (*Node Package Manager*) é um serviço gerenciador de dependência de *javascript* que possui um grande repositório de bibliotecas que trabalham sobre o NodeJs, sendo um dos maiores ecossistemas de material *open-source* do Mundo competindo em número, até mesmo com as RubyGems da linguagem ruby [2].

Como uma máquina virtual voltada a eventos assíncronos, temos que o Node é perfeito para aplicações escaláveis. Ele foi construído para lidar com varias conexões concorrentes, sem perda considerável de desempenho e tem a característica de entrar em repouso quando nenhum trabalho esta sendo realizado pelo servidor [25].

A execução do *javascript* no NodeJs é feita em apenas uma thread, nesse contexto operações concorrentes se referem a capacidade do loop de eventos de executar funções de *callback* depois que um trabalho é terminado. Qualquer código concorrente executando operações de entrada e saída não podem bloquear o loop de eventos de execução do *Javascript* [25].

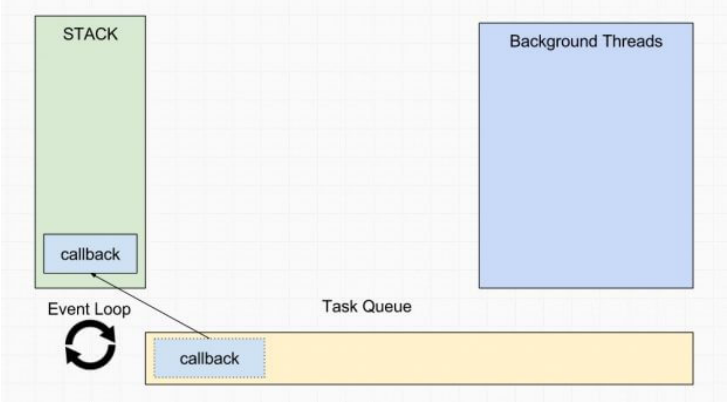


Figura 3.4: *Loop* de eventos do NodeJs.

O *loop* de eventos como pode ser visto na Figura 3.4 é o que mantem o NodeJs com um estrutura baseada em eventos. O *Event driven* é um fluxo de controle determinado por eventos ou alterações de estado, a maioria das implementações possuem um core (central) que escuta todos os eventos e chama seus respectivos *callbacks* quando eles são lançados (ou têm seu estado alterado) [25].

Ações como I/O são enviadas para serem executadas em outra *thread*, permitindo que a maquina do NodeJs siga trabalhando e a *stack* siga executando as próximas funções. Essas funções enviadas para que sejam executadas em outra *thread* precisam de um *callback*. Um *callback* é basicamente uma função que será executada quando a função principal terminar [25].

Esses *callbacks* podem ter responsabilidades diversas, como por exemplo, chamar outras funções e executar alguma lógica.

Como o V8 é single *thread* e só existe uma pilha de execução, os *callbacks* precisam esperar a sua vez de serem chamados. Enquanto esperam, eles ficam em um lugar chamado *task queue*, ou fila de tarefas. Sempre que a thread principal finaliza uma tarefa, a pilha fica vazia e uma nova tarefa é adicionada para posteriormente ser executada [2].

Em resumo, o *loop* de eventos é o responsável por inserir *callbacks* de funções na fila de tarefas e, posteriormente, transferir esse *callback* para pilha de execução de maneira que ele seja executado posteriormente [2].

3.3.5 AngularJs

Angular Js é um *framework javascript open-source* desenvolvido e mantido pela google que tem o objetivo de facilitar o desenvolvimento de sistemas webs principalmente construídos sobre a estrutura de single page applications.

O Angular Js apresenta uma estrutura própria de desenvolvimento que segue o padrão MVC apresentada na seção teórica desse trabalho. O objetivo ao utilizar a estrutura MVC ao invés de liberar que o programador construa sua própria estrutura é garantir que sistemas desenvolvidos em AngularJs possam crescer e se tornar complexos estimulando que o sistema continue bem construído, de fácil entendimento e manutenibilidade.

Esse *framework* tem como base a utilização de *templates HTML* para renderização de paginas seguindo o formato padrão da Web. O Angular Js permite que a sintaxe do HTML se estenda de forma que componentes apresentem características que tornem a página dinâmica. Basicamente o angular tem o trabalho de relacionar funções implementadas em javascript com os elementos renderizados na tela.

O Angular apresenta as ferramentas de *data binding* e injeção de dependência que permitem diminuir a quantidade de código escrito, tornando a aplicação muito mais sucinta e limpa. Todo esse processamento é executado pelo *browser*, tornando-o uma ferramenta ideal para trabalho com tecnologias de servidor.

Em conjunto com o Angular Js é utilizado nesse projeto o servidor Node Js para execução do código javascript do servidor *Front-End*. Como descrito na seção 4.5.5 utilizaremos o Node com objetivo de manter o desempenho da aplicação ate mesmo para grandes quantidades de requisições no servidor. O Angular possui o serviço $http que permite realizar facilmente requisições http de qualquer tipo em nosso servidor *Back-End* facilitando a integração entre as duas partes do sistema.

Data Binding

O *data binding* no contexto do angular consiste em relacionar uma variável em *javascript* com um elemento em HTML, essa ligação torna o dado apresentado na tela do usuário dinâmico no sentido em que qualquer alteração na variável javascript é instantaneamente atualizada na tela do usuário. O mesmo processo acontece para o caso em que o elemento do HTML é alterado, essa alteração é automaticamente passada para a variável *javascript* do código em execução. [4]

Esse processo facilita o desenvolvimento pois diminui a quantidade de código necessária para alteração dos dados no código *javascript* quando uma alteração durante a execução do sistema é feita pelo usuário. Um exemplo clássico é a marcação de um *checkbox* na página, em um sistema utilizando apenas ferramentas clássicas da web, um código utilizando uma ferramenta como JQUERY precisaria ser criado para buscar o identificador do elemento de *checkbox* no html e implementar um *watcher* associado a essa variável para receber informações da alteração do estado desse elemento. [14]

O AngularJs realiza esse processo para o programador apenas adicionando na escrita do elemento um texto escrito *ng-model= nomeDaVariavel* dentro das chaves do objeto. Seguindo a sintaxe do HTML 5 de um objeto do tipo checkbox temos que a escrita se torna *<input type="checkbox"ng-model="nomeDaVariavel»*. Esse tipo de sintaxe do Angular que permite ser escrita da mesma forma como a propriedade de um elemento HTML é o que chamamos de diretivas. [4]

Com o auxilio dessa ferramenta do Angular podemos facilmente receber e mostrar dados de formulários que são vastamente utilizados na aplicação. Apenas com o auxílio das propriedades do ng-model podemos construir objetos que serão posteriormente enviados no formato JSON para o Back-End da aplicação inserindo, atualizando, deletando ou recebendo objetos do banco de dados central. [4]

Injeção de depêndencia

A Injeção de dependência é um padrão de desenvolvimento de software que lida com a questão de como componentes em uma arquitetura orientada a objeto cuidam da suas dependências [14]. O AngularJs adota esse padrão e o implementa por meio de um subsistema próprio que cria componentes, resolve dependências e provê essas dependências a componentes que façam o pedido do uso do serviço [4].

Todos os elementos do AngularJs obrigatoriamente usam esse sistema de injeção de dependências para seu funcionamento o que convêm com o principio do Angular de obrigar os programadores que utilizam a ferramenta a seguirem o padrão MVC definido [4].

3.3.6 Componentes do AngularJs

Nesta seção são descritos os principais componentes do AngularJs utilizados no projeto.

- **Scope:**

  Scope é o objeto do Angular responsável por implementar a camada de modelo da aplicação no contexto de uma Página Web. Apenas por meio dele que conseguimos ter acesso a variáveis de *controllers* no DOM(*Document Object Model*) da aplicação,

sendo assim variáveis que são manipuladas em tela devem ser obrigatoriamente inseridas dentro desse objeto [4].

O Scope é implementado como um serviço do AngularJs e deve, consequentemente ser injetado no *controller* pelo submodulo de injeção de dependências. Cada *controller* deve apresentar seu próprio *scope* sendo que esses *scopes* não devem comunicar entre si diretamente, apenas por meio de serviços [4].

O scope também apresenta funções importantes que auxiliam a manipulação dos modelos da aplicação como:

- A função *watch* que observa mudanças em variáveis do controller pelo DOM e chama um callback quando certa variável é modificada;
- A função *apply* usada para propagar e atualizar o DOM para qualquer mudança realizada no objeto scope.

Por último, vale lembrar que a aplicação do Angular como todo possui um elemento chamado de *RootScope*, ele é o objeto pai de todos os outros *scopes* da aplicação podendo ser acessado por todos os controladores da aplicação. O *RootScope* não deve ser usado para transferir dados entre *controllers* diferentes. Para manter a segurança e a performance da aplicação devem ser usados serviços para fazer a transferência de dados entre telas [4].

- **Diretivas:**

  Diretivas são marcadores de elementos DOM que podem ser usados como atributos de uma classe CSS, um nome de elemento ou como atributo de um elemento HTML. As diretivas permitem adicionar comportamentos ou transformar elementos estáticos de um documento HTML [4].

  O *framework* AngularJs vem com um conjunto de diretivas implementadas, como *ng-bind*, *ng-Model* e *ng-Class* que auxiliam o desenvolvimento do sistema e o tornam, inicialmente, uma ferramenta robusta. Da mesma maneira como *controllers* e serviços, o AngularJs permite a criação de suas próprias diretivas dando ao poder do programador do projeto e a contribuidores da comunidade *open-source* a estenderam ainda mais o domínio e as capabilidades do framework [4].

  Quando o AngularJs inicia a aplicação ele identifica essas diretivas no documento HTML e relaciona o código javascript da lógica correspondente implementada com os ids que identificam o elemento no documento.

```
<input ng-model="foo">
```

Figura 3.5: Diretiva do tipo propriedade.

Os três tipos de declaração de diretivas podem ser vistos no exemplo das Figuras 3.5, 3.6 e 3.7, na Figura 3.5 tem-se uma diretiva que atua como um objeto HTML que possui lógica própria, e renderiza outros elementos em HTML, esse modelo permite

tornar o código modular e reutilizável estimulando o reaproveitamento do código e evitando repetição desnecessária.

```
<span my-dir="exp"></span>
```

Figura 3.6: Diretiva do tipo classe.

A Figura 3.6 representa o tipo de sintaxe de diretiva geralmente utilizada para adicionar propriedades de CSS a um determinado elemento.

```
<person>{{name}}</person>
```

Figura 3.7: Diretiva do tipo elemento.

A Figura 3.7 é a sintaxe mais comum, usada quando se deseja adicionar alguma lógica nova a um elemento HTML, geralmente, sem renderizar um novo elemento do DOM. Os exemplos clássicos de diretivas como *ng-Model*, *ng-Class* e *ng-Bind* seguem esse formato.

- **Module:**

  Módulo é o componente pai de todas as estruturas de uma aplicação desenvolvida utilizando AngularJs. Ele trabalha como um *container* que integra diferentes partes da aplicação como *controllers*, serviços, filtros, diretivas e módulos filhos [4].

  Aplicações em AngularJs não possuem uma função *main*, o módulo declarativamente especifica como a aplicação deve ser inicializada. O Angular utiliza esse método por obter algumas vantagens como:

  – O processo de declaração é fácil de entender;
  – Reuso de módulos em outros módulos como um pacote de código que se comporta similar a uma biblioteca;
  – Permitir que módulos da aplicação possam ser carregados em qualquer ordem;
  – Permitir que testes unitários apenas carreguem os módulos necessárias para execução de cada teste, acelerando o processo.

  A diretiva *ng-App* é usada pela aplicação para relacionar um documento HTML com um módulo assim como é feito com *controllers*, sendo assim para encontrar o ponto inicial da aplicação basta procurarmos em que arquivo a diretiva foi chamada [4].

  O Módulo pai deve declarar e receber por injeção de dependência os módulos filhos da aplicação, serviços, arquivos de tradução e configuração, filtros e diretivas. Em resumo o módulo é o elemento que permite que o AngularJs siga uma estrutura bem definida para projetos de grande e pequeno porte conectando todos os elementos da aplicação [4].

- **Controllers:**

  No AngularJs um *controller* é definido como uma função de *javascript* construtora que recebe por injeção de dependência o serviço Scope do AngularJs, um *controller* pode ser conectado a um objeto em HTML por meio da diretiva *ng-controller*. O Angular Js por meio dessa diretiva relaciona os elementos do arquivo com um dos *controllers* da aplicação. Um novo Scope é criado e injetado no *controller* de forma que as variáveis utilizadas por *ng-model* se conectam com as variáveis em *javascript* do *controller* correspondente [4]. Os *controllers* tem aS funções de:

  - Iniciar o estado de um objeto *scope*;
  - Adicionar lógicas e funções ao objeto *scope* permitindo que lógicas de negócio podem ser chamadas a partir do DOM da página alvo;
  - Executar a lógica de negócio da página alvo, chamando serviços que realizam por exemplo requisições HTTP em um servidor alvo.

  Para seguir o padrão imposto pelo Angular e garantir a manutenibilidade da aplicação os *controllers* não devem ser usados para:

  - Manipular objetos HTML, o *controller* deve apenas tratar da lógica de negócio da aplicação;
  - Formatar entrada de campos. Para isso existe o serviço do angular de form control;
  - Formatar saída de campos. Para isso existe o serviço filter do AngularJs;
  - Reusar codigo entre vários controladores, para isso deve ser criado um serviço ou diretiva do angular.

Capítulo 4

Desenvolvimento do Sistema

Neste capitulo são apresentadas a estrutura e todo o processo de desenvolvimento do sistema, utilizando as tecnologias apresentadas no cápitulo anterior. Além disso, são aparesentados os resultados obtidos durante cada uma das etapas de desenvolvimento, e as figuras referentes aos *layouts* construídos no *Front-End*.

4.1 Estrutura do Sistema

A *Stack* é um termo comum no desenvolvimento Web definido como a combinação de ferramentas complementares utilizadas em conjunto para a construção de um sistema. Nesta seção é feita uma breve descrição de como as ferramentas escolhidas operam entre si e compõem a estrutura do projeto.

Conectando todas as ferramentas descritas durante o Capitulo 3 tem-se na Figura 4.1 uma estrutura simplificada do funcionamento do projeto, seguindo o fluxo completo de busca de um dado no *Back-End*, até a etapa de renderização da tela com o dado obtido.

O *Back-End* implementado pela ferramenta Ruby on Rails é responsável por implementar a API que recebe as requisições HTTP do *Front-End* da aplicação e do aplicativo RockDroid que execução em um Hardware com sistema operacional Android.

O *Ruby on Rails* é configurado para acessar e realizar consultas sobre o banco de dados PostgreSQL. As requisições HTTP, seguindo a estrutura REST, são roteadas pelo Rails para funções que realizam a ação alvo da requisição.

A construção do banco de dados em formato de linha do tempo é feita pela ferramenta migration do Rails que realiza ações de criar, deletar, atualizar linhas e colunas de tabelas no modelo de banco de dados relacional, além de realizar operações como definir chaves primárias secundárias e propriedades dos campos de uma determinada tabela. Os relacionamentos entre as tabelas é feito utilizando a ferramenta *ActiveModel* do Rails, que permite definir relacionamentos entre tabelas do tipo N-N, 1-1, N-1, etc.

O Rails faz o trabalho de traduzir código em Ruby para consultas de busca, inserção, atualização e remoção no banco de dados com o auxílio da ferramenta Active Model do Rails. O AngularJs implementa toda a parte visual do projeto, com o modelo de aplicação de página única enviando requisições HTTP ao servidor Rails apenas para obter dados, realizar ações no banco de dados da aplicação, e realizar o processo de autenticação do sistema. Objetos HTML fazem parte do servidor executando o Angular e não precisam ser buscados da API em Rails.

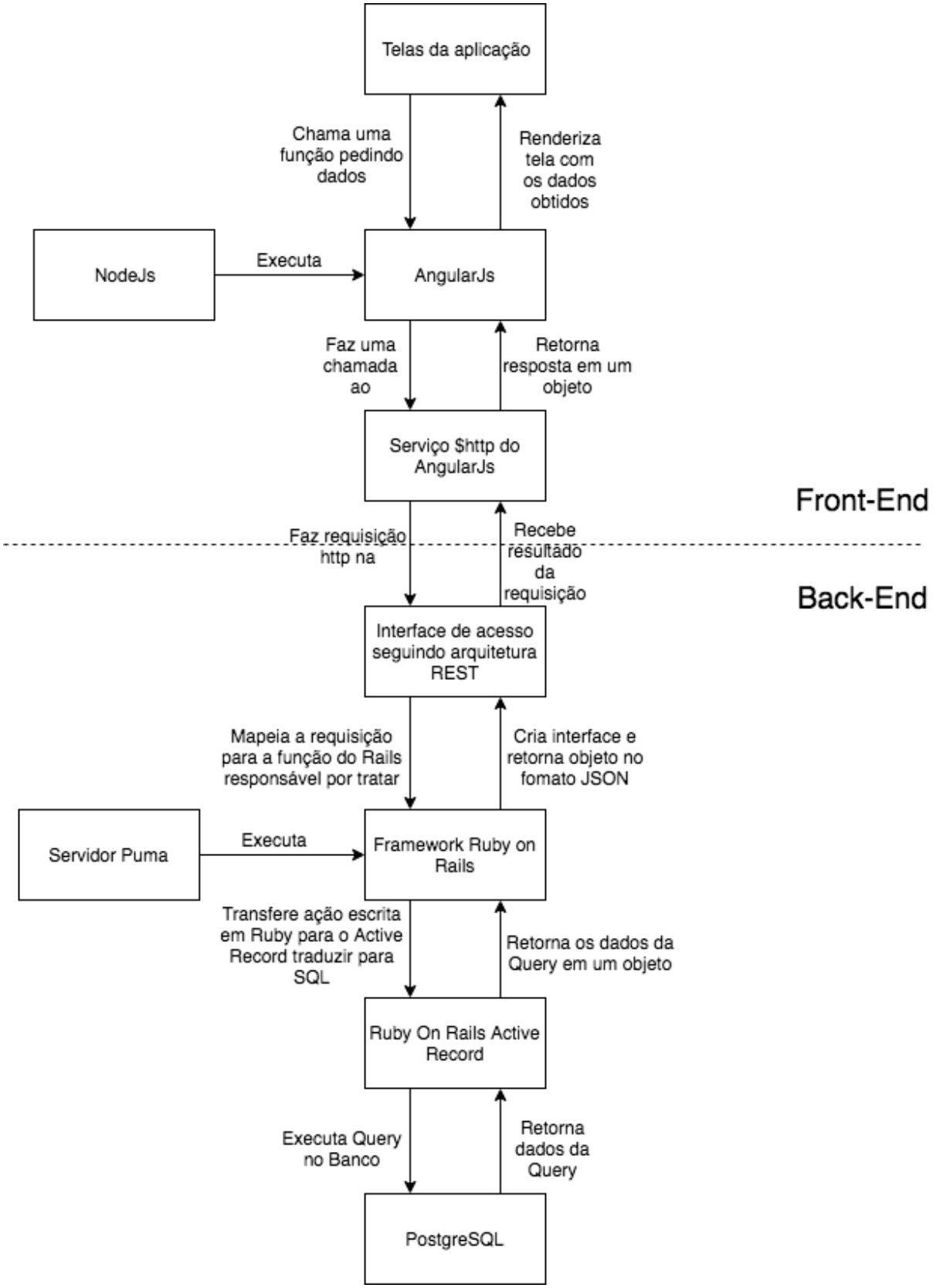


Figura 4.1: Funcionamento simplificado do sistema.

A aplicação *Front-End* em Angular opera sobre o servidor NodeJs enquanto a aplicação Rails opera sobre o servidor Ruby Puma. As duas aplicações executam na mesma máquina e conversam entre sí por meio de uma porta de acesso na maquina hospedeira. A porta padrão de acesso ao servidor Rails é a 3000 enquanto a do servidor NodeJs é 8000.

Por ultimo, tem-se que todas as requisições HTTP realizadas pelo AngularJs com o auxilio do serviço $HTTP seguem a arquitetura REST e utilizam como modelo de dados padrão o formato JSON, o que significa basicamente que as duas partes do sistema são configuradas para conversarem entre se e entenderem esse formato de dados.

4.2 Back-End

Nesta seção é descrito o processo de construção do banco de dados e da criação da API de acesso aos recursos do *Back-End* por meio da ferramenta *Ruby On Rails*.

4.2.1 Gerando o Banco com o Rails

O primeiro passo para construção do *Back-End* após a instalação da linguagem Ruby e do *framework Ruby on Rails* é a configuração do arquivo do arquivo database.yml. Nesse arquivo devem ser inseridos qual o tipo de banco de dados que será acessado, a porta de acesso, o número máximo de *threads* de execução e o usuário e senha para acesso nesse banco. O banco escolhido foi o PostgreSQL e para seu funcionamento no Rails foi instalado a gema *pg* que faz o trabalho de configurar a classe ActiveRecord do Rails para realizar todas as ações nesse banco.

Após configuração inicial do arquivo, foi executado no terminal do Ruby on Rails o comando *rake db:create*, esse comando cria o banco de dados a partir das configurações descritas no arquivo *database.yml*.

Depois que o banco de dados foi criado com sucesso, é inicializado o processo de criação das tabelas do banco a partir da geração dos arquivos de migração pelo comando no terminal do Rails que segue o padrão *rails g migration nomeDaAção*. Como base para a criação das tabela é usado o modelo de dados contido na Figura 4.2 que corresponde ao mesmo modelo de dados implementado no banco de dados local gerado pelo aplicativo *mobile*.

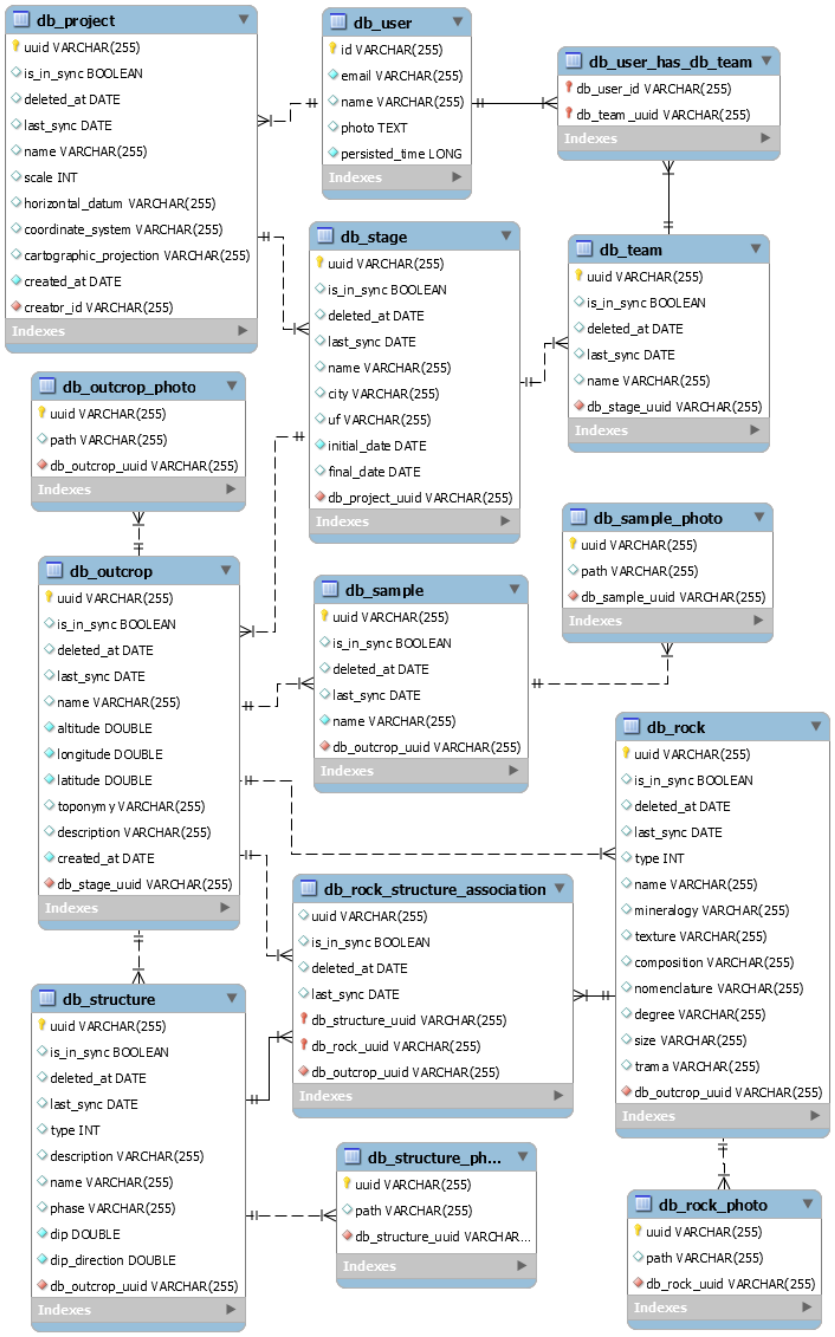


Figura 4.2: Modelo relacional do banco de dados.

Um arquivo de migração é criado para cada tabela do banco de dados. Na Figura 4.3 pode-se ver um exemplo de código contido em uma *migration* para a criação de uma tabela.

```ruby
class CreateProjects < ActiveRecord::Migration[5.0]
  def change
    create_table :projects, id: false do |t|
      t.string :uuid, unique: true, index: true, null:false
      t.string :name
      t.string :creator_id
      t.string :user_id
      t.integer :deleted_at, :limit => 8
      t.float :persisted_time, null: false

      t.timestamps
    end
  end
end
```

Figura 4.3: Exemplo de criação de tabela.

A Figura 4.4 contém a estrutura de linha do tempo formada pelos arquivos de migrações. No título de cada arquivo é possível visualizar os *timestamps* criados a partir do momento em que o arquivo foi gerado.

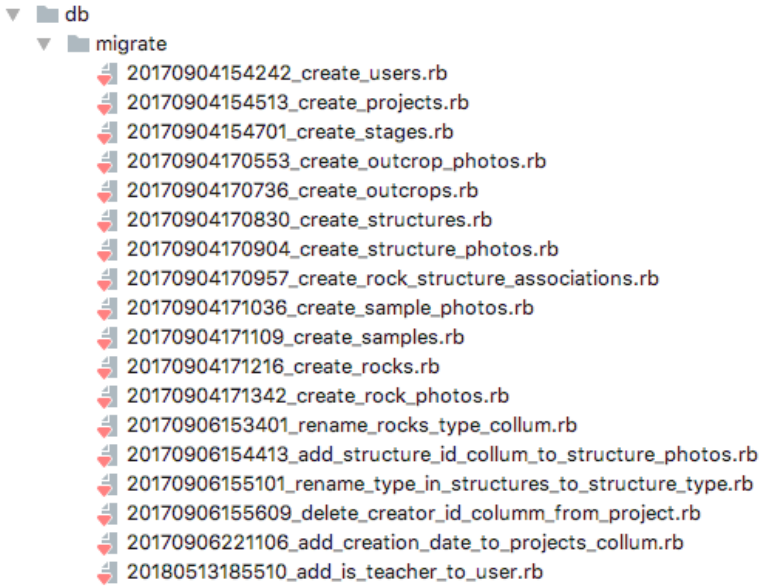


Figura 4.4: Lista de *Migrations*.

Com o fim da criação de todos os arquivos de migração, é necessário utilizar o comando *rake db:migrate* no terminal do Rails para executar todos os arquivos na ordem definida pelas *timestamps*. Após o fim da execução é possível abrir e ver toda a estrutura do banco criada com o auxílio de uma ferramenta de visualização como o PgAdmin.

Para finalizar a construção do banco, a segunda etapa de desenvolvimento que consistiu em definir pelo Rails o relacionamento entre as tabelas do banco de dados. Para cada tabela deve ser criado um arquivo na pasta *model do framework*, esse arquivo deve conter uma classe que estende a classe ApplicationRecord do Ruby on Rails. Na Figura 4.5 tem-se o exemplo para a tabela *outcrop* que possui relacionamento de 1-N com mais 6 tabelas do banco é ilustrado.

```ruby
class Outcrop < ApplicationRecord
  self.primary_key = 'uuid'
  belongs_to :stage
  has_many :outcrop_photos, dependent: :destroy
  has_many :structures, dependent: :destroy
  has_many :rocks, dependent: :destroy
  has_many :samples, dependent: :destroy
  has_many :rock_structure_associations, dependent: :destroy
  validates_presence_of :name
  validates_presence_of :latitude
  validates_presence_of :longitude
  validates_presence_of :stage_id

  before_create do
    self.persisted_time = DateTime.now.strftime('%Q')
  end

  before_create :generate_token

  protected

  def generate_token
    self.uuid = loop do
      random_token = SecureRandom.urlsafe_base64( n nil, padding false)
      break random_token unless Outcrop.exists?(uuid: random_token)
    end
  end
end
```

Figura 4.5: Exemplo de configuração de modelo.

Após criar o model para todas as classes foi gerado um diagrama com todos os *models* e relacionamentos do banco, que pode ser visto na Figura 4.6.

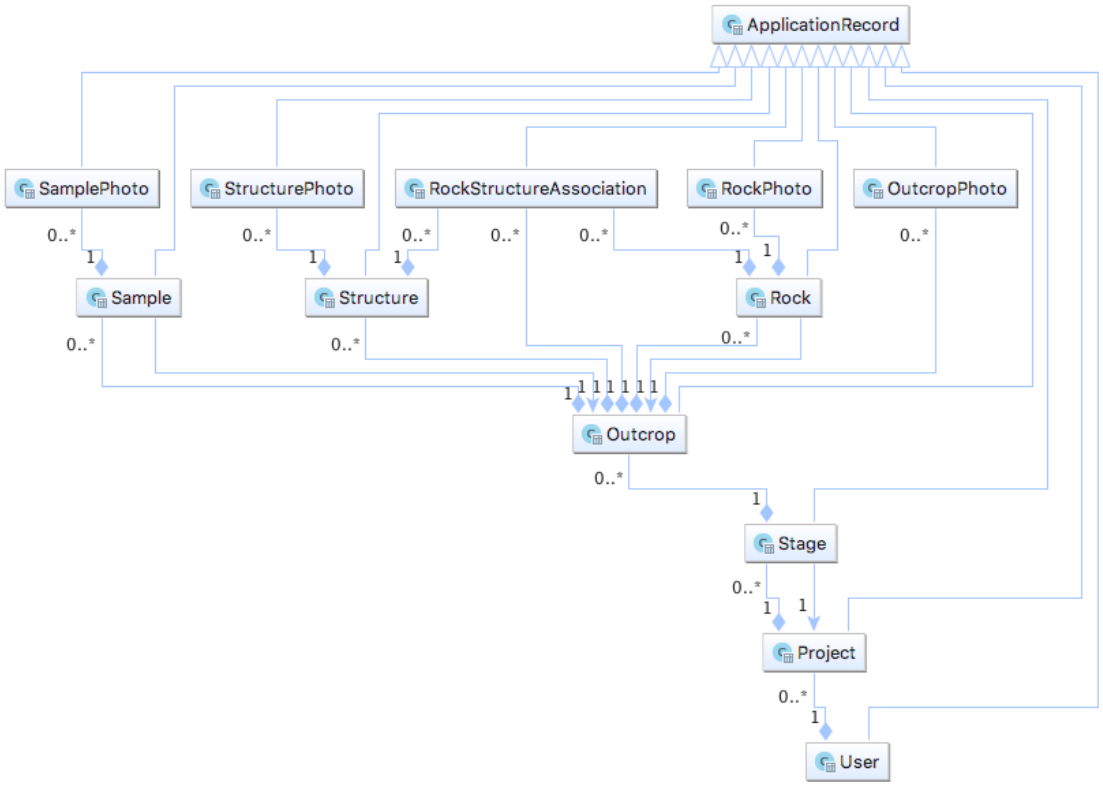


Figura 4.6: Diagrama de relacionamentos do banco.

Agora que todas as classes estão relacionadas, o acesso a elementos com relacionamento pode ser feito facilmente com as funções de acesso do Ruby on Rails. Por exemplo, para fazer uma consulta no banco, buscando em forma de um objeto, todos os elementos contidos na tabela de *structures* que pertencem a um determinado outcrop e que contêm um id definido pelo parâmetro uuid, basta chamar a linha de código '@strucutres = Outcrop.find(params[:uuid]).structures'. A variável @strucutres receberá como resultado um objeto contendo a lista de estruturas correspondentes.

4.2.2 Gerando Rotas de Acesso a API

Agora que o banco está completo inicia-se o processo da criação das rotas de acesso seguindo o modelo REST definido no Capitulo 2. O arquivo *routes.rb* do Ruby on Rails é o local em que por convenção, devem ser definidos todas as rotas do sistema. Na Figura 4.7 estão contidas a definições de algumas da principais rotas de acesso da API criada.

```
Rails.application.routes.draw do
  mount RailsAdmin::Engine => '/admin', as: 'rails_admin'
  resources :rock_photos
  resources :rocks, except: [:show]
  resources :samples, except: [:show]
  resources :sample_photos, except: [:show]
  resources :structure_photos, except: [:show]
  resources :structures, except: [:show]
  resources :outcrops, except: [:show]
  resources :outcrop_photos, except: [:show]
  resources :stages, except: [:show]
  resources :rock_structure_associations, except: [:show]
  resources :projects, except: [:show]
  resources :users, except: [:show]

  delete 'projects', controller: 'projects', action: 'destroy'
  delete 'rocks', controller: 'rocks', action: 'destroy'
  delete 'samples', controller: 'samples', action: 'destroy'
  delete 'structures', controller: 'structures', action: 'destroy'
  delete 'stages', controller: 'stages', action: 'destroy'
  delete 'users', controller: 'users', action: 'destroy'
  delete 'outcrops', controller: 'outcrops', action: 'destroy'
  delete 'rock_structure_associations', controller: 'rock_structure_associations', action: 'destroy'


  delete 'outcrop_photos', controller: 'outcrop_photos', action: 'destroy'
  delete 'rock_photos', controller: 'rock_photos', action: 'destroy'
  delete 'sample_photos', controller: 'sample_photos', action: 'destroy'
  delete 'structure_photos', controller: 'structure_photos', action: 'destroy'

  get 'rock_photos/search/findById', controller: 'rock_photos', action: 'show'
  get 'rocks/search/findById', controller: 'rocks', action: 'show'
  get 'samples/search/findById', controller: 'samples', action: 'show'
  get 'sample_photos/search/findById', controller: 'sample_photos', action: 'show'
  get 'associations/search/findById', controller: 'rock_structure_associations', action: 'show'
  get 'structure_photos/search/findById', controller: 'structure_photos', action: 'show'
  get 'structures/search/findById', controller: 'structures', action: 'show'
  get 'outcrops/search/findById', controller: 'outcrops', action: 'show'
  get 'outcrop_photos/search/findById', controller: 'outcrop_photos', action: 'show'
  get 'stages/search/findById', controller: 'stages', action: 'show'
```

Figura 4.7: Rotas de acesso a API.

A definição de rotas segue um padrão definido pelo Rails, em que o primeiro termo é o tipo da requisição HTTP que deve ser feita, o segundo termo é uma *string* que indica

qual o endereço de acesso a aquela ação, lembrando que para algumas rotas o parâmetro de identificador único pode ser passado pelo endereço da requisição HTTP ao invés do corpo da requisição.

O parâmetro *controller* é o responsável por indicar o arquivo da pasta de *controllers* em que a função que contém o código que deve tratar e responder aquela requisição deverá ser chamada. O parâmetro *action* indica qual função desse determinado *controller* deve ser chamada quando essa rota é acessada.

O parâmetro *resource* usado no arquivo *routes.rb* do projeto é uma facilidade do Rails que permite gerar automaticamente rotas para as ações de criar, excluir, deletar, buscar um elemento ou uma lista de elementos de uma determinada tabela do banco.

Agora que as rotas de acesso estão definidas fez-se o trabalho de criar todos os *controllers* e funções definidas pelo arquivo de rotas. Como a API possui acesso a todos os modelos do banco de dados foi necessário a criação de um arquivo de *controller* para cada tabela, em todos os arquivos foram implementados as funções que buscam, removem, atualizam e adicionam elementos ao banco de dados.

Como convenção foi definido que em todo projeto os dados serão transferidos no formato JSON então todos os objetos que serão enviados na resposta da requisição são transformados para esse formato com o auxilio da função *render :json* do Ruby On Rails.

Na criação das rotas e *controllers* foi tomado o cuidado de definir os nomes das rotas e os parâmetros que devem ser enviados pela requisição HTTP de forma igual as requisições que estavam configuradas no aplicativo seguindo o modelo REST. Como a API e o aplicativo seguem o mesmo modelo a integração entre as duas partes consistiu apenas em definir no aplicativo a rota base das requisições com o endereço de IP de acesso ao servidor.

4.2.3 Painel de Administração

Agora que o banco e as rotas de acesso estão completas, foi instalado *a gema railsAdmin* que permite gerar um painel administrador com base nas tabelas e modelos construídos pelo Ruby on Rails. Esse painel de administração possui as propriedades de:

- Fazer qualquer ação CRUD(*Create, Read, Update, Delete*) nas tabelas do banco com facilidade;
- Adicionar ou remover elementos relacionados no banco;
- Fazer buscas de dados por meio de filtros;
- Exportar qualquer dado de qualquer tabela nos formatos CSV/JSON/XML;
- Permitir integração com o sistema de autenticação implementado;
- Validação dos dados inseridos em formulários com base nas validações implementadas na API do Ruby on Rails.

Após a execução do comando *rails g rails admin install* no terminal do Rails pode-se acessar por meio da rota */admin* o painel no servidor. A Figura 4.8 mostra o *dashboard* inicial renderizado quando se acessa a rota.

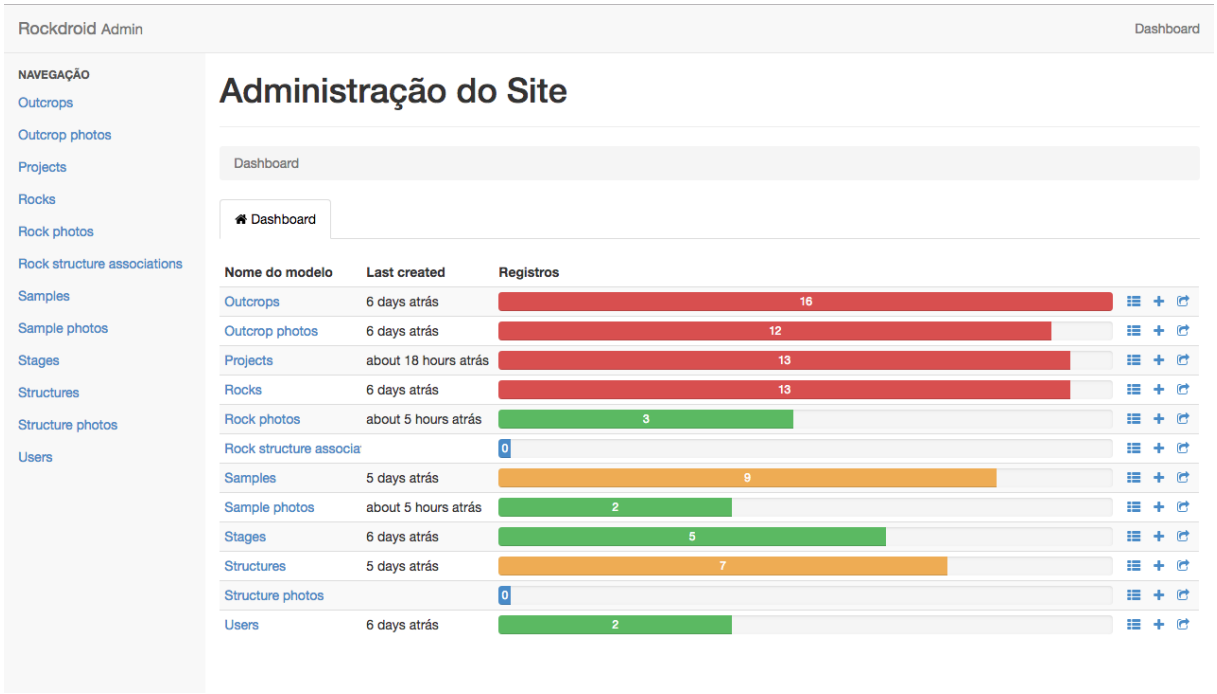


Figura 4.8: *Dashboard* inicial do painel.

O menu de navegação lateral permite acesso a qualquer uma das tabelas do banco de dados, quando um dos elementos é selecionado tem-se acesso a uma tela, como mostrado na Figura 4.9, que mostra todos os elementos da tabela de projetos dos geólogos com botões para realizar as ações de adicionar, remover, atualizar e deletar qualquer um dos elementos da lista.

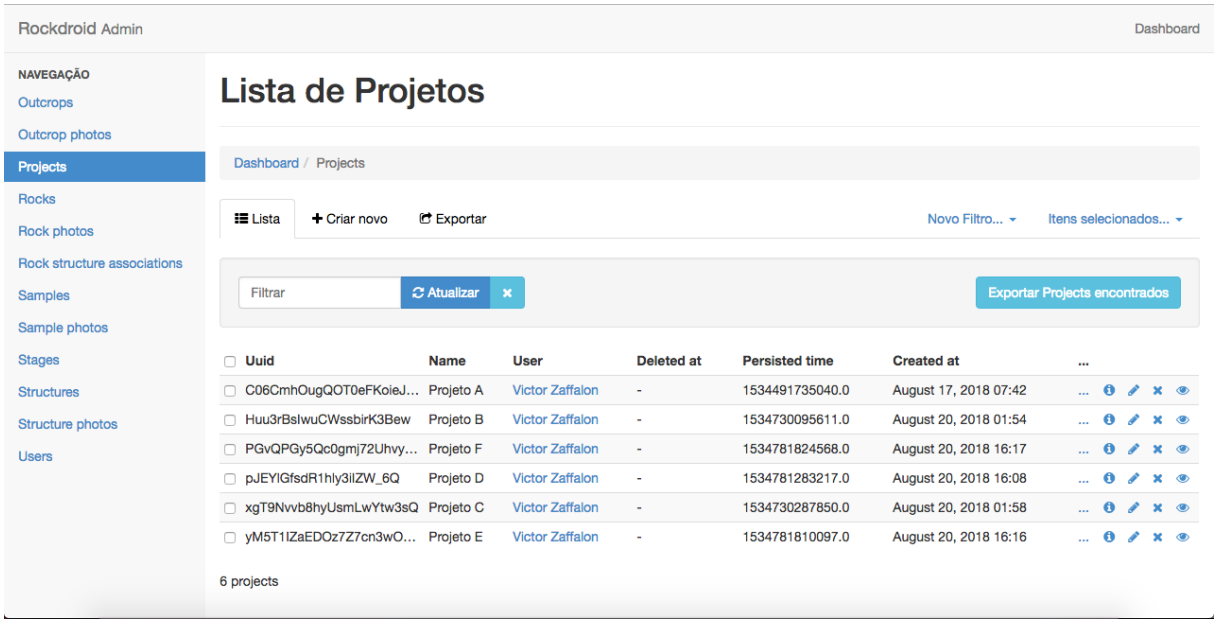


Figura 4.9: Tela de lista de elementos na tabela de projetos.

Na Figura 4.10 tem-se um exemplo de tela de criação de elemento no banco de dados, a figura representa um formulário de criação de projeto, sendo possível a configuração de

campos de relacionamento como adicionar e remover *stages* de um projeto, e decidir qual é o usuário dono do projeto.

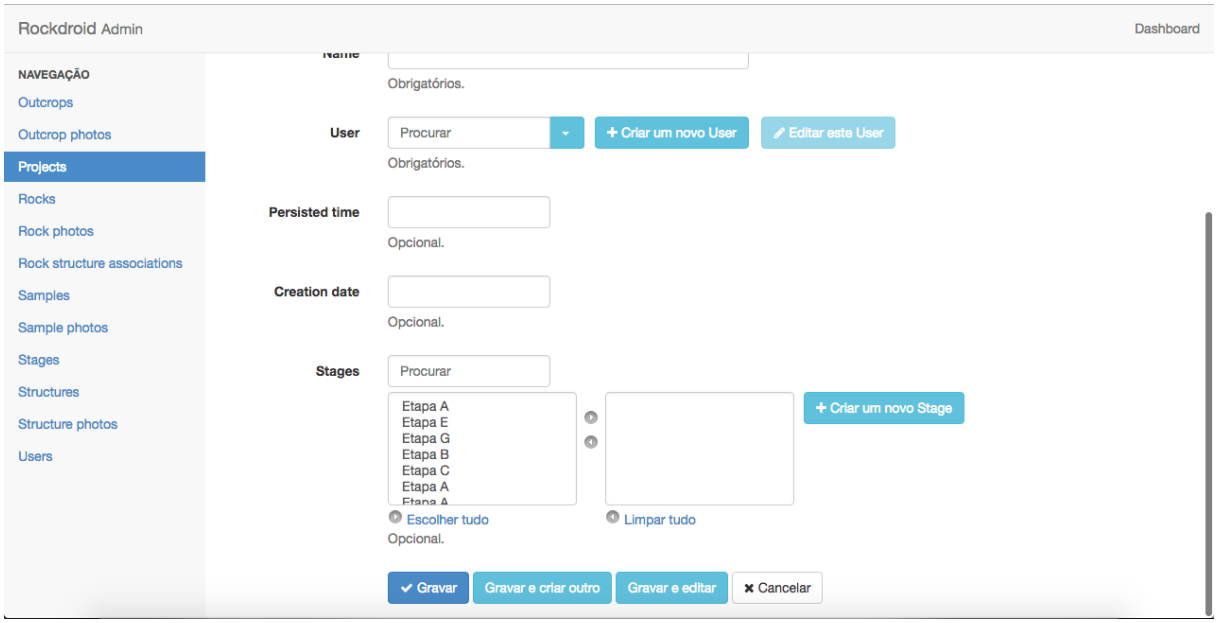


Figura 4.10: Tela de criação de projetos

Nas Figuras 4.11 e 4.12 são mostradas a aba da tabela que permite a exportação dos dados e de seus elementos relacionados. É possível selecionar apenas os campos que se deseja que apareçam no arquivo, além disso, o sistema permite selecionar o tipo de separador e codificação dos dados do arquivo, sendo o tipo padrão definido como (',') e (UTF-8). O arquivo exportado pode ser no formato JSON, CSV e XML.

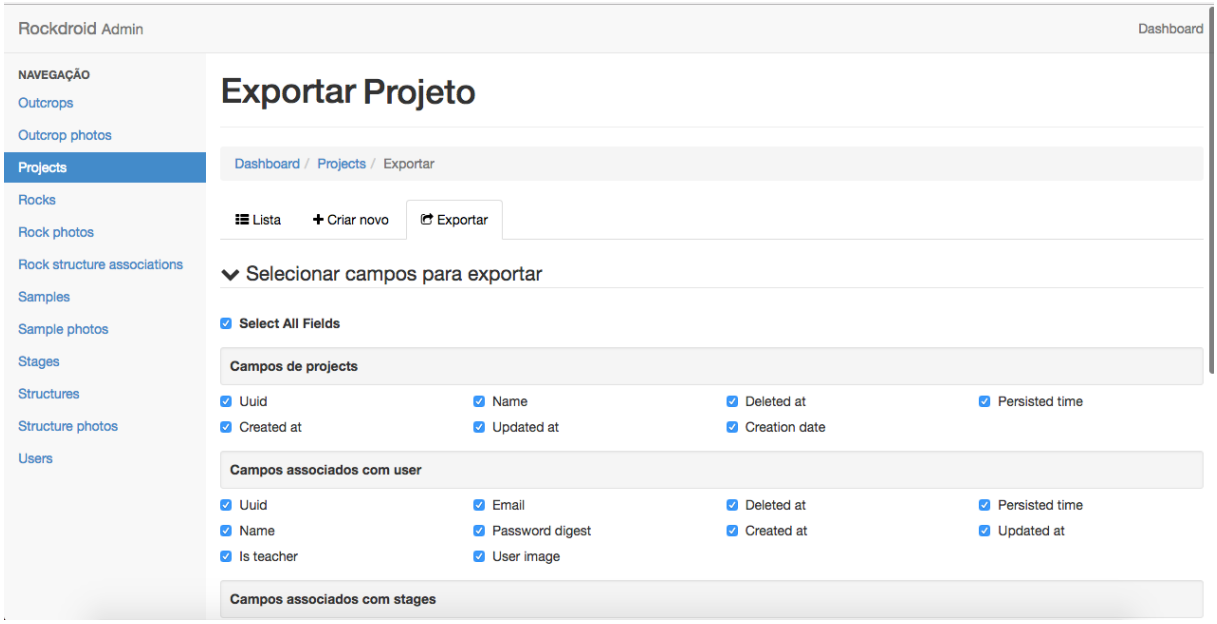


Figura 4.11: Exportando projeto.

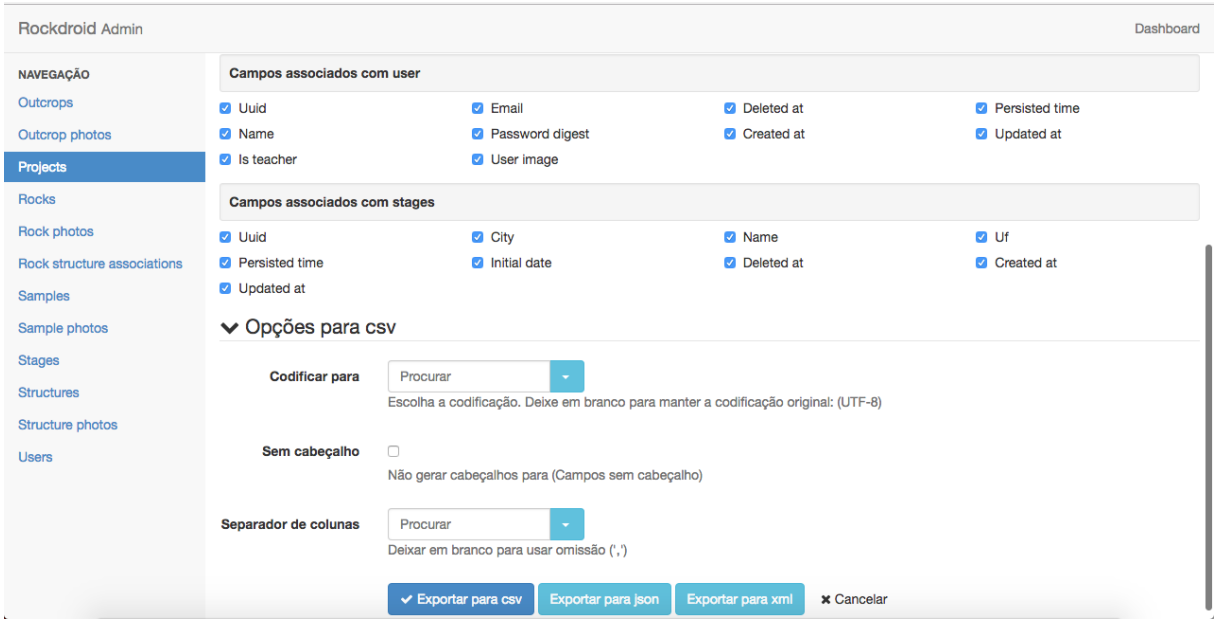


Figura 4.12: Seleção de formato de exportação.

4.3 Front-End

A etapa final do desenvolvimento da plataforma Web de Visualização dos dados consistiu na construção das telas da aplicação, sua integração com o *Back-End* da aplicação, e a verificação do funcionamento do sistema e dos formulários para cada ação de usuário possível. As funcionalidades desenvolvidas no sistema que serão apresentadas nesta seção são definidas por:

- Cadastro e login de usuário;
- Exibir um tutorial explicativo do funcionamento;
- Criar, editar, visualizar e excluir projetos, etapas, afloramentos, rochas, amostras e estruturas;
- Adicionar e remover fotos de afloramentos, rochas amostras e estruturas;
- Visualizar afloramentos cadastrados em um mapa;
- Fazer busca de afloramentos por latitude e longitude no mapa;
- Exibir um dashboard com informações dos dados obtidos como número de elementos criados, número de etapas por Unidades federativas, quantidade de afloramentos por altitude;
- Exportar projetos para o formato que possa ser executado no Excel e fazer download de fotos cadastradas.

4.3.1 Cadastro de Usuário e Login

Ao acessar o endereço do site, um código de decisão verifica se o usuário já está logado ou não no sistema, caso o usuário não esteja conectado, então ele é direcionado a tela de *login* (Figura 4.13).

Login

A tela de *login* constitui-se de um formuário em que email e senha do usuário devem ser preenchidos. Se o usuário ainda não possuir uma conta, ele pode criá-la por meio do proprio aplicativo ou acessando a opção registre-se da tela inicial. Ao clicar na opção (Registe-se) o aluno é redirecionado a tela de registro para criação de sua conta (Figura 4.13).

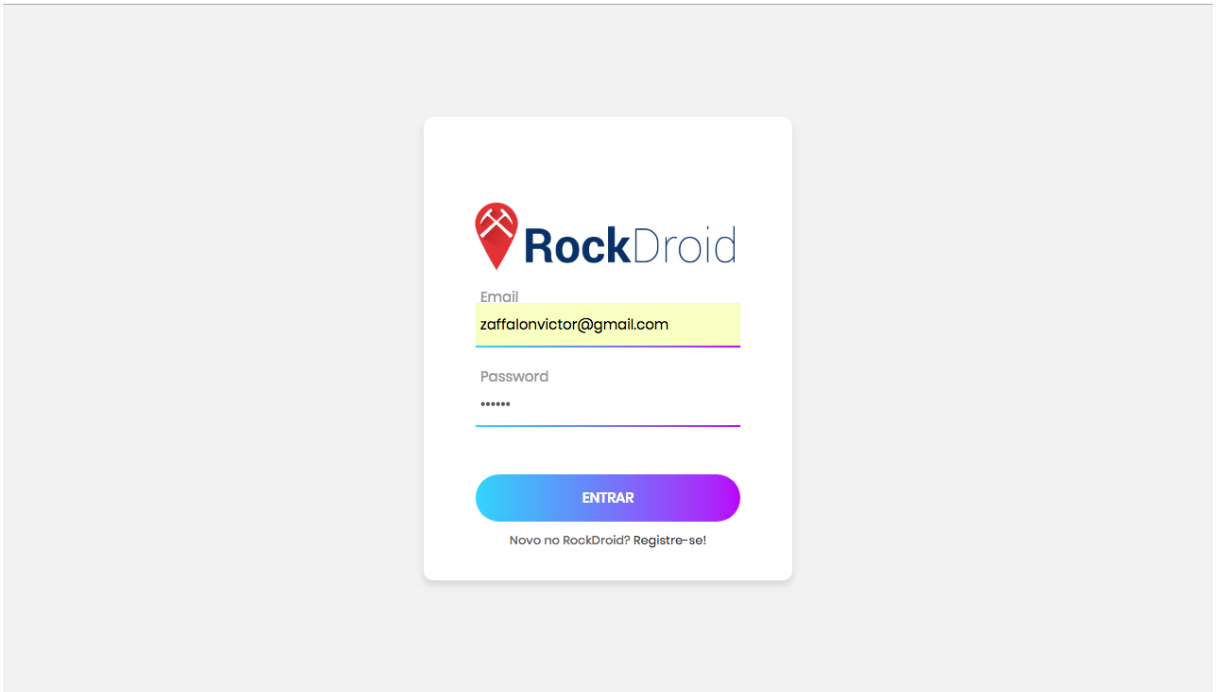


Figura 4.13: Tela de Login.

Registro

A tela de registro é formada por um formulário com os campos de nome, email, senha e confirmação de senha, seguindo o mesmo padrão do campo de registro do aplicativo Rockdroid (Figura 4.14).

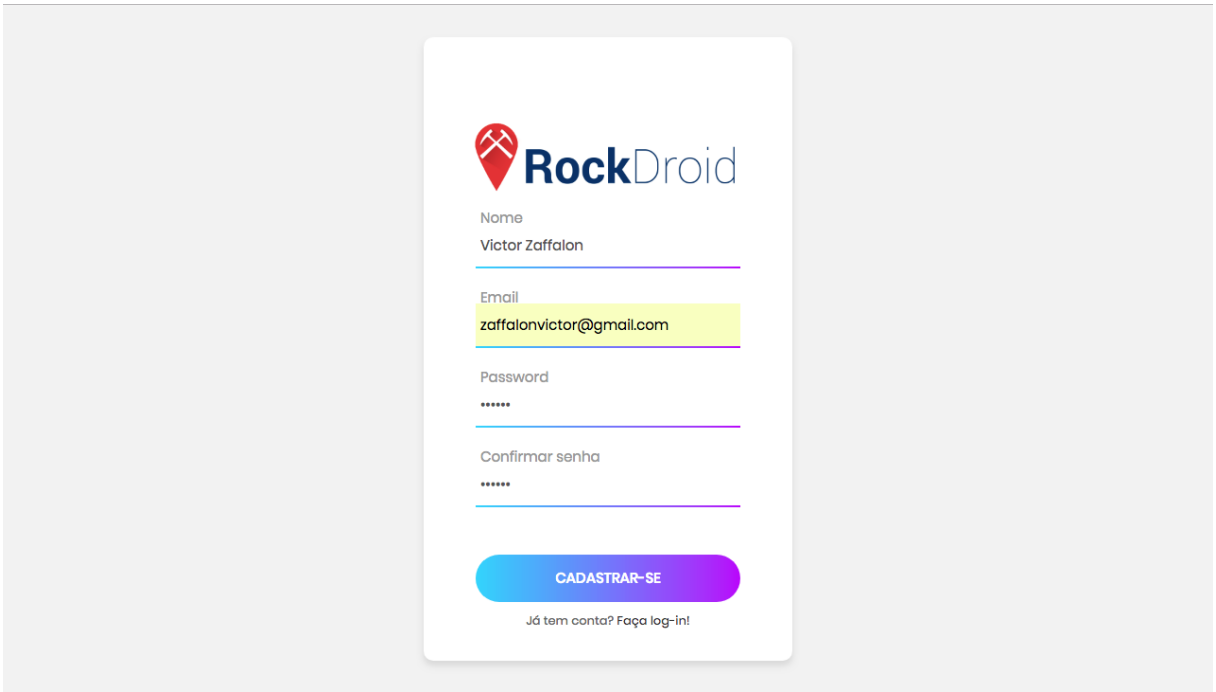


Figura 4.14: Tela de registro.

Minha Conta

A tela de meu perfil (Figura 4.15) que pode ser acessada a partir da foto no canto direito superior da *toolbar* de acesso do sistema, permite visualizar os dados cadastrados na conta do usuário. Esses dados obtidos no momento da criação da conta são o nome, email, data de criação da conta e o tipo de conta, sendo as duas opções possíveis de tipo definidas como aluno ou professor. Essa tela também permite adicionar ou alterar a foto relacionada a conta, de maneira a melhorar a identificação dos alunos usando o sistema.

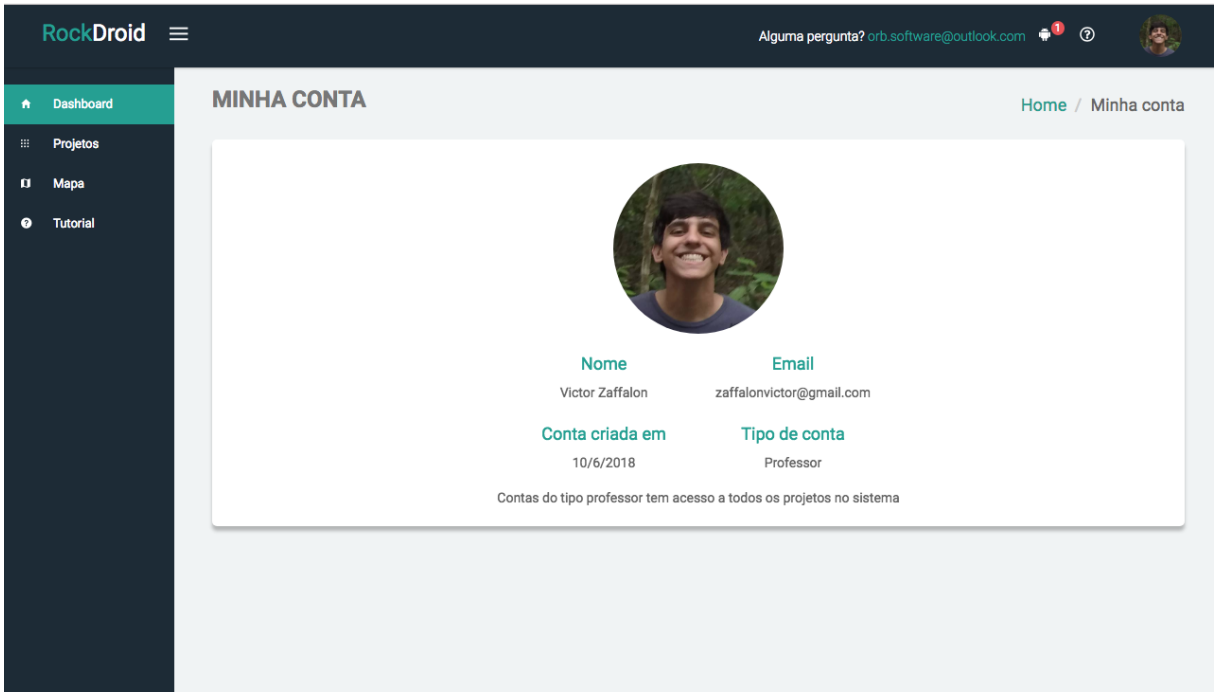


Figura 4.15: Tela de minha conta.

4.3.2 Tutorial

Uma vez que o usuário esteja logado no sistema, um código verifica se é o primeiro acesso no sistema, caso seja o usuário é redirecionado para a aba de tutorial que apresenta uma breve explicação de como usar e quais são as funcionalidades do sistema (Figura 4.16 e Figura 4.17).

O tutorial consiste em cinco etapas explicativas:

- Como fazer o *download* do aplicativo na loja da *google play* pelo link no canto superior direito no menu do sistema;
- Como criar uma conta de acesso pelo aplicativo ou pelo sistema Web;
- Como criar um projeto para inserção dos dados geográficos pelo aplicativo;
- Como sincronizar os dados do aplicativo com o banco de dados central;
- Como acessar os projetos no sistema Web para visualização dos dados coletados a partir do aplicativo.

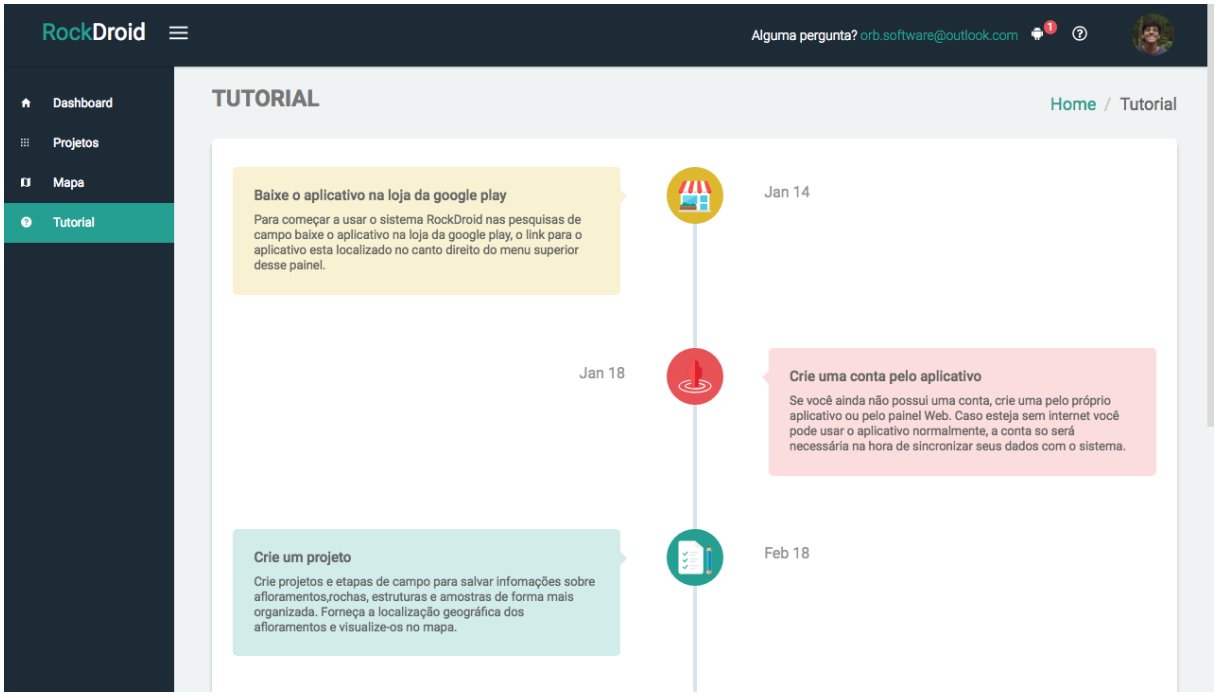


Figura 4.16: Tela de tutorial parte 1.

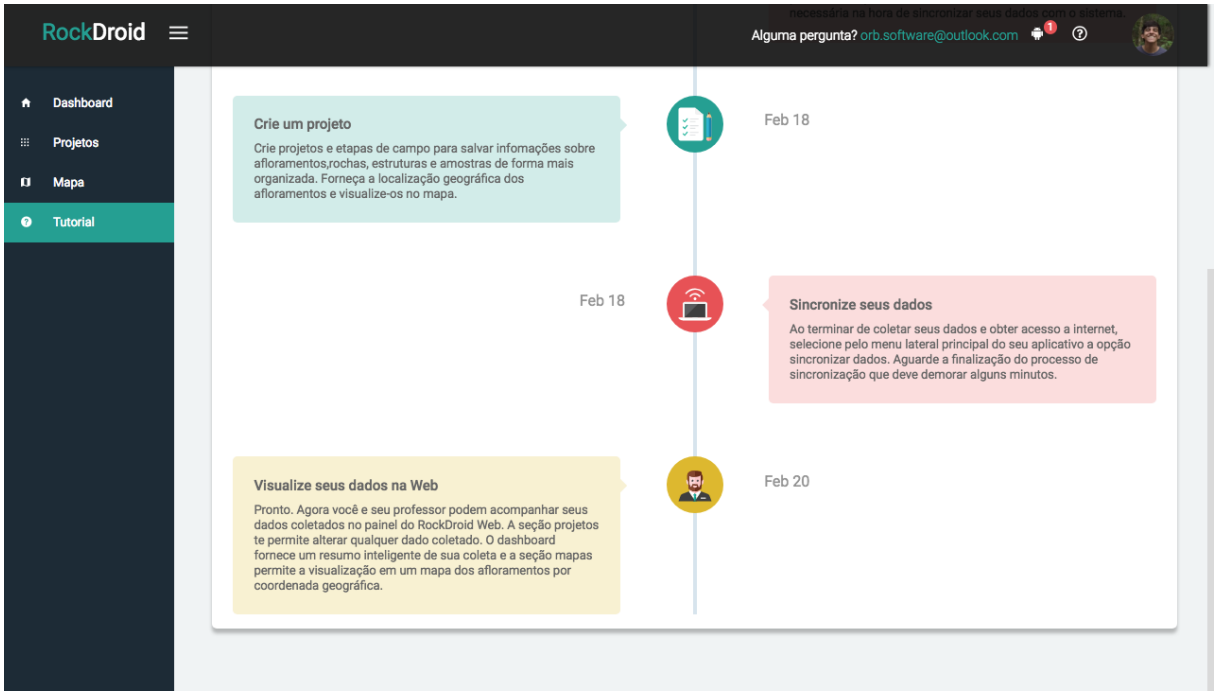


Figura 4.17: Tela de tutorial parte 2.

4.3.3 Projetos

A aba projetos (Figura 4.18) apresenta a lista de projetos criadas pelo aluno pelo aplicativo, e que foram sincronizados com o banco de dados, além dos projetos criados

por meio do sistema web. No caso do usuário acessando ter uma conta do tipo professor, a lista mostra os projetos de todos os alunos, listados a partir da data de criação, sendo ordenados da data mais atual de criação até a data de criação mais antiga.

Na coluna ações existe os botões de editar, excluir e exportar. A ação exportar cria e faz o *download* de um arquivo em formato de planilhas do Excel(XLS) contendo todos os dados do projeto selecionado. Ao clicar em um dos projetos na tabela, o usuário é redirecionado para a página de lista de etapas do elemento selecionado.

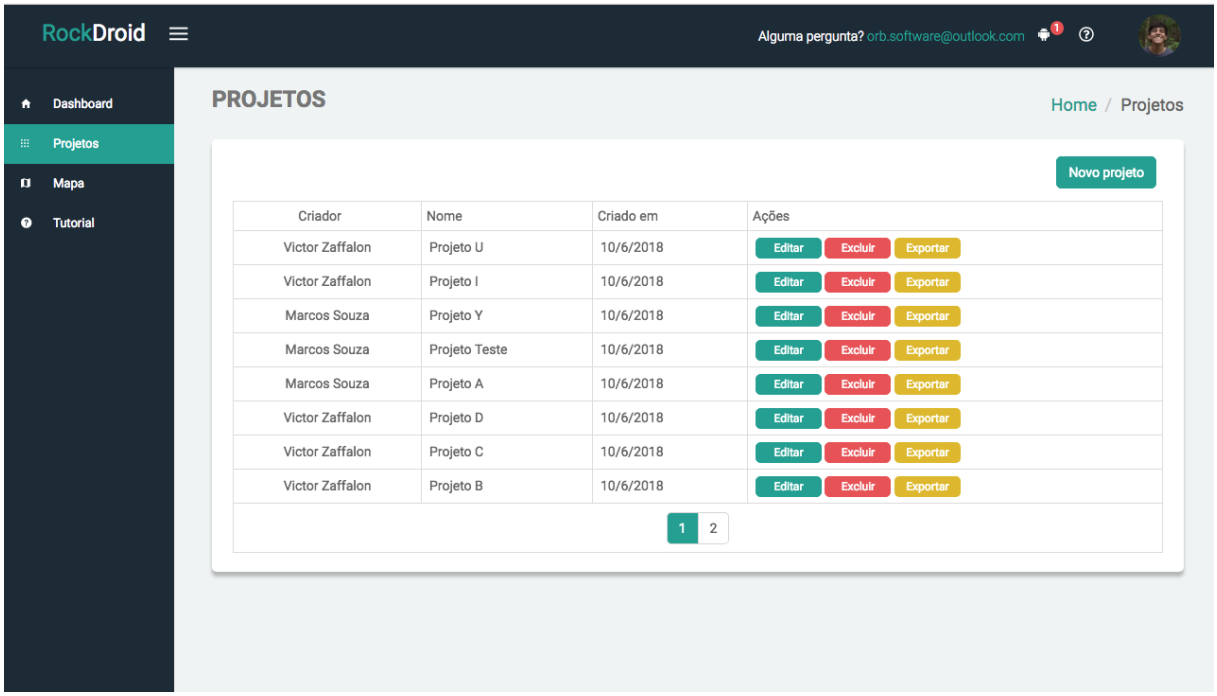


Figura 4.18: Listagem de projetos.

Criação de Projetos

Quando o usuário clica no botão (novo projeto), o sistema abre um modal (caixa de informações) com um formulário de criação de projeto (Figura 4.19). O modelo de dados do RockDroid define que não devem existir projetos sem etapas de campo, logo, após A criação do projeto, um modal com o formulário de criação de etapas é aberto, e a etapa deve ser criada para que o projeto seja criado com sucesso.

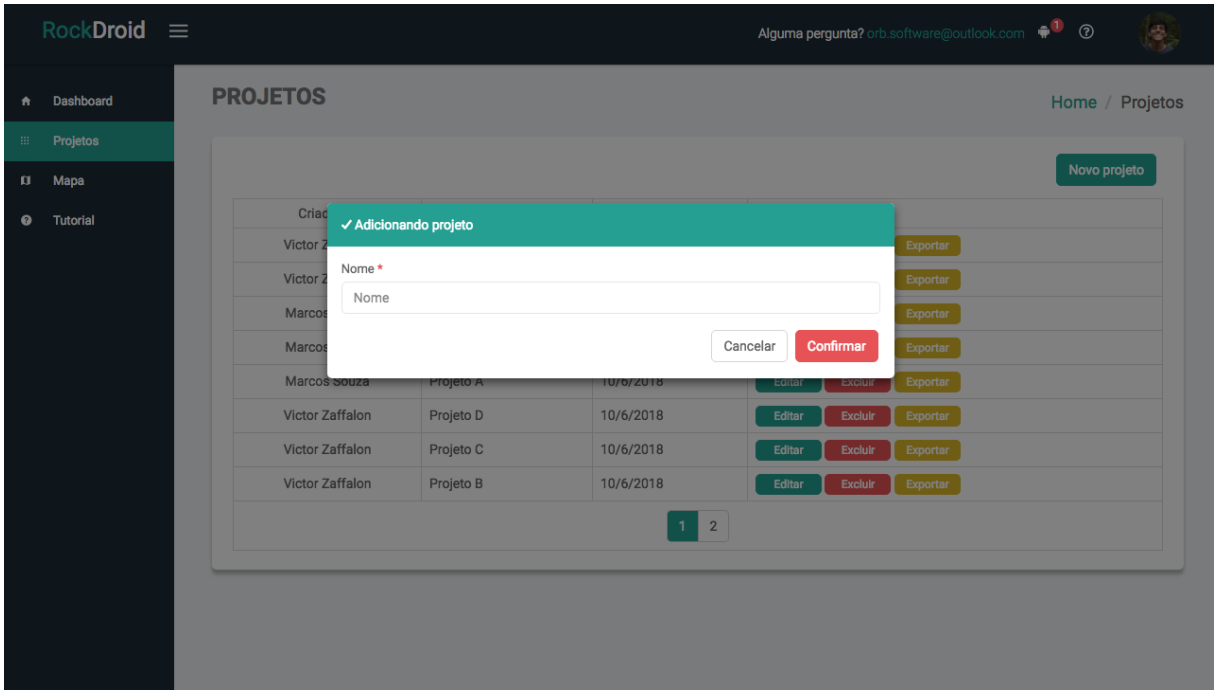


Figura 4.19: Criação de projetos.

Edição de Projetos

O nome do projeto pode ser editado a qualquer momento ao clicar na ação editar na lista de projetos. Um formulário de edição (Figura 4.20) é aberto, permitindo a atualização dos dados. Ao se confirmar a edição a lista de projetos é atualizada com os novos valores.

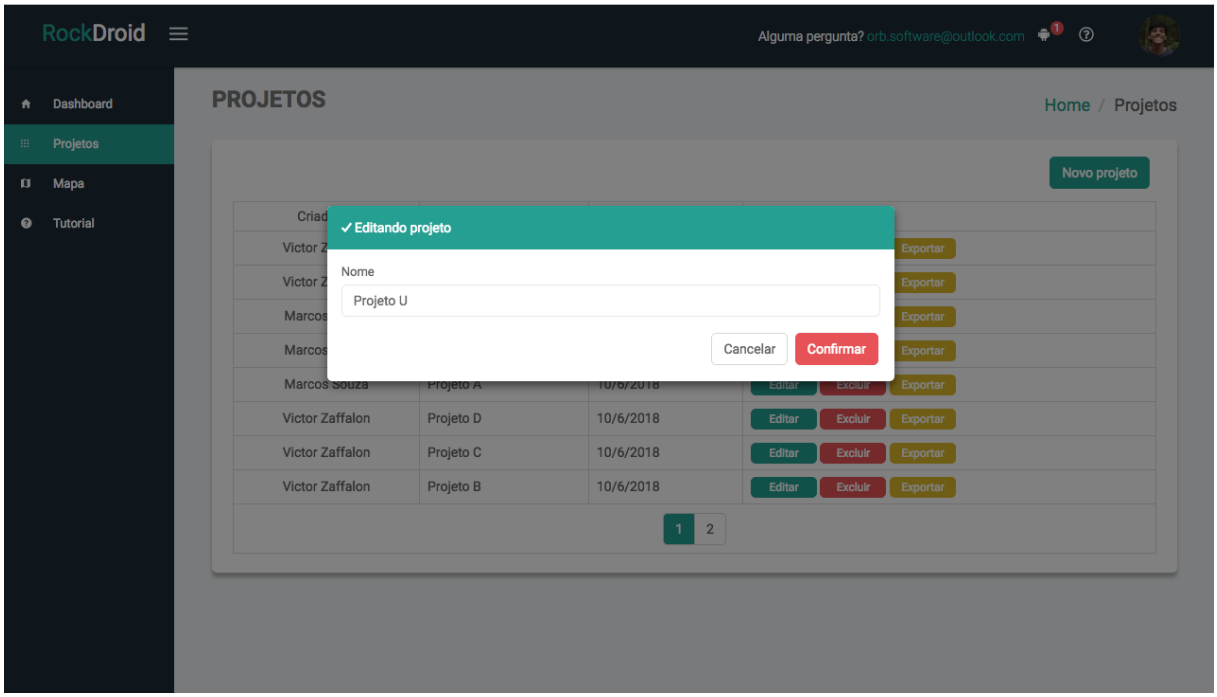


Figura 4.20: Edição de projetos.

Exclusão de projetos

A ação excluir projeto remove o projeto do sistema e todos as entidades relacionadas como etapas, afloramentos, estruturas, rochas e amostras. Como essa é uma ação considerada destrutiva, um modal é mostrado ao usuário pedindo a confirmação da ação e alertando das consequências do processo antes que a ação seja completada (Figura 4.21).

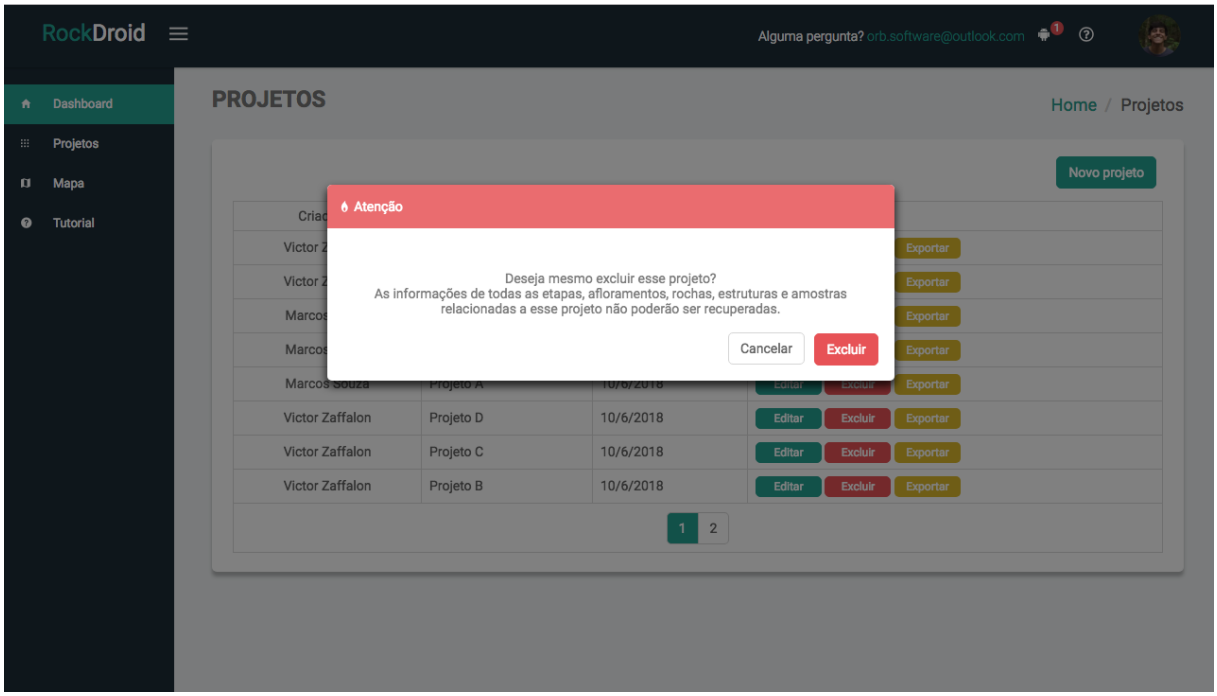


Figura 4.21: Exclusão de projetos.

Exportação para xls

Na Figura 4.22 é possível ver a planilha gerada de projetos com a informação do projeto ao clicar no botão de exportar. Em conjunto com a planilha contendo os dados referentes ao projeto, também é criado uma planilha para cada tabela referente aos elementos contidos dentro do projeto exportado.

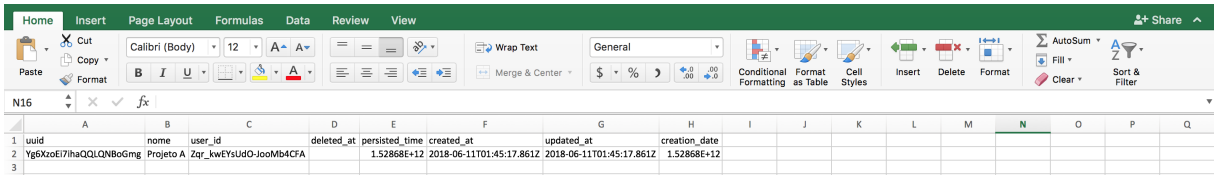
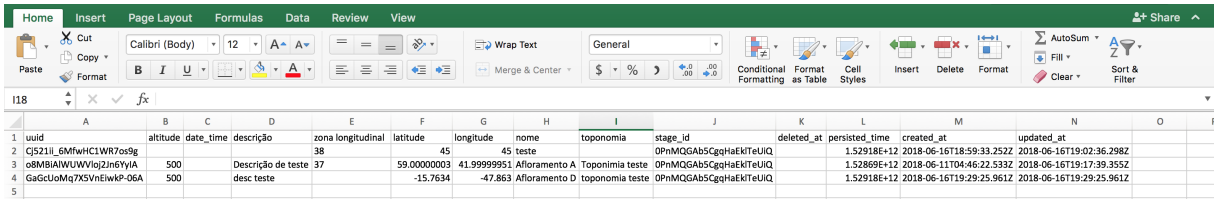


Figura 4.22: Planilha de projeto.

A Figura 4.23 mostra a planilha de afloramentos contidos dentro do projeto selecionado e que foi gerada em conjunto com as planilhas de etapas, amostras, estruturas e rochas.



uuid	altitude	date_time	descrição	zona longitudinal	latitude	longitude	nome	toponomia	stage_id	deleted_at	persisted_time	created_at	updated_at
Cj521il_6MfwHC1WR7os9g				38	45	45	teste		0PnMQGAb5CgqHaEklTeUiQ		1.52918E+12	2018-06-16T18:59:33.252Z	2018-06-16T19:02:36.298Z
o8MBlAlWUWVlojZJn6YyIA	500		Descrição de teste	37	59.00000003	41.99999951	Afloramento A	Toponimia teste	0PnMQGAb5CgqHaEklTeUiQ		1.52869E+12	2018-06-11T04:46:22.533Z	2018-06-16T19:17:39.355Z
GaGcUoMq7X5VnEiwkP-06A	500		desc teste		-15.7634	-47.863	Afloramento D	toponomia teste	0PnMQGAb5CgqHaEklTeUiQ		1.52918E+12	2018-06-16T19:29:25.961Z	2018-06-16T19:29:25.961Z

Figura 4.23: Planilha de afloramentos.

4.3.4 Etapas

Criação de etapas

O formulário de criação de etapas (Figura 4.24) é aberto automaticamente assim que um projeto é criado. Depois que a etapa é criada, a lista de etapas é atualizada mostrando os dados criados. Os campos de criação de etapa são o nome da etapa, o município e a Unidade Federativa em que se realiza a etapa, além da data de início que pode ser escolhida a partir de um calendário. Por padrão, caso nenhuma data seja escolhida, ela fica definida como o dia atual obtido a partir da biblioteca *javascript* moment.js [7].

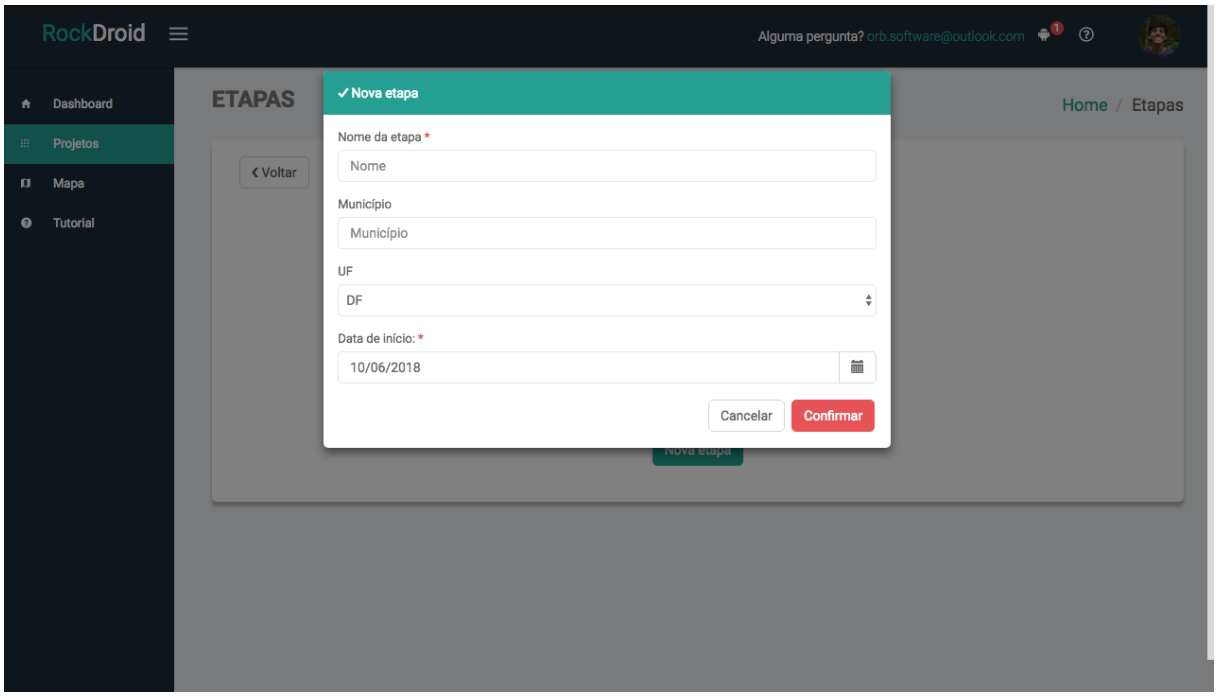


Figura 4.24: Criação de etapas.

Edição de Etapas

Um formulário de edição (Figura 4.25) é aberto permitindo a edição dos dados quando a ação editar de um elemento da lista de etapas é selecionada. No formulário de edição de etapa podem ser modificados os campos de nome da etapa, data de início, Município e Unidade Federativa. Ao se confirmar, a edição a lista de etapas é atualizada com os novos valores.

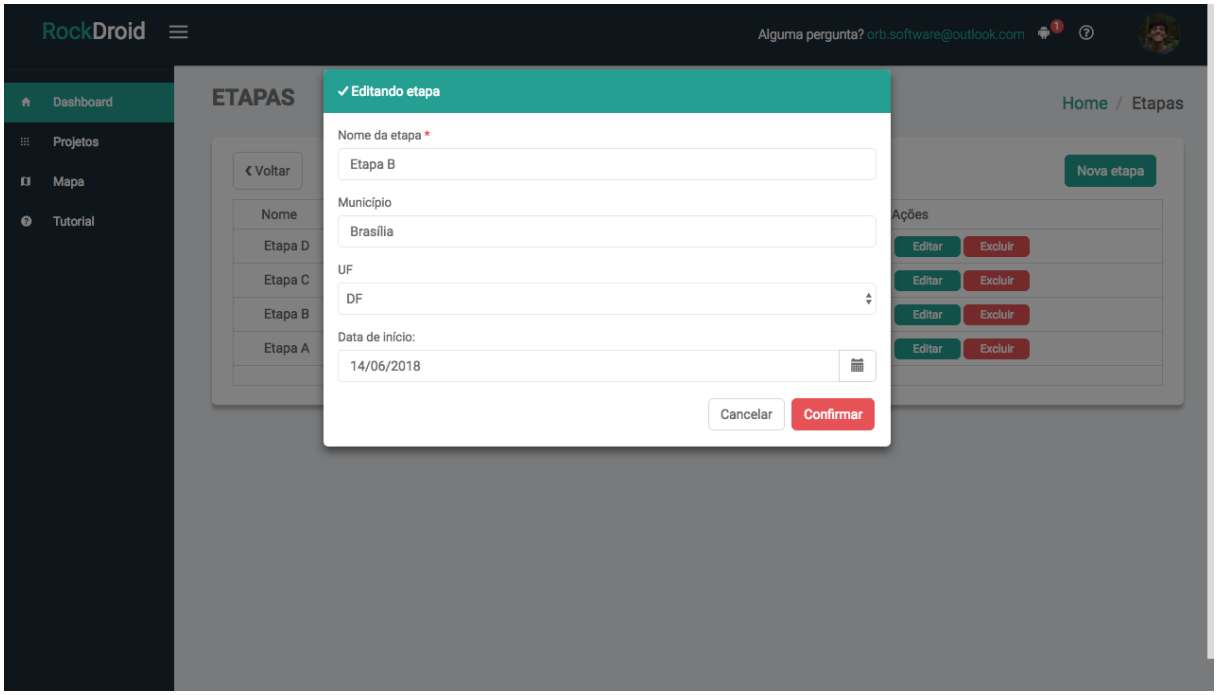


Figura 4.25: Edição de etapa.

Exclusão de Etapas

A ação excluir etapa remove a etapa do sistema e todos as entidades relacionadas com ela como afloramentos, estruturas, rochas e amostras. Como essa é uma ação considerada destrutiva, um modal (Figura 4.26) é mostrado ao usuário, pedindo a confirmação da ação e alertando das consequências do processo antes que a ação seja completada. Ao finalizar, a ação, o modal fecha e a lista de etapas é atualizada.

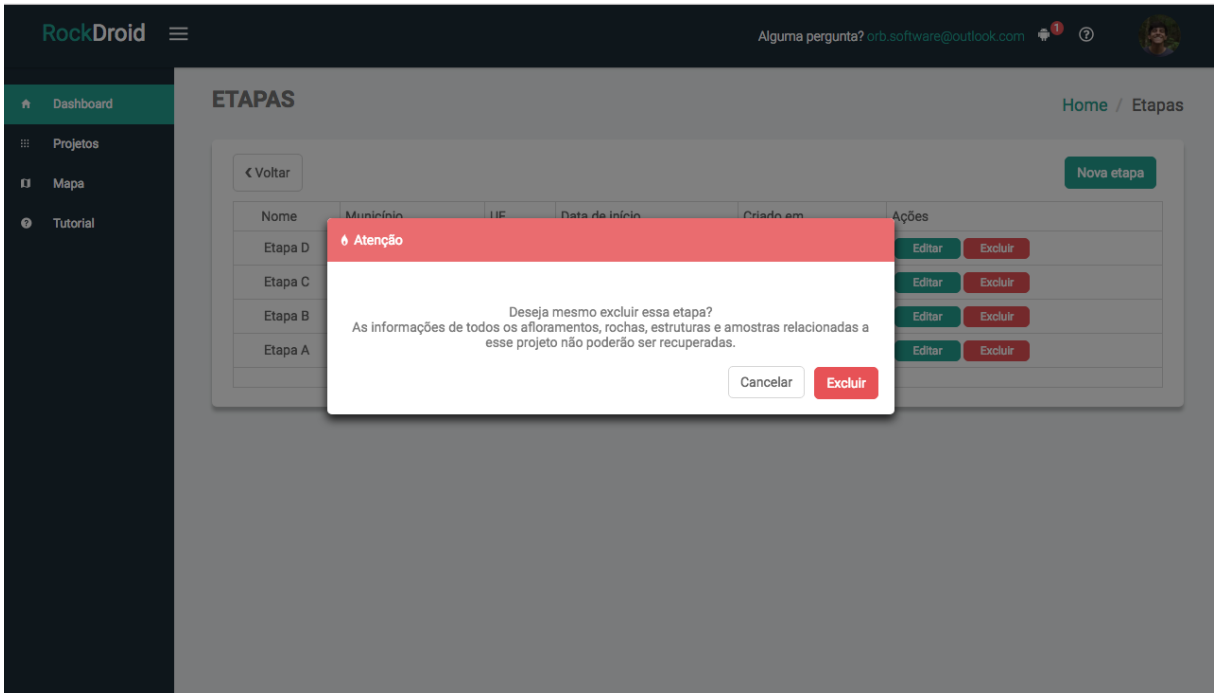


Figura 4.26: Exclusão de etapas.

4.3.5 Afloramentos

A lista de afloramentos (Figura 4.27) mostra todos os afloramentos da etapa selecionada, nessa lista cada elemento possui a ação de editar e excluir, assim como na lista de etapas. Ao clicar em um elemento da lista, o usuário é redirecionado para a página de informações do afloramento.

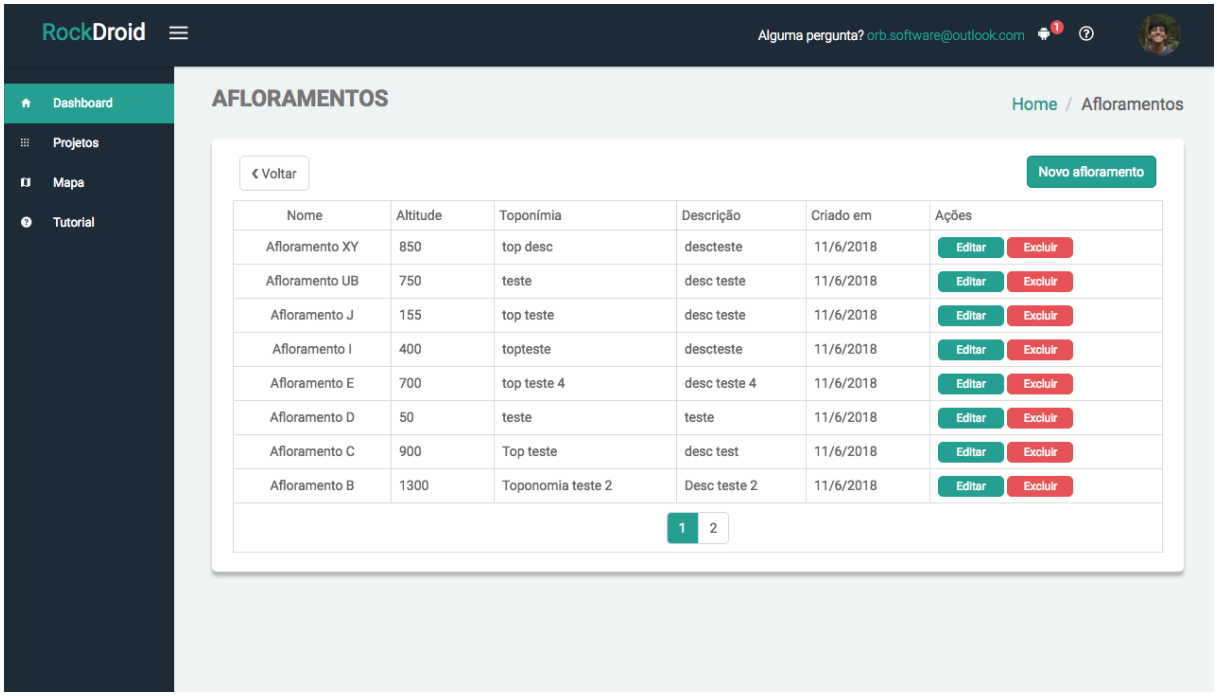


Figura 4.27: Listagem de Afloramentos.

Criação de afloramentos

Os campos de criação de afloramento podem ser vistos na Figura 4.28, dentre os campos listados são obrigatórios apenas os campos de nome, latitude e longitude no caso de escolha do uso de coordenadas geográficas(WGS-84).

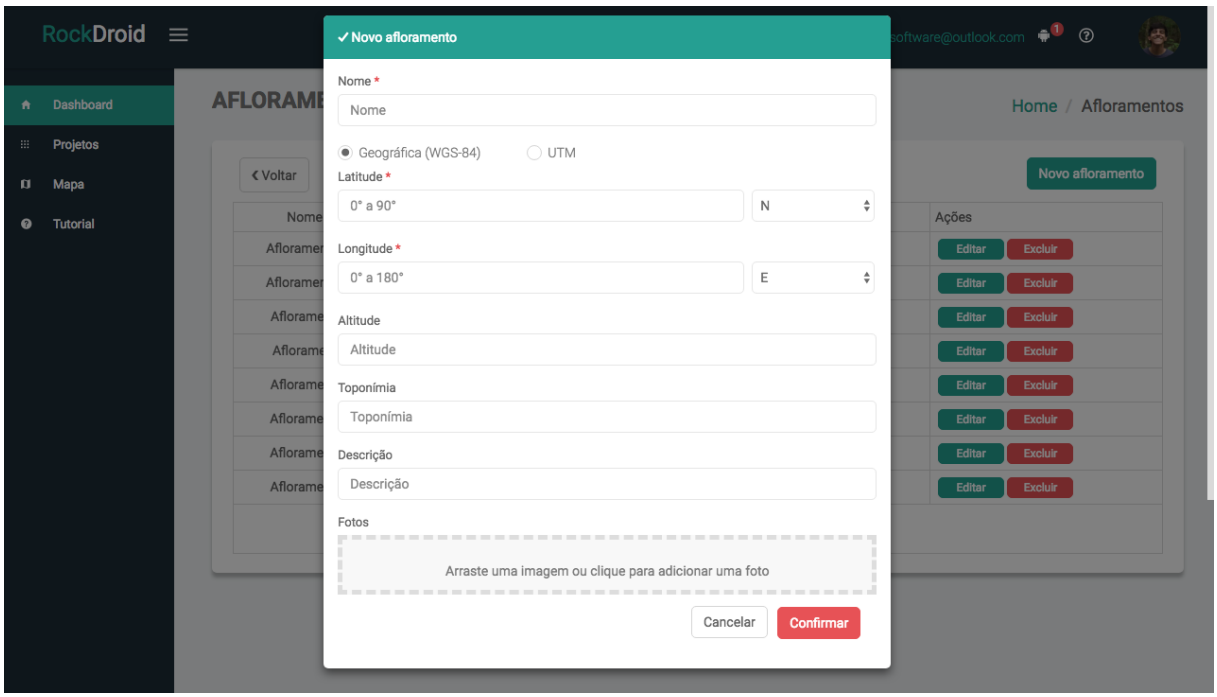


Figura 4.28: Criação de afloramentos.

Caso a opção de inserir as coordenadas no formato UTM (Figura 4.29) seja selecionada, o formulário se altera, os campos latitude e longitude somem e dão lugar aos campos Zona longitudinal, Zona latitudinal, Easting e Northing. Caso o usuário selecione a opção WGS-84, o sistema faz a conversão entre os dois formatos e preenche os campos de latitude e longitude com os valores correspondentes aos inseridos no formato UTM. A conversão contraria (WGS-84 para UTM) acontece caso o usuário tenha inserido a latitude e a longitude, e selecionado a opção UTM.

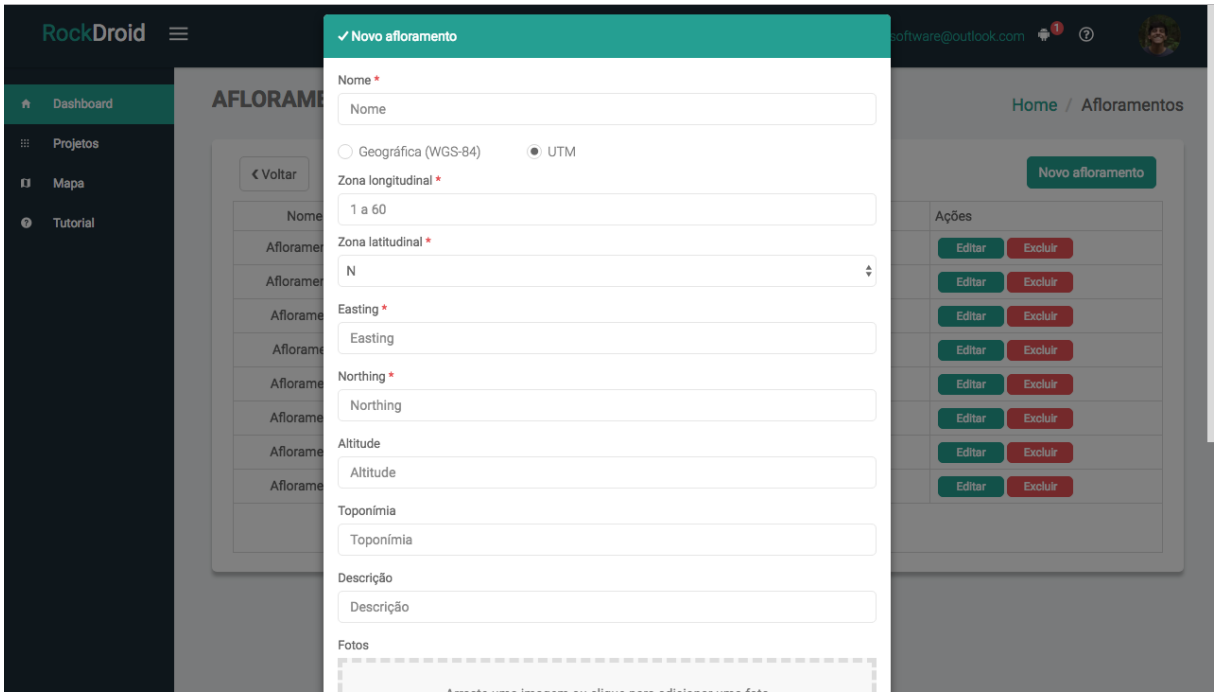


Figura 4.29: Criação de afloramentos por UTM

Por último, no formulário de criação de afloramentos, é possível adicionar fotos nos formatos de arquivo de imagem *png* e *jpg*, clicando ou arrastando fotos para o layout de adicionar fotos do formulário (Figura 4.30). Mais de uma foto pode ser adicionada ao mesmo tempo, e assim que uma imagem é inserida ela aparece na lista de imagens dentro do formulário. Todas as imagens são transformadas para base64, e salvas no banco de dados assim que o usuário termina de criar o afloramento clicando no botão 'Confirmar'.

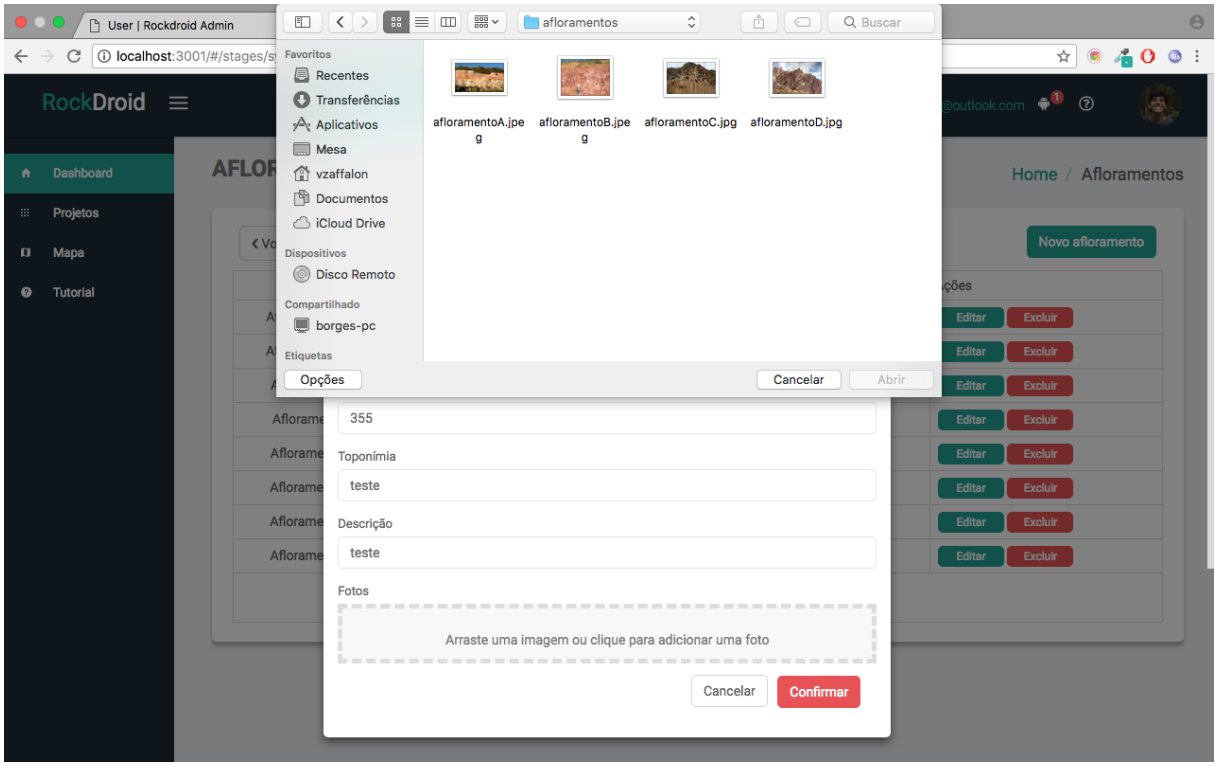


Figura 4.30: Adicionando foto em afloramento.

Edição de Afloramentos

Ao clicar na ação (editar) em um dos elementos da lista de afloramentos, o modal de editar afloramentos é aberto (Figura 4.31).

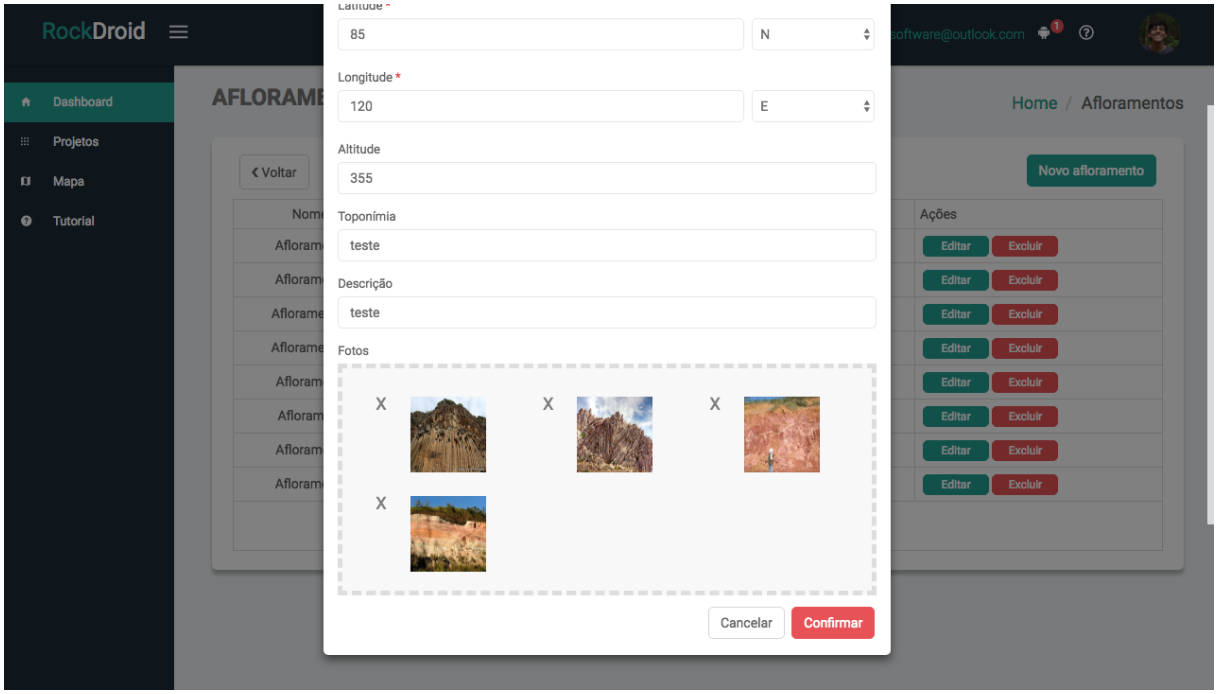


Figura 4.31: Edição de Afloramentos.

A edição de afloramentos, além de permitir alterar as coordenadas geográficas dos dados inseridos e os campos padrões de dados dos afloramentos, também permite excluir imagens clicando no X de um elemento da lista de imagens exibidas, ou adicionar novas imagens (Figura 4.32).

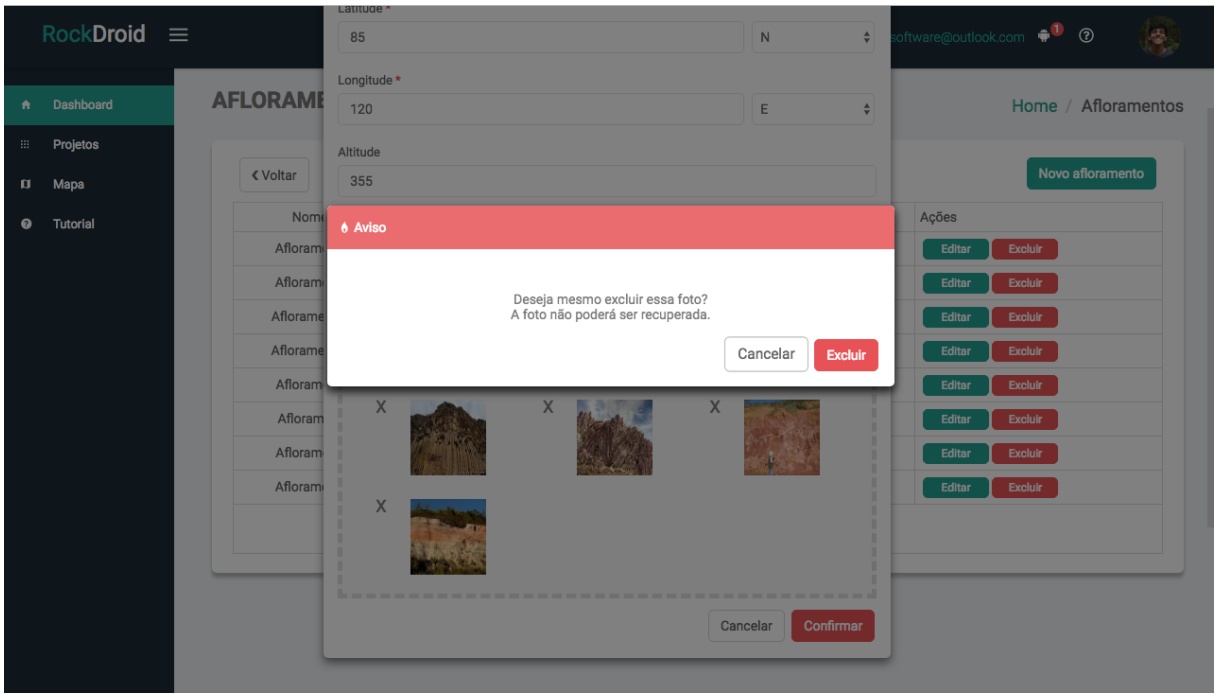


Figura 4.32: Exclusão e adição de fotos em afloramento.

Exclusão de Afloramentos

Ao clicar na ação 'Excluir' em um elemento da lista de afloramentos, um modal de confirmação é aberto avisando que os dados relacionados a aquele afloramento não podem ser recuperados caso a ação seja confirmada (Figura 4.33).

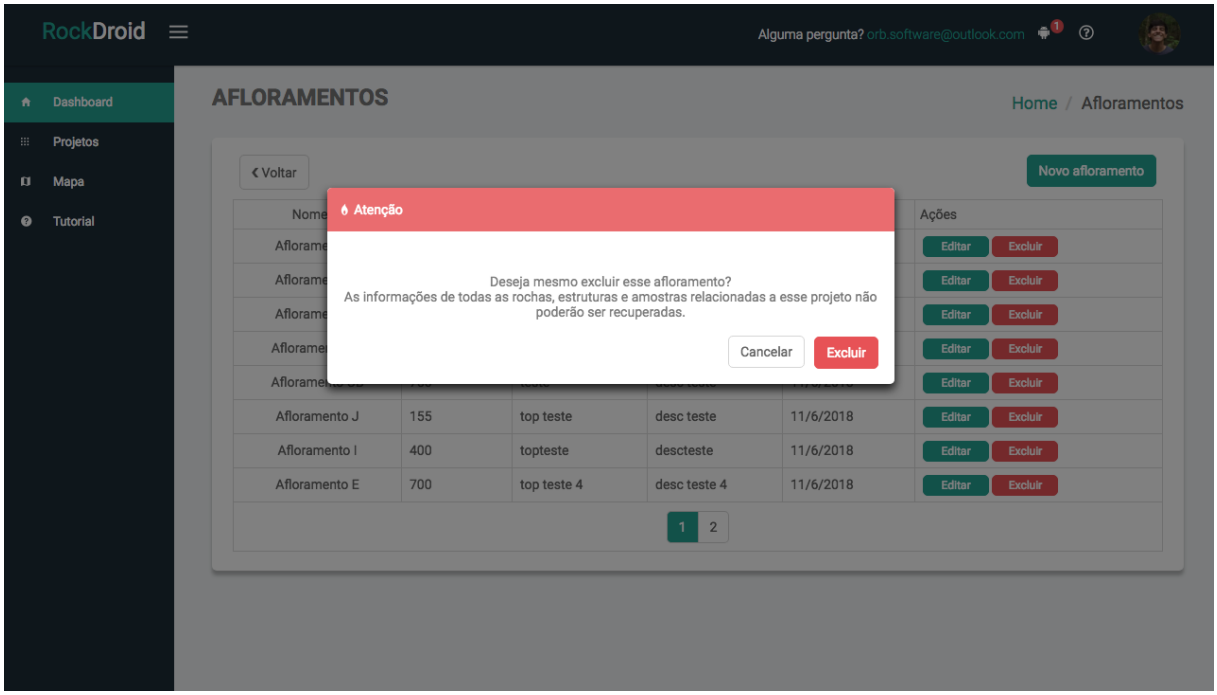


Figura 4.33: Exclusão de Afloramentos.

4.3.6 Informações do afloramento

Ao selecionar um elemento na lista de afloramentos, o usuário é redirecionado para a tela de informações do afloramento (Figura 4.34). Essa tela apresenta todas as informações cadastradas do afloramento, assim como a lista de todas as rochas, amostras, estruturas primárias e secundárias relacionadas a ele.

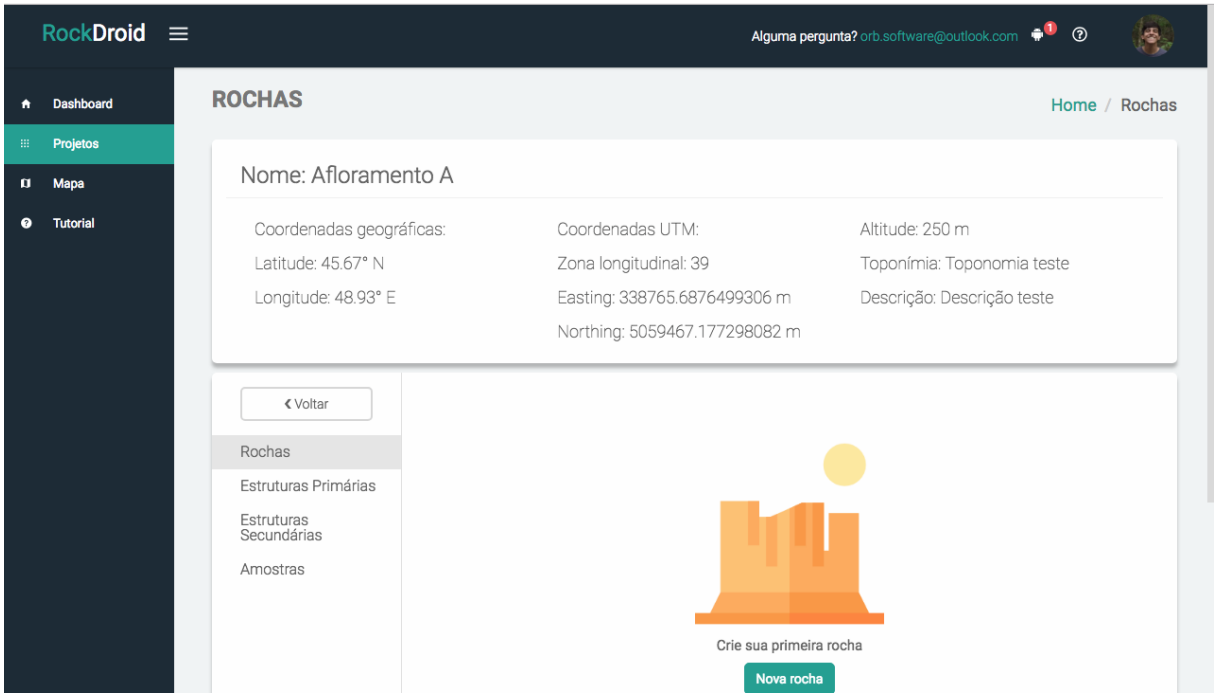


Figura 4.34: Informações do afloramento.

Listagem de Rochas

Dentro de informações do afloramento, ao selecionar a aba rochas, uma lista mostrando as rochas adicionadas ao afloramento aparece (Figura 4.35). Essa lista possui as mesma ações de todas as outras tabelas exibidas, criar, excluir e editar os elementos da lista.

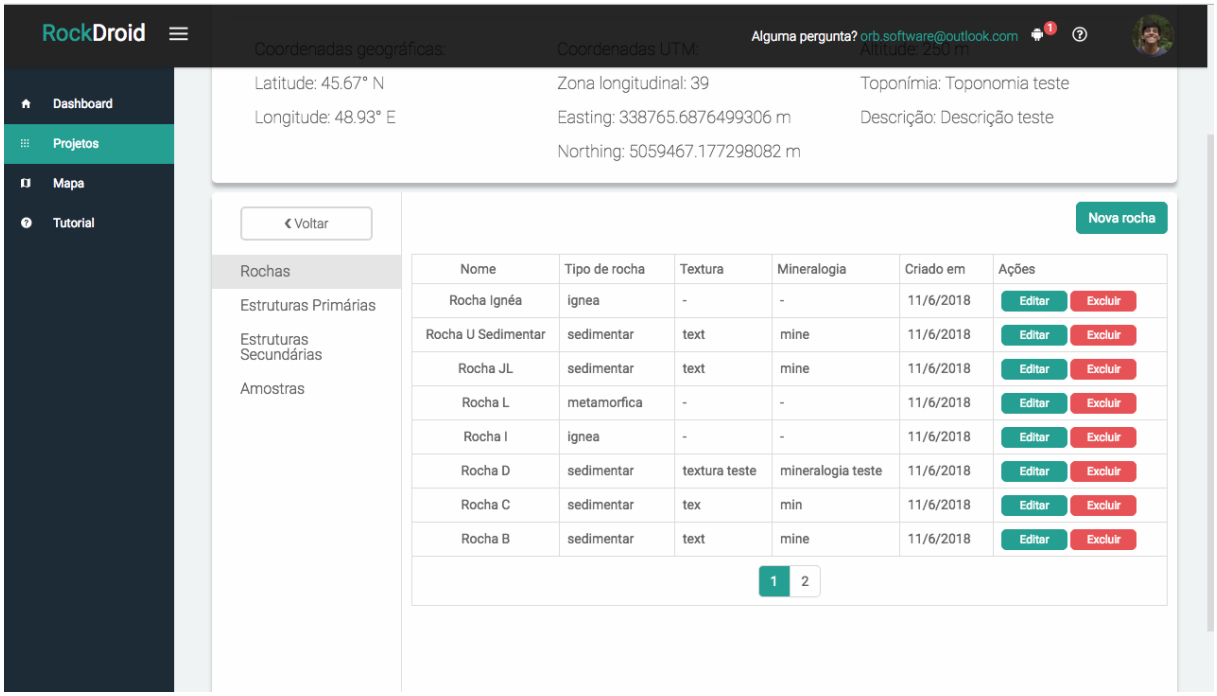


Figura 4.35: Listagem de rochas do afloramento.

Criação de Rochas

Ao clicar na ação 'NOVA ROCHA' o formulário de criação de rocha é aberto, existem três tipos de rochas que podem ser inseridas, sendo elas do tipo ígnea, metamórfica ou segmentar. Para cada seleção de tipo de rocha, um formulário diferente é renderizado na tela do usuário.

O formulário de Rocha Ígnea (Figura 4.36) possui os campos de cor e composição, textura, tamanho, trama, mineralogia, nomenclatura. Além disso, existe a possibilidade de relacionar a rocha ígnea com estruturas primárias e secundárias.

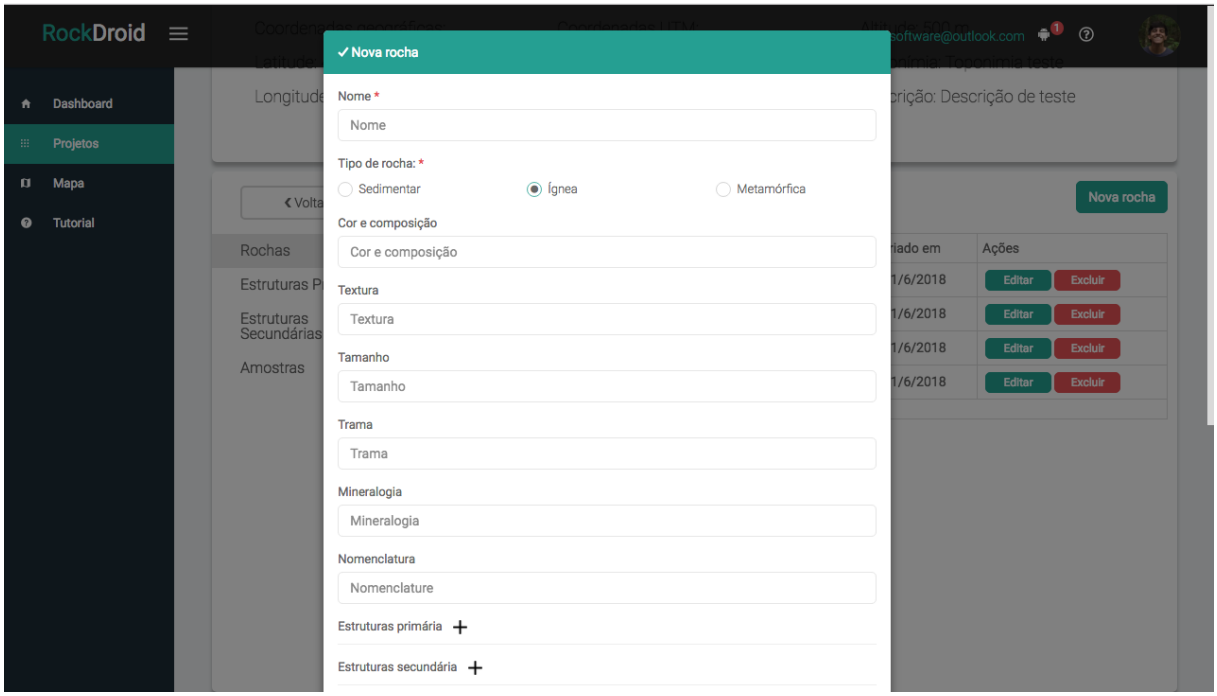


Figura 4.36: Criação de rocha ígnea.

Ao clicar no botão com ícone de '+' ao lado de estruturas, uma lista com todas as estruturas do afloramento aparece (Figura 4.37). O usuário só precisa selecionar um elemento da lista para adicionar o relacionamento desse elemento com a rocha.

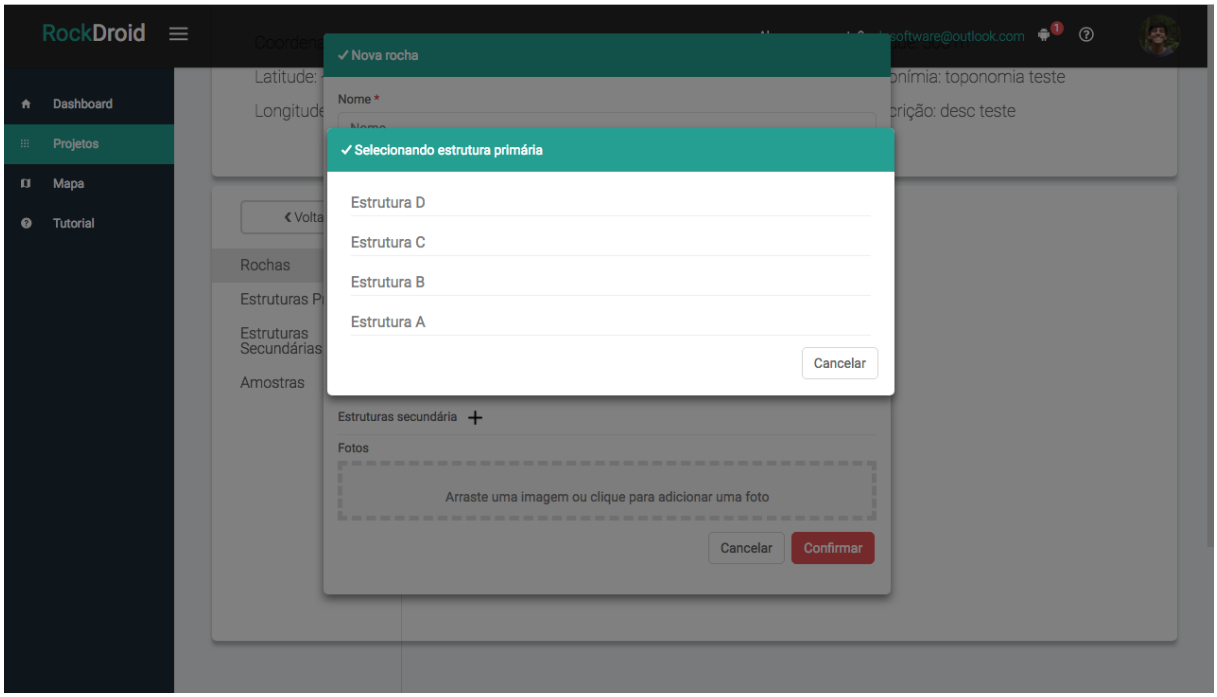


Figura 4.37: Relacionando estrutura a rocha.

O formulário tipo Rocha Metamórfica (Figura 4.38) possui os campos grau metamórfico e composição, além das opções de adicionar estruturas e fotos.

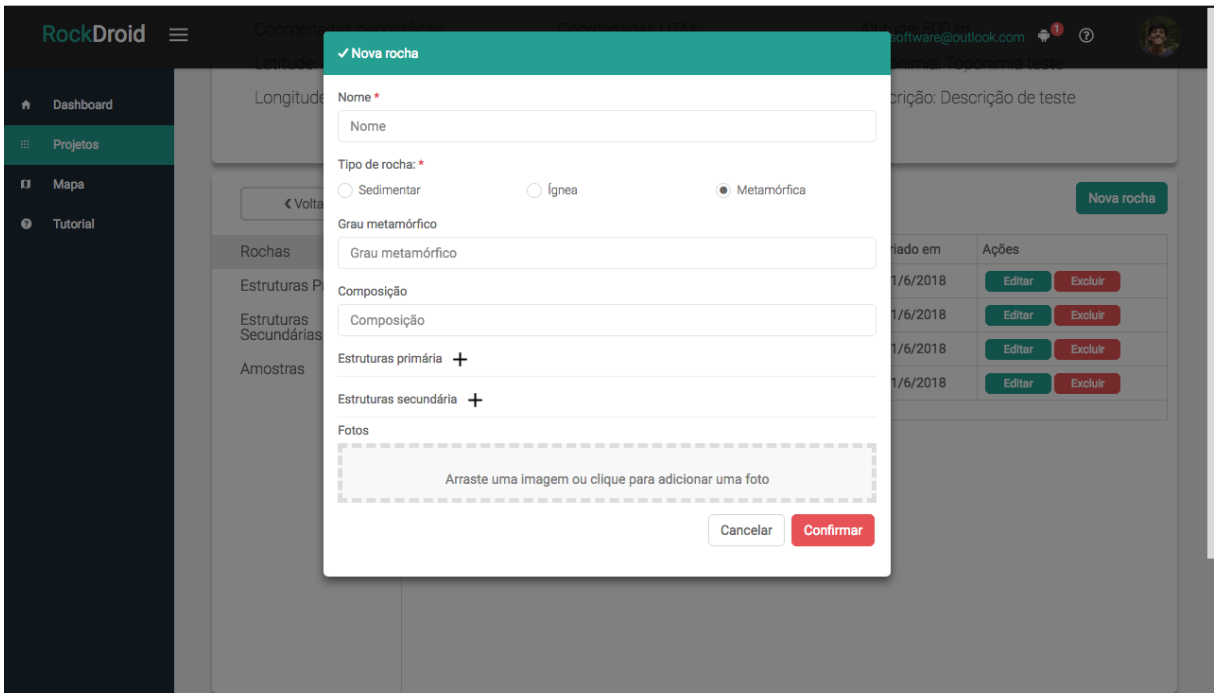


Figura 4.38: Criação de Rocha Metamórfica.

O tipo Rocha Sedimentar (Figura 4.39) possui os campos, textura e mineralogia, além dos campos padrões de adicionar fotos e estruturas..

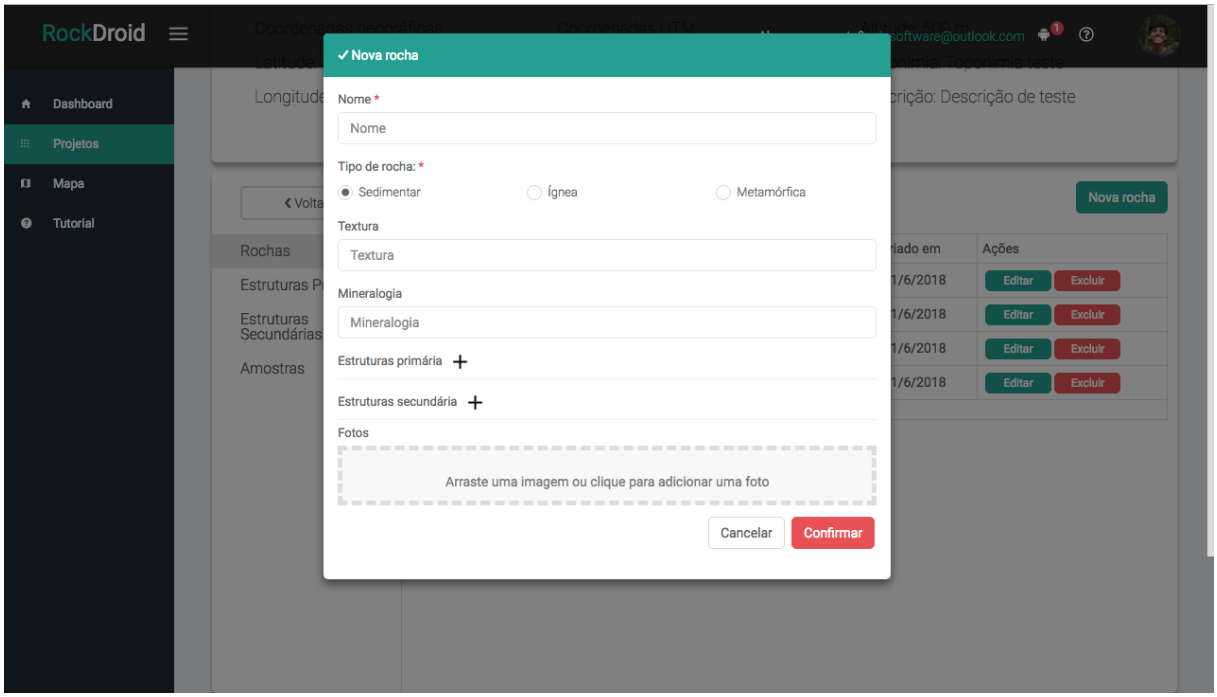


Figura 4.39: Criação de rocha segmentar.

Edição de Rochas

A edição de rochas, assim como as outras edições do sistema, permite adicionar e remover novas fotos, alterar os dados dos campos inseridos e, até mesmo, alterar o tipo de rocha.

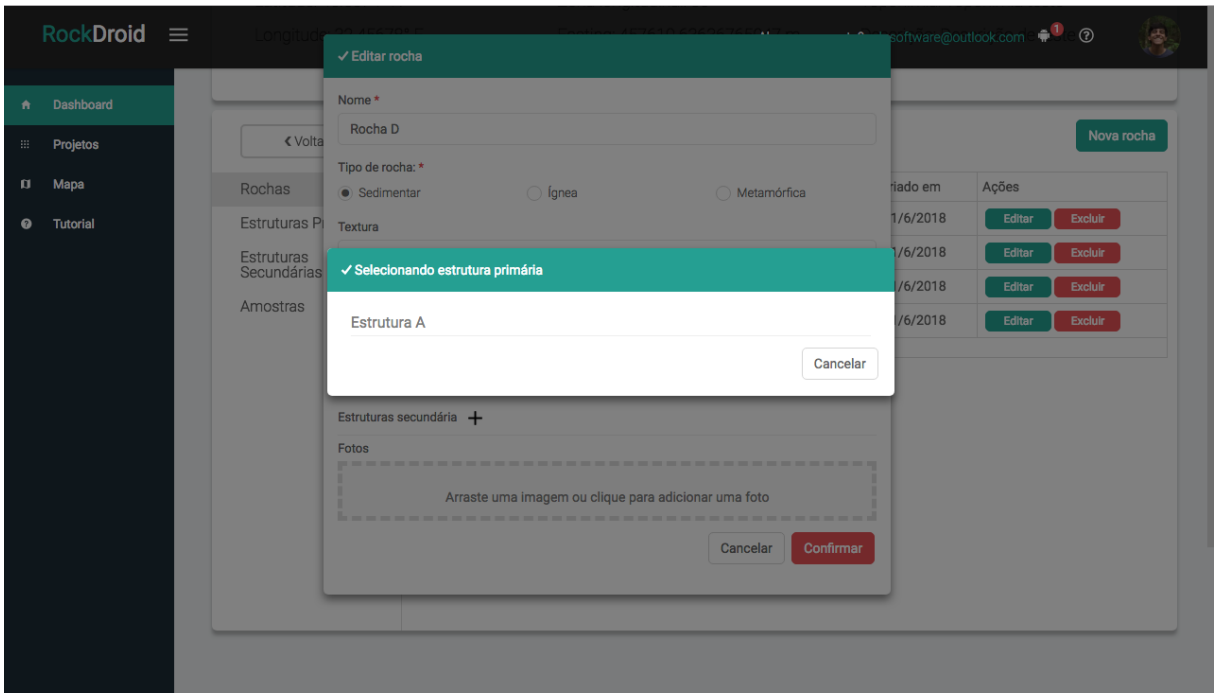


Figura 4.40: Edição de rocha.

Exclusão de Rochas

O modal de exclusão de rochas segue o mesmo formato padrão de confirmação de ação de todos os outros do sistema (Figura 4.41).

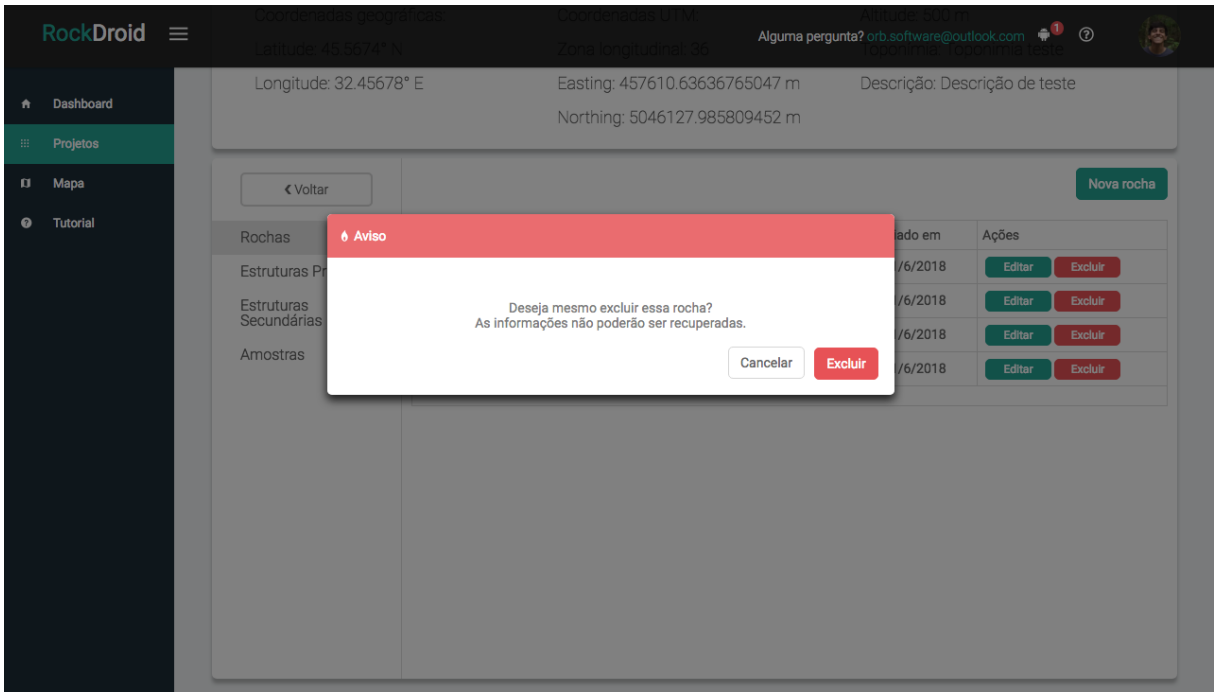


Figura 4.41: Exclusão de rochas.

Listagem de Amostras

A lista de amostras (Figura 4.42) segue o mesmo padrão das listas do sistema, quando uma quantidade muito alta de elementos é adicionada na lista, uma paginação da lista é criada para facilitar visualização dos dados.

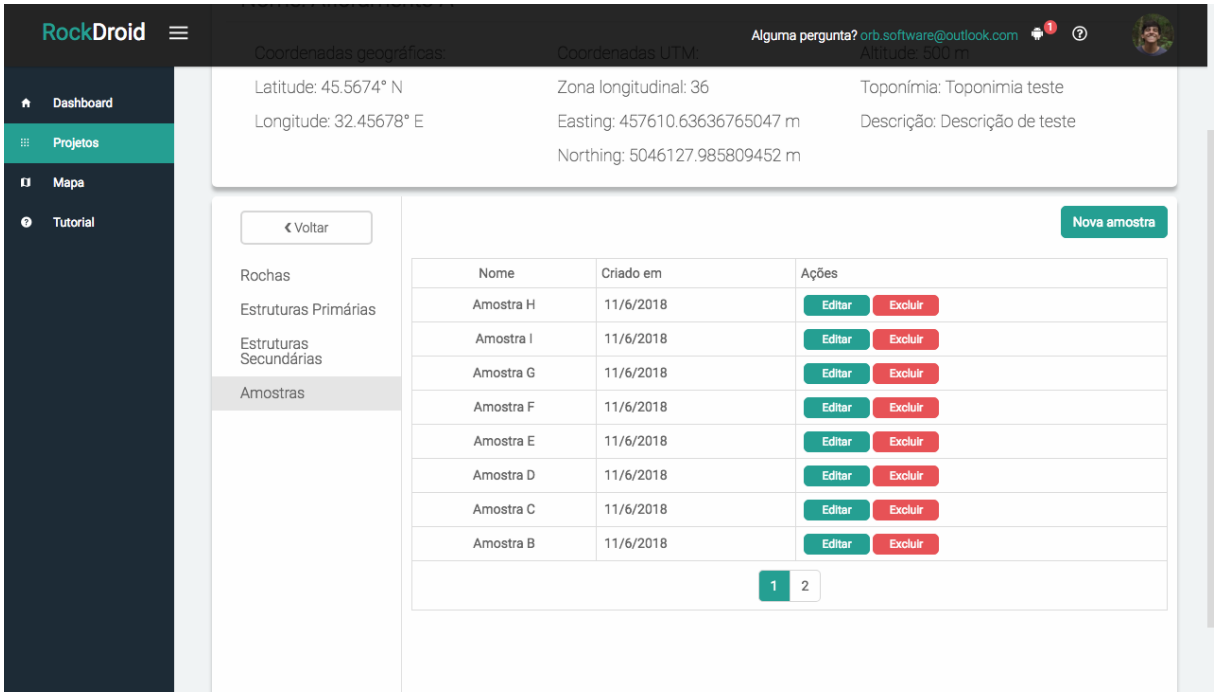


Figura 4.42: Lista de amostras.

Criação de Amostras

A criação de amostras (Figura 4.43) é bem simples sendo necessário apenas inserir o nome da amostra e algumas fotos.

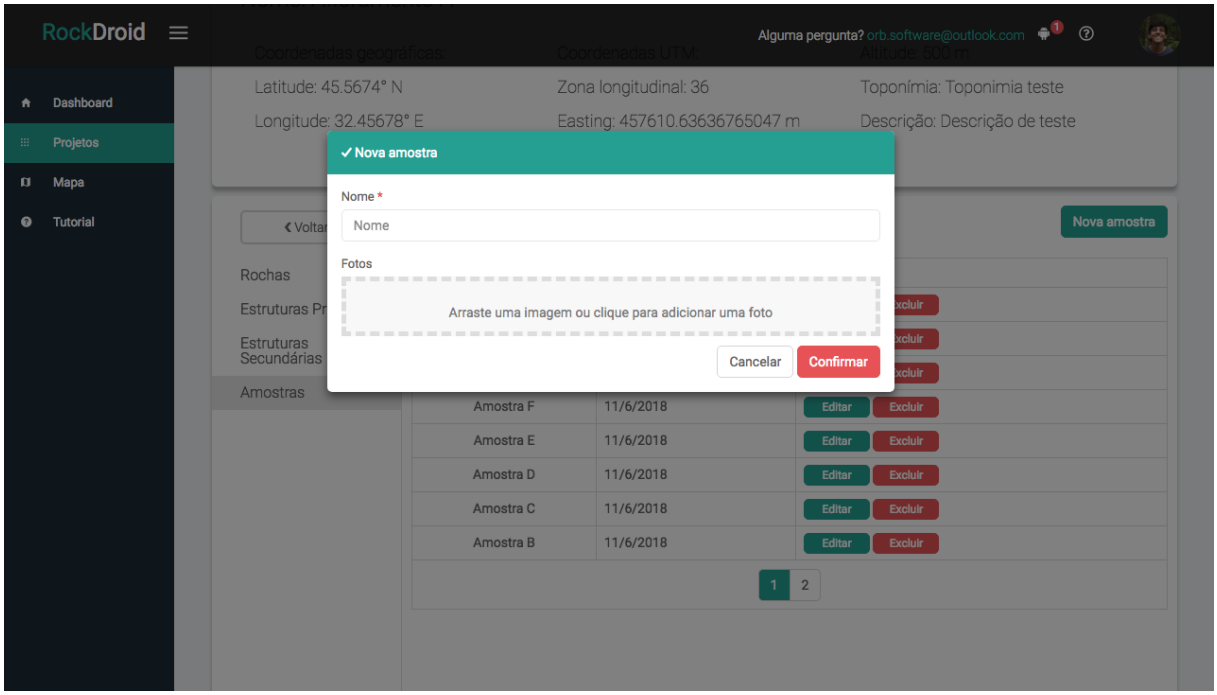


Figura 4.43: Exclusão de etapas.

Edição de Amostras

Editar amostras, assim como os outros formulários, com fotos permite além de alterar os campos, adicionar e remover fotos (Figura 4.44). A exclusão de amostras também segue o padrão com modal de confirmação do sistema (Figura 4.45).

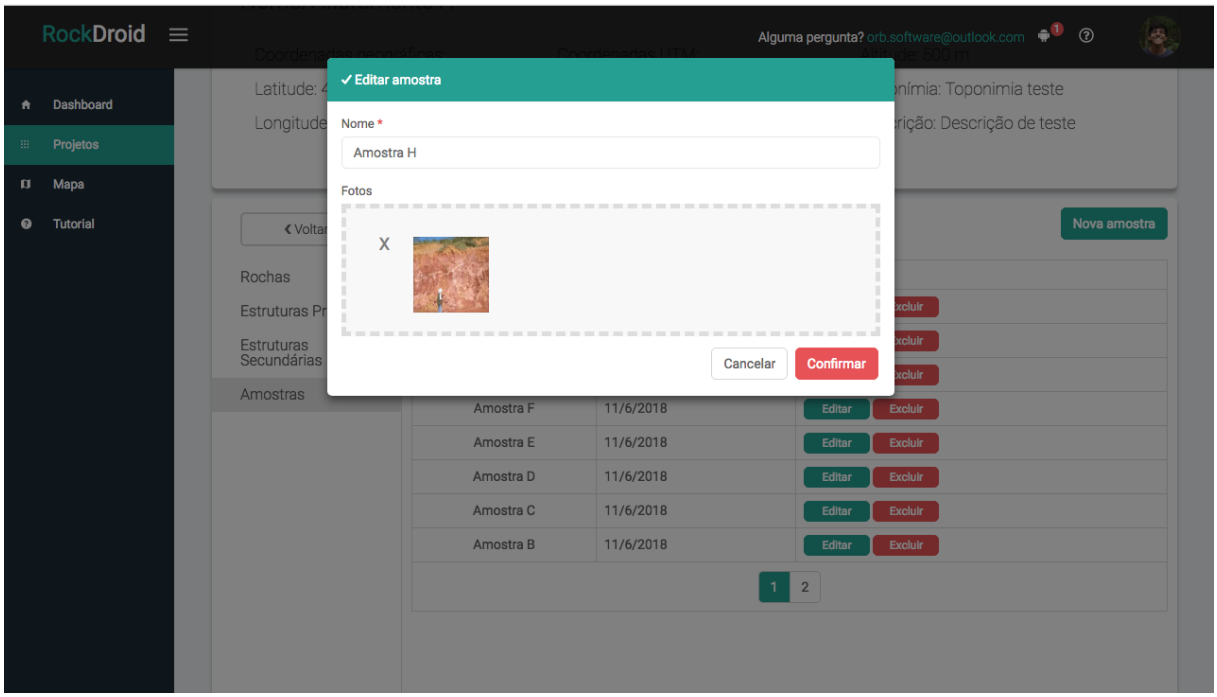


Figura 4.44: Edição de amostras.

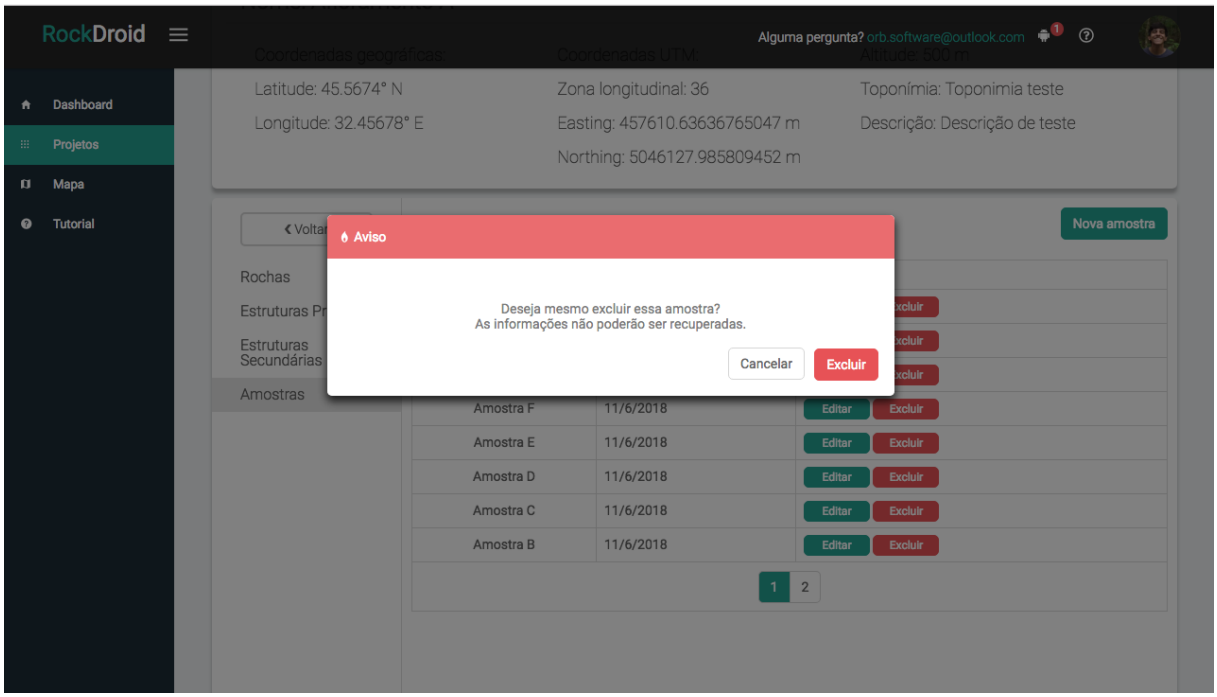


Figura 4.45: Exclusão de amostras.

Listagem de Eestruturas

Seguindo o modelo de dados, as estruturas podem ser do tipo estruturas primárias ou secundárias, como os campos dos dois tipos de estruturas são diferentes é exibido uma lista para cada tipo (Figuras 4.46 e 4.47).

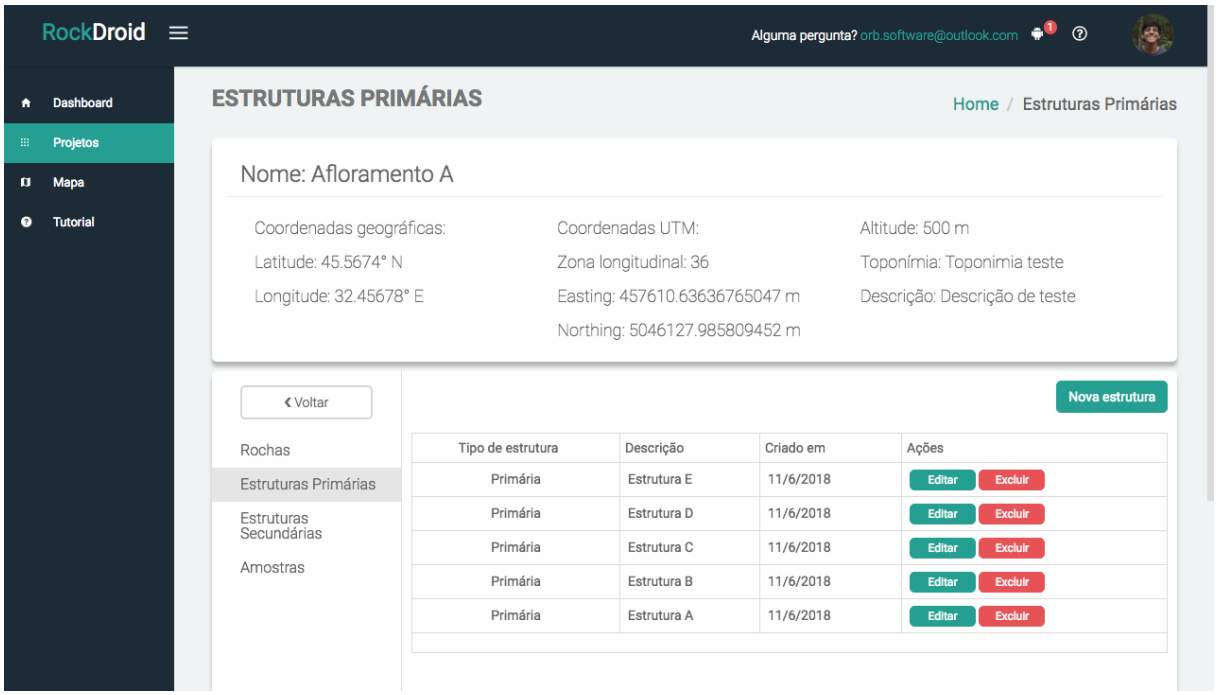


Figura 4.46: Lista de estruturas primárias.

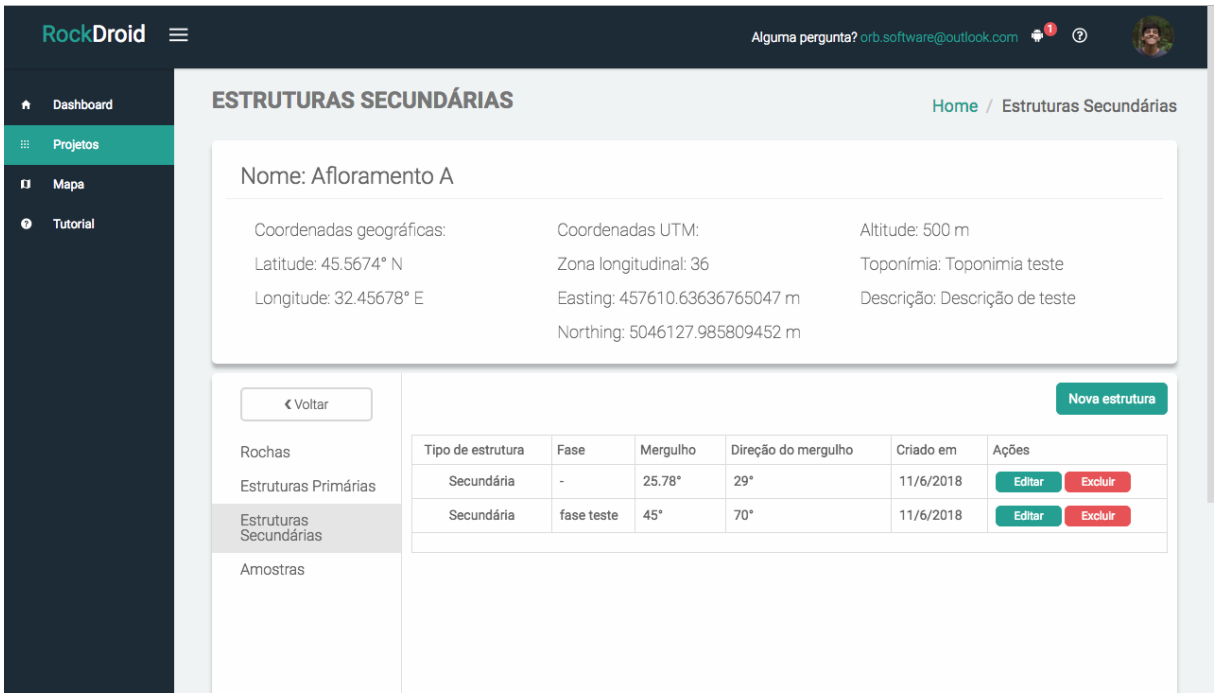


Figura 4.47: Lista de estrutura secundárias.

Criação de Estruturas

Tanto para a criação de estruturas primárias como secundárias, um mesmo modal de criação é usado, ao selecionar o tipo de estruturas os campos do formulários são alterados.

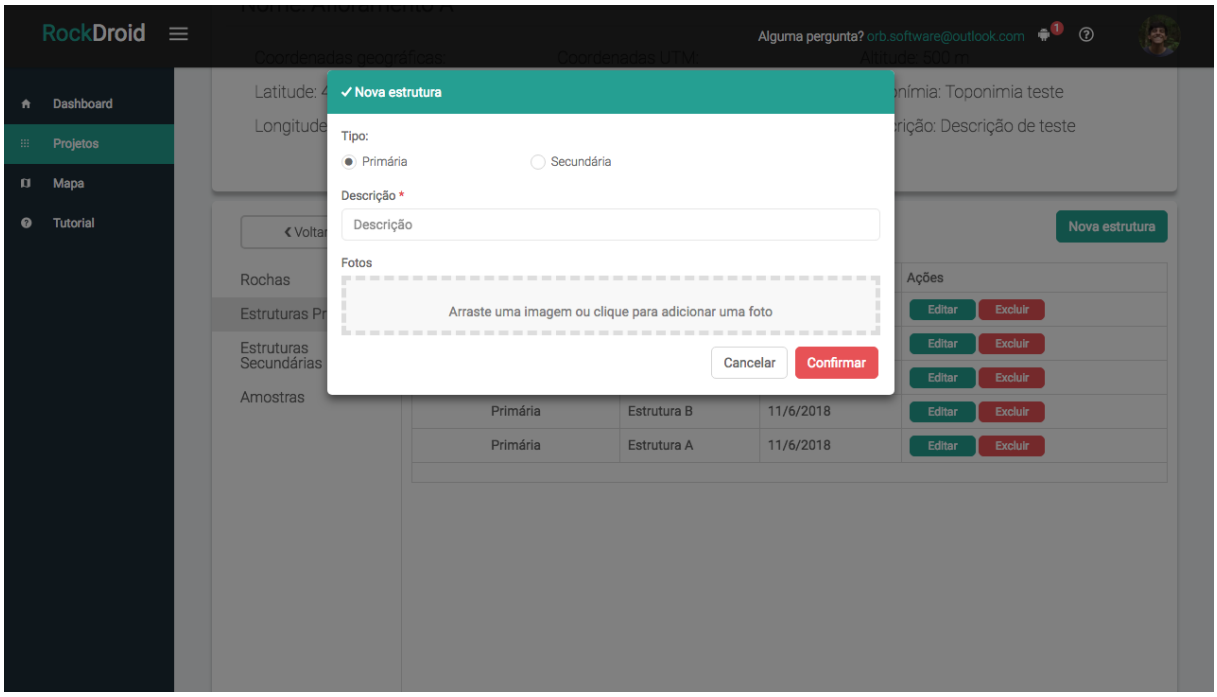


Figura 4.48: Criação de estrutura primária.

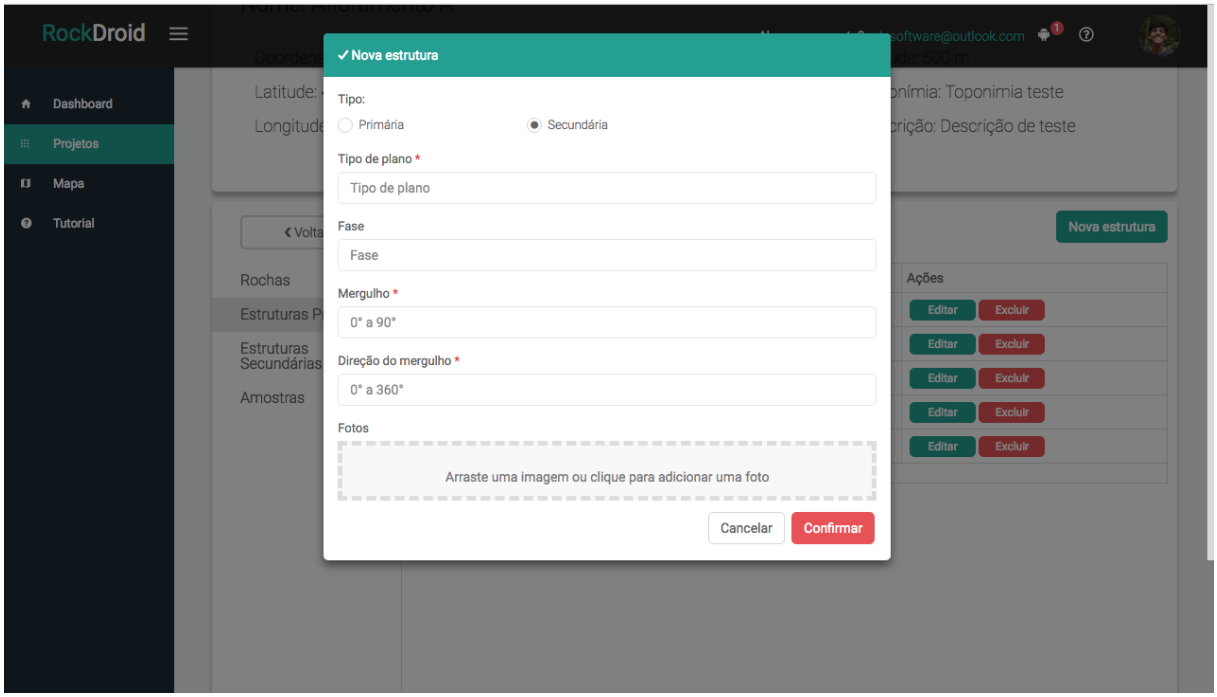


Figura 4.49: Criação de estrutura secundária.

Edição e Exclusão de Estruturas

A edição (Figura 4.50) permite alterar o tipo da estrutura e seus dados, além de permitir também adicionar e remover fotos da estrutura. A exclusão segue o mesmo padrão de todas as ações de excluir do sistema.

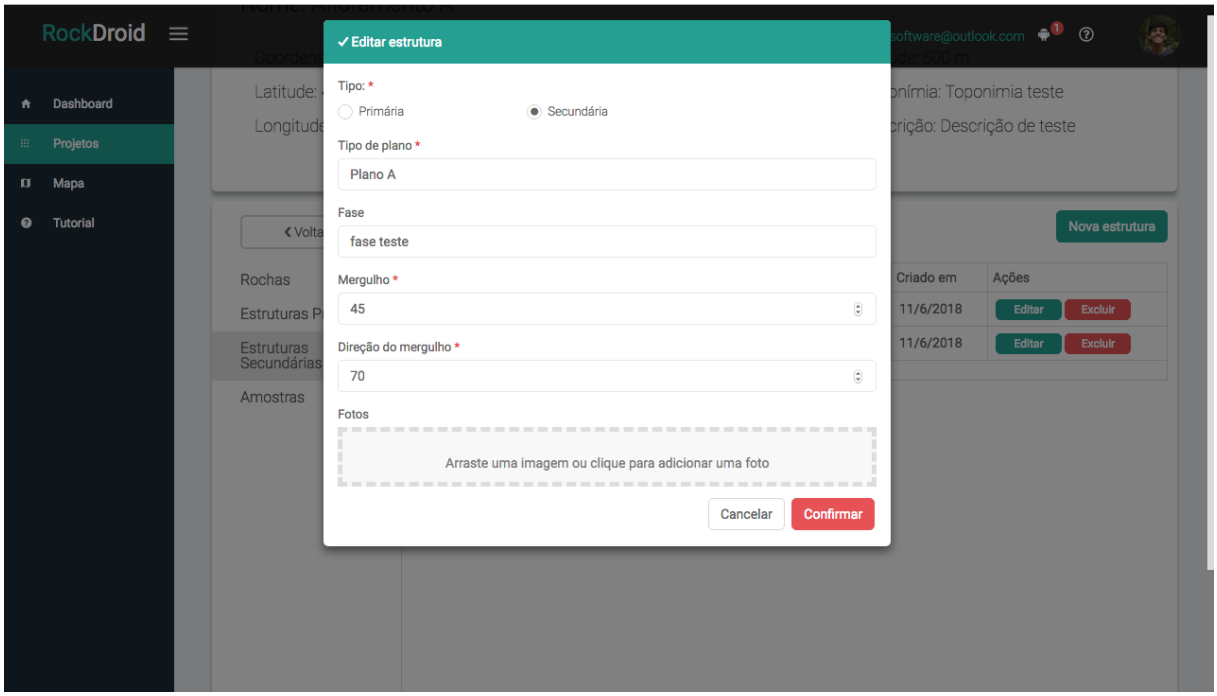


Figura 4.50: Edição de estrutura.

4.3.7 Mapa de afloramentos

O mapa de afloramentos permite visualizar todos os afloramentos criados pelo usuário a partir de pontos em um mapa do OpenStreetMap (Figura 4.51). Para facilitar a buscar dos elementos foram criados duas funcionalidades que são descritas a seguir:

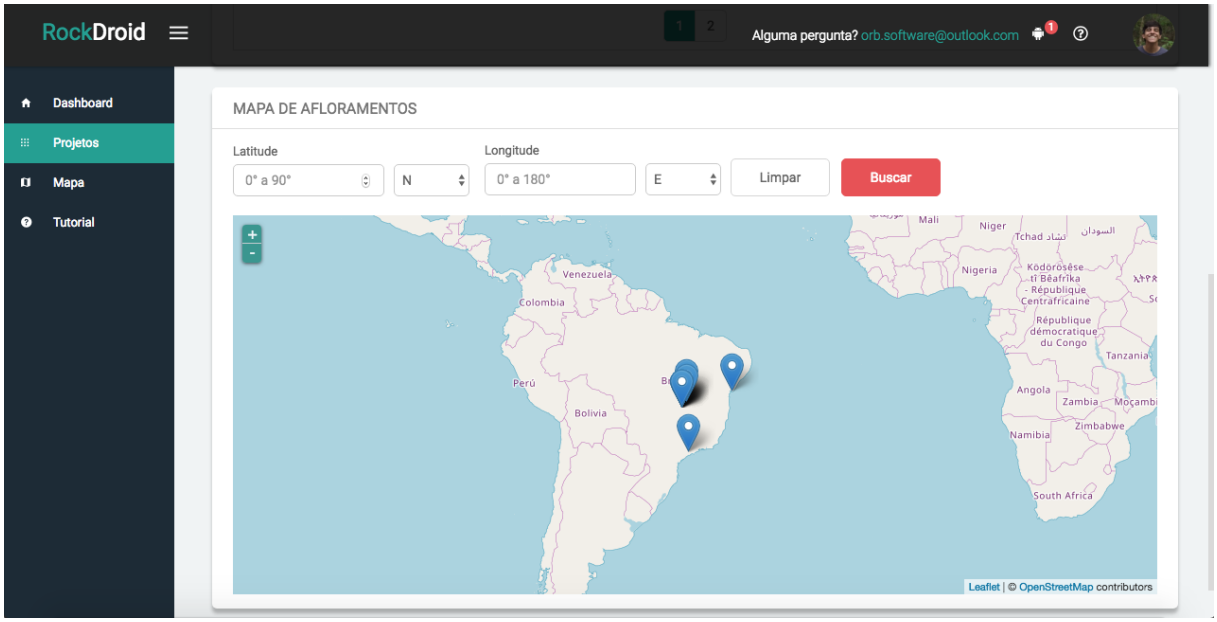


Figura 4.51: Mapa de afloramentos.

Busca por Lista de Afloramentos

Uma lista com todos os afloramentos (Figura 4.52) do usuário é exibida, sendo que quando o usuário selecionar um elemento da lista, o mapa é movimentado para a coordenada geográfica correspondente (Figura 4.53).

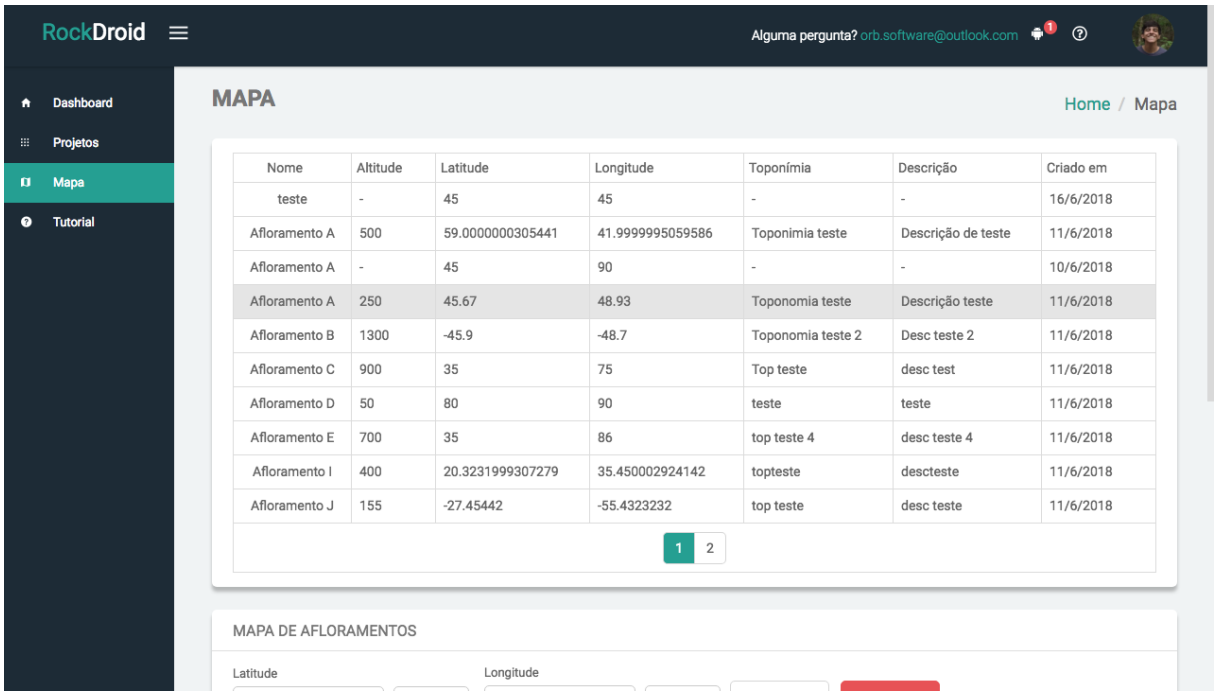


Figura 4.52: Busca no mapa por afloramento.

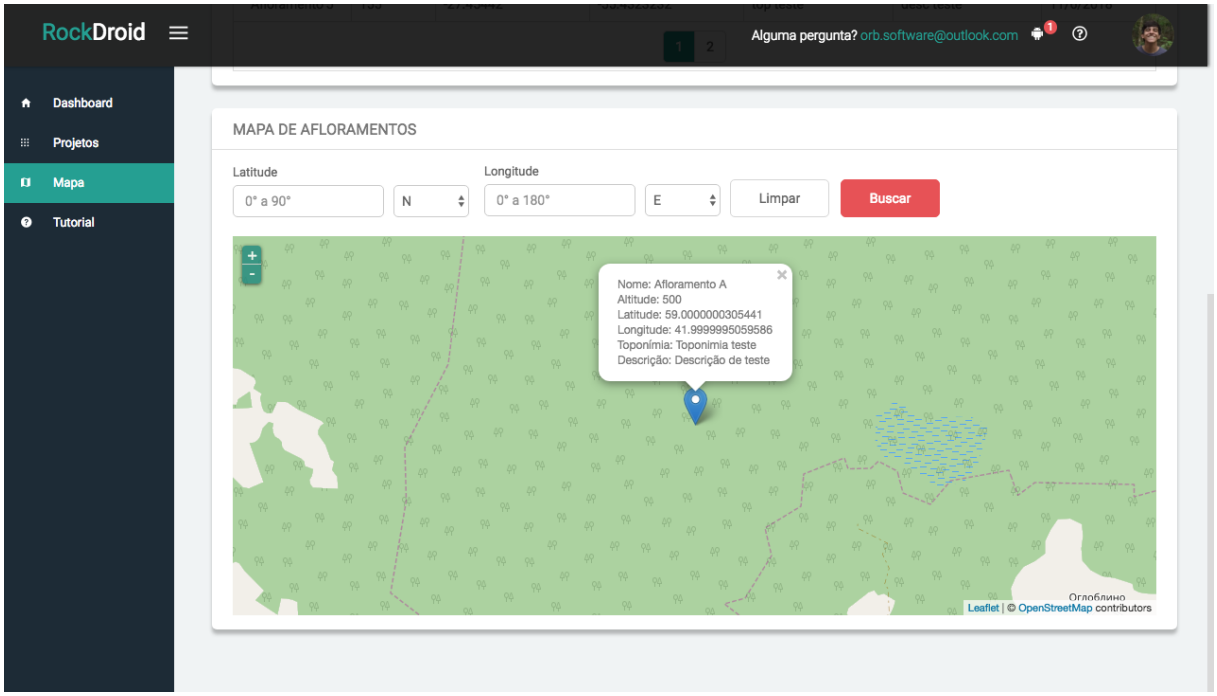


Figura 4.53: Seleção de afloramento no mapa.

Busca por Latitude, Longitude

A busca por latitude e longitude no mapa de afloramentos pode ser vista na Figura 4.54, e apresenta um formulário em que o usuário pode inserir uma latitude e uma longitude. Dessa forma, ao clicar no botão de (buscar) o mapa é redirecionado para a latitude e longitude inserida.

Os dois modos permitem que o usuário encontre os afloramentos facilmente, e evite perda de tempo navegando pelo mapa desnecessariamente. Ao clicar em um dos afloramentos no mapa, uma lista de informações sobre o afloramento é exibida sobre o marcador que indica o elemento, facilitando a identificação dos dados de cada afloramento.

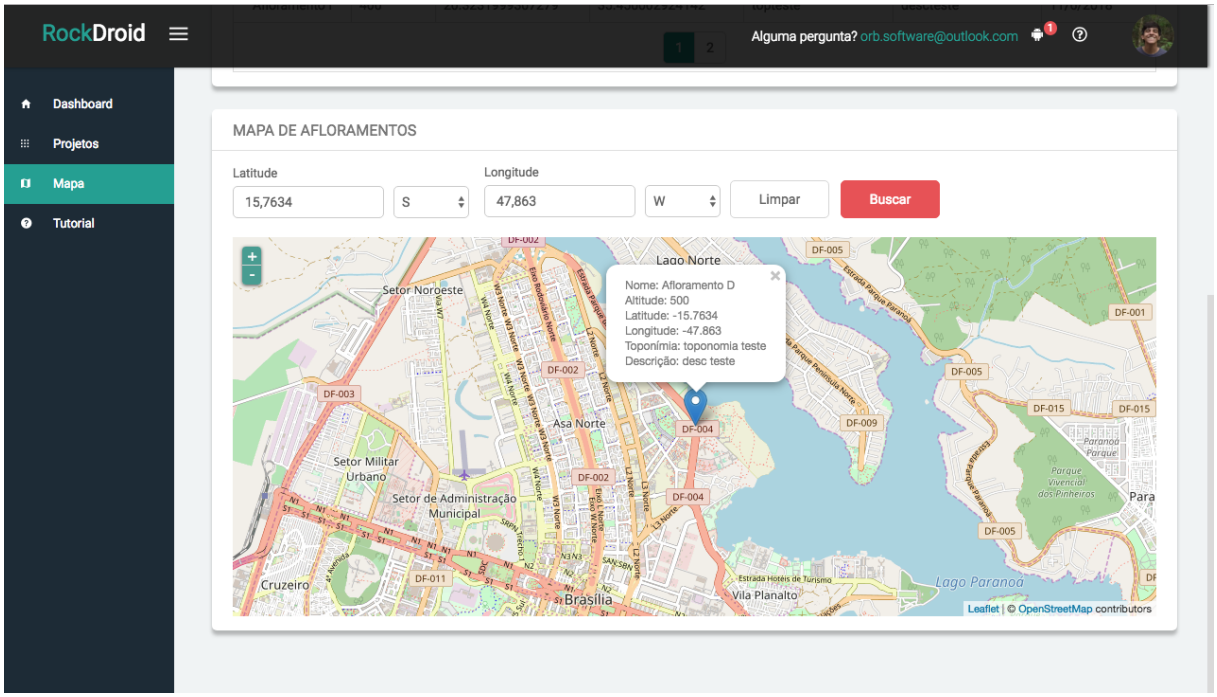


Figura 4.54: Busca no mapa por coordenada geográfica

4.3.8 Dashboard

O Dashboard é responsável por mostrar dados que podem vir a ser úteis para o aluno ou professor utilizando o sistema, e que são calculados a partir dos dados inseridos pelo usuário no sistema, seja pelo aplicativo, ou pelo sistema web.

Números de Projetos, Etapas, Afloramentos

O painel exibido na Figura 4.55 mostra o número de projetos, etapas, afloramentos, rochas e estruturas criadas. Além disso também exibe para o professor o número e o nome dos alunos utilizando o sistema.

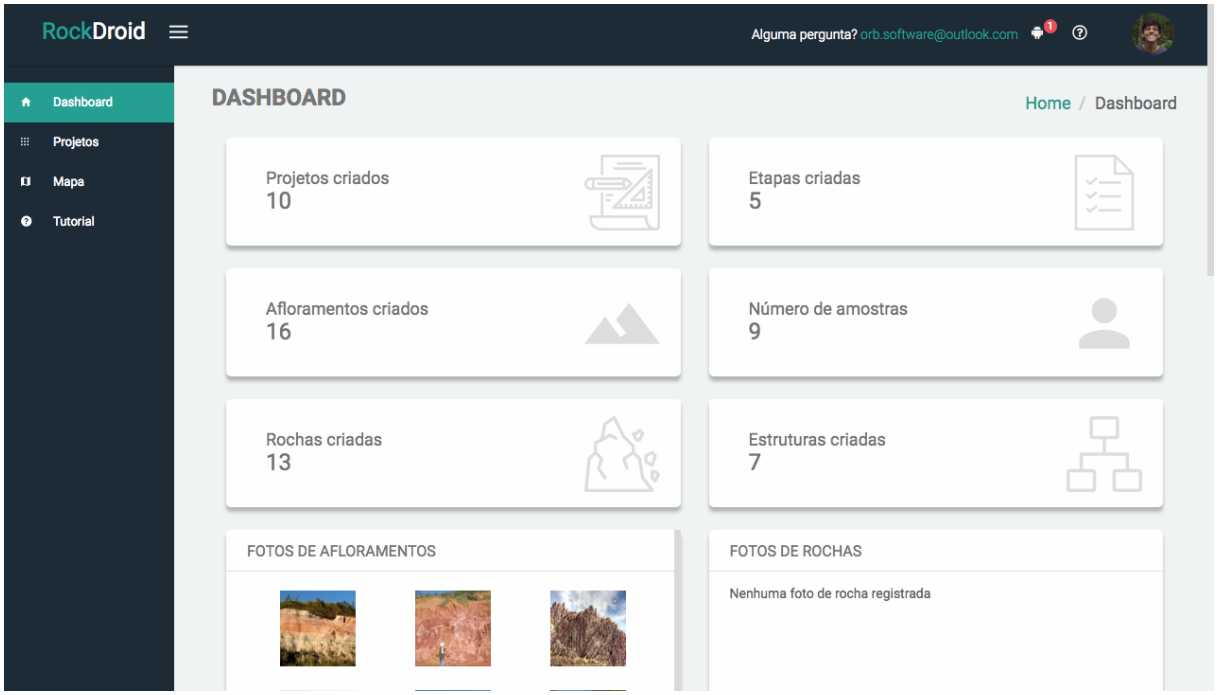


Figura 4.55: Dashboard.

Fotos de Rochas, Afloramentos, Estruturas e Amostras

Listas com todas as fotos de rochas, afloramentos, estruturas e amostras adicionadas pelo usuário do sistema são exibidas no *dashboard*. O usuário pode fazer o *download* da foto que se deseja obter apenas clicando no item da lista. O *download* se inicia automaticamente para uma imagem em formato *.png* do mesmo tamanho da imagem inserida pelo sistema web ou aplicativo (Figura 4.56).

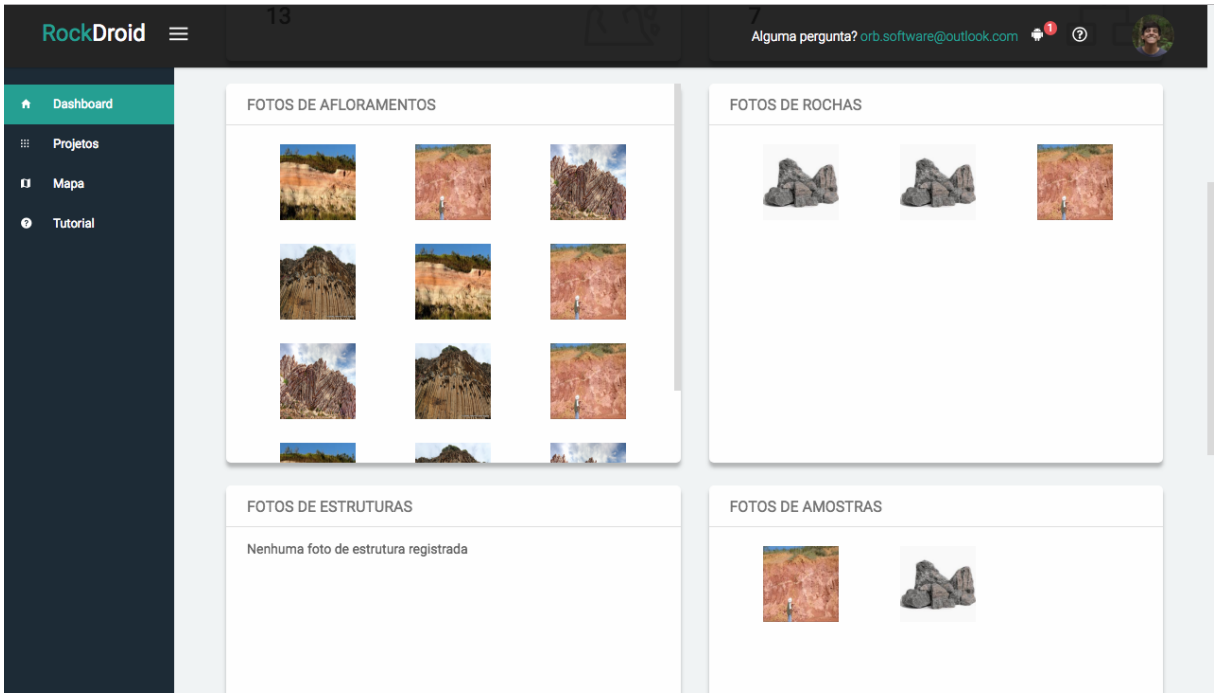


Figura 4.56: Seção de fotos do dashboard.

Etapas por Unidade Federativa

A Figura 4.57 exibe o gráfico de um mapa do Brasil com todas as Unidades Federativas, e o número de etapas criadas para cada UF.

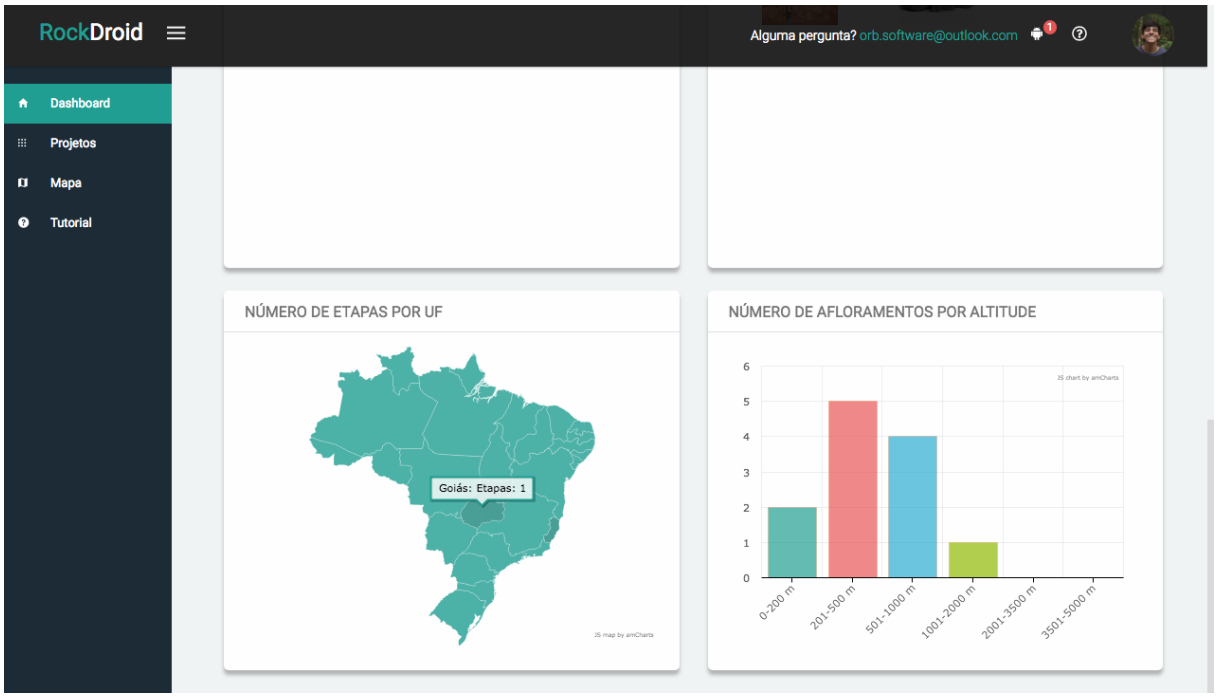


Figura 4.57: Gráfico de etapas por Unidade Federativa

Afloramentos por Altitude.

Na Figura 4.57 também é exibido um gráfico de barras com o número de afloramentos existentes para cada intervalo de altitude. Os intervalos de valores são divididos em cinco categorias, sendo elas de 0-200(m), 201-500(m), 501-1000(m), 1001-2000(m), 2001-3500(m) e acima de 3500(m).

4.3.9 Modo Aluno e Professor

Existem no sistema dois tipos de conta de usuário os quais são o tipo aluno e professor. A conta do tipo aluno pode utilizar todas as mesma funcionalidades do aplicativo, mas apenas visualiza os dados que ele mesmo inseriu. A conta do tipo professor consegue ter acesso aos dados obtidos a partir de todos os alunos do sistema.

A aba projetos para o caso em que uma conta do tipo professor fez *login* no sistema, também exibe o nome do aluno que criou o projeto. O objetivo do modo professor é fornecer uma forma de o pesquisador chefe ter acesso e verificar os dados obtidos por seus alunos durante a saída de campo.

Ativando Modo Professor

Para ativar o modo professor deve ser pedido ao usuário com conta de nível de administrador do sistema para que entre no sistema de administração do banco de dados e defina a variável de usuário is_teacher como verdadeira na seção da tabela *users* (Figura 4.58).

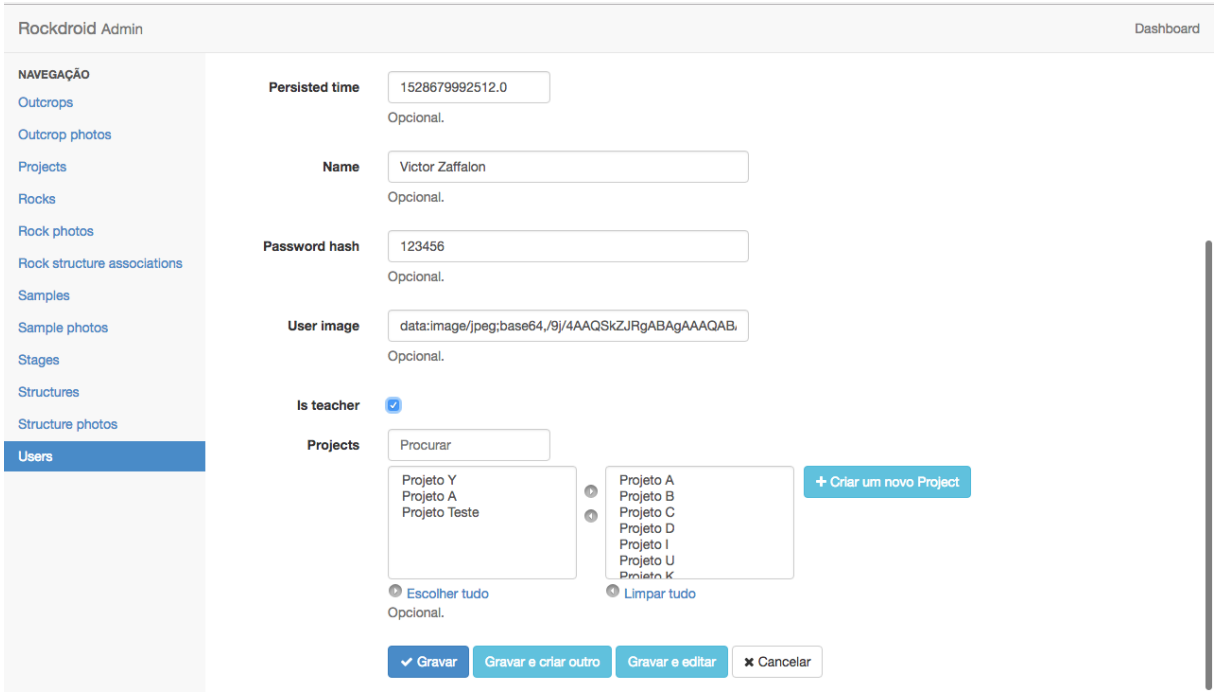


Figura 4.58: Ativando modo professor.

Dashboard

O Dashboard (Figura 4.59) exibe dados diferentes com base no tipo de conta que acessou o sistema. Para conta do tipo professor, os dados exibidos são relativos a todos os alunos cadastrados, diferentemente, do acesso de conta em modo aluno. Nesse caso o Dashboard só exibe dados relativos aos adicionados pelo próprio aluno.

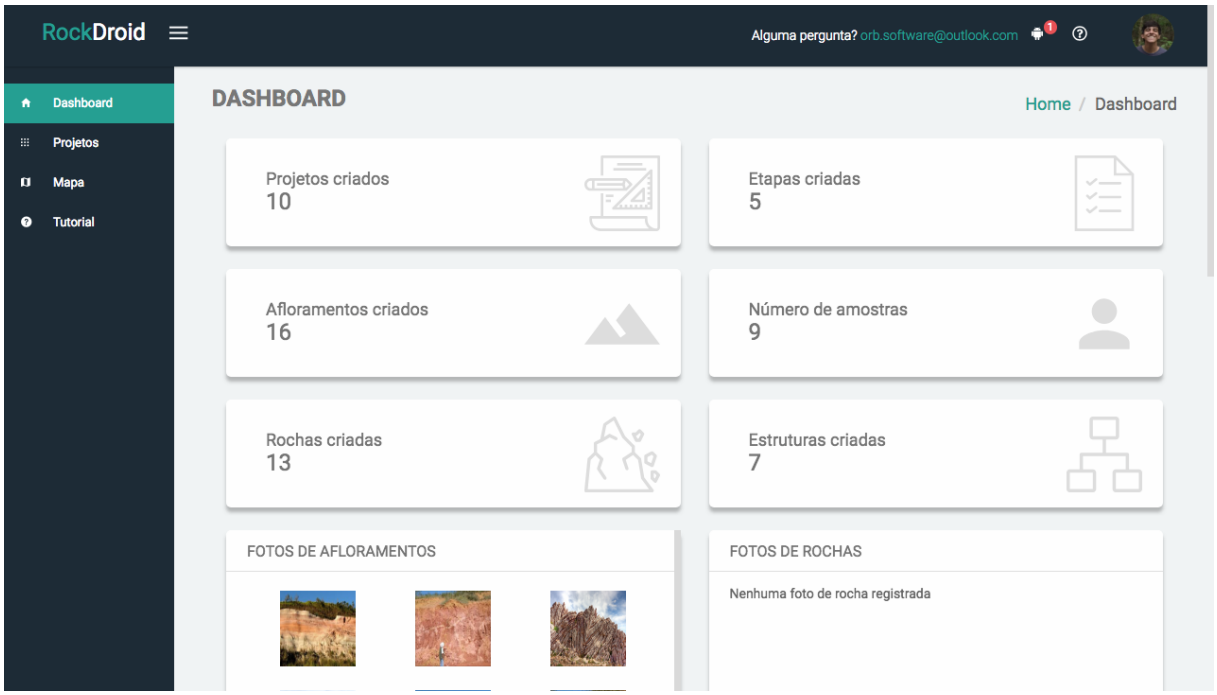


Figura 4.59: Dashboard de aluno.

4.3.10 Teste de Validação de Formulários

Nesta etapa são feitos alguns teste relativos a validação dos campos dos formulários do sistema. Esta etapa do desenvolvimento do sistema tem como objetivo impedir que valores inválidos, ou não esperados, sejam inseridos no formulário, ajudando a garantir a consistência dos dados obtidos pelos alunos e inseridos no banco de dados.

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

- Projeto:
  - Nome: campo obrigatório.
- Etapa de campo:
  - Nome: campo obrigatório.
  - Data de início: campo obrigatório.
- Afloramento:

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

- Rocha:
  - Nome: campo obrigatório.
  - Tipo de rocha: campo obrigatório.

- Amostra:
  - Nome: campo obrigatório.

- Estrutura primária:
  - Descrição: campo obrigatório.

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

Para os atributos que são obrigatórios, os testes consistiram em deixar os campos dos formulários vazios e fazer o envio da requisição. O Sistema Web não permitiu que os formulários fossem salvos, exibindo mensagens de erro para os campos em branco com a mensagem (Campo obrigatório) até que eles fossem preenchidos. Para os atributos numéricos que possuem limites inferiores e superiores definidos, os testes envolveram a inserção de números fora do intervalo de cada campo. O Sistema Web exibiu mensagens de erro específicas para cada atributo, informando ao usuário quais eram os valores permitidos. Por fim, o formulário é salvo apenas quando valores entre os intervalos válidos foram inseridos.

Nas Figuras 4.60 e 4.61 são mostrados exemplos de validações do formulário de adicionar afloramentos. Nas imagens são exibidas as mensagens de erro referente a campos vazios e aos campos de latitude e longitude fora do intervalo desejado.

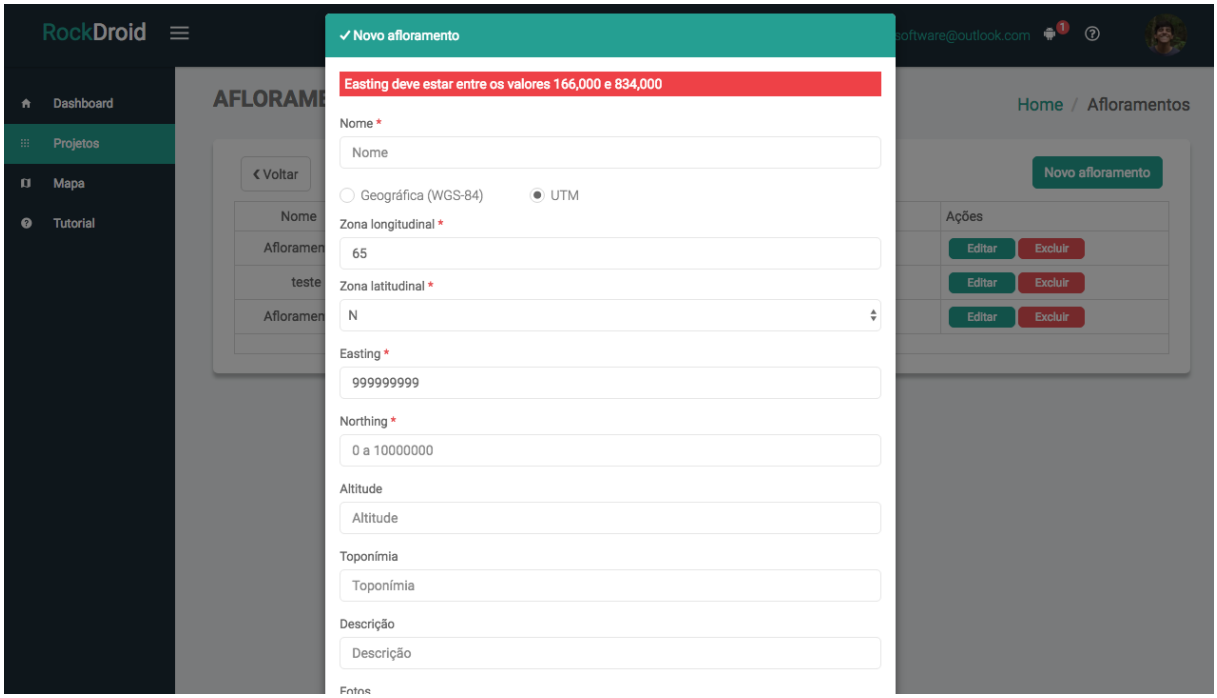


Figura 4.60: Validação de adição de afloramento.

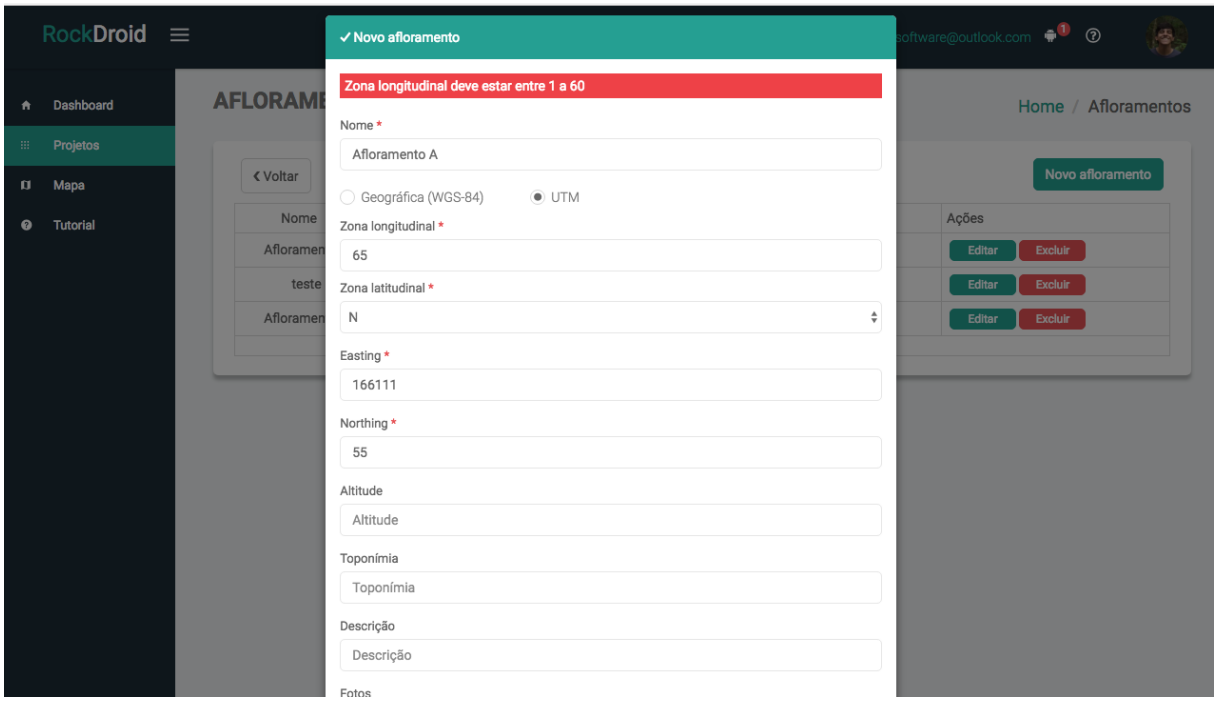


Figura 4.61: Validação de adição de afloramento B.

Request HTTP Inválido

Todos as validações feitas pelos formulários do sistema Web são, primeiramente implementadas, no *Back-End*, de forma a sempre garantir a consistência dos dados inseridos.

Caso uma requisição HTTP seja enviada com dados incorretos, a requisição é invalidada e seu resultado retorna com uma resposta e código de erro.

A Figura 4.62 representa um *request* do tipo POST em que o campo nome não foi enviado, a api na etapa de validação de campos identifica que o parâmetro não foi enviado, e retorna como resposta uma mensagem de erro com código 422 (Entidade não processada).

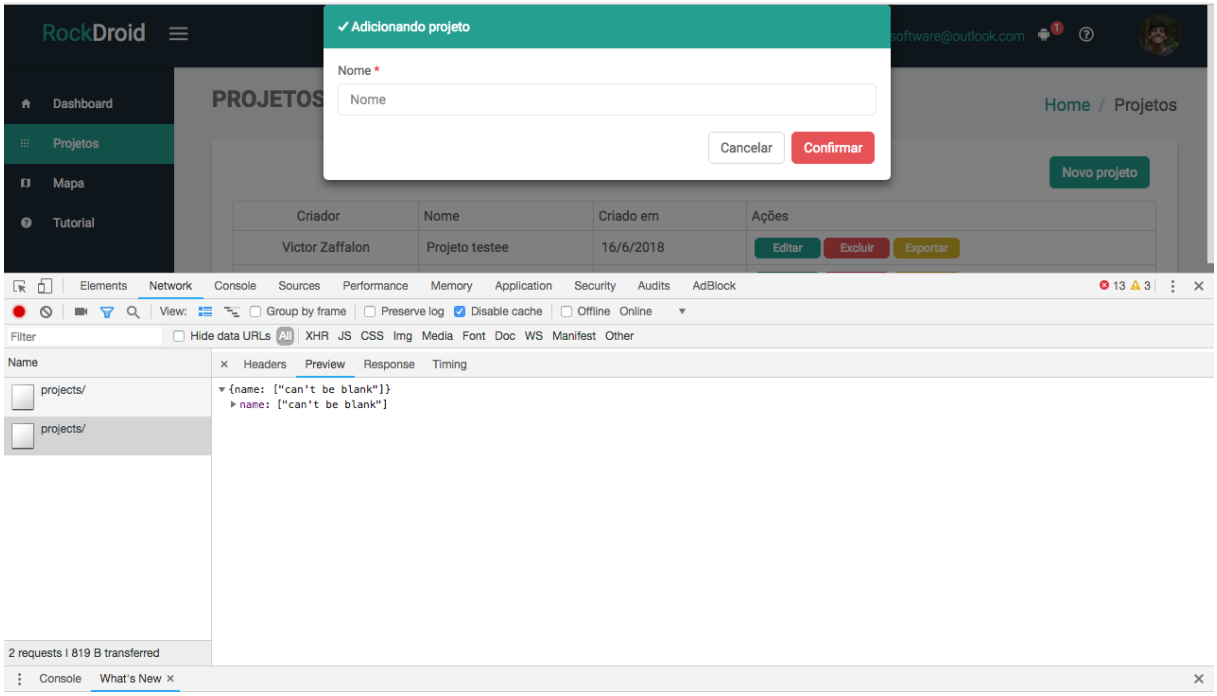


Figura 4.62: Requisição invalidada pela API.

4.3.11 Teste de Navegadores

Para validar a qualidade de um sistema Web é preciso garantir que o sistema funcione nos navegadores mais utilizados da época em que o sistema foi desenvolvido. De acordo com [6], os navegadores mais utilizados no ano de 2018 são o google chrome, o safari, o internet explorer e o firefox. Como uma das etapas de finalização do sistema, foram feitos testes de funcionamento do sistema para esses quatro *browsers*, e avaliado como o sistema funciona para cada um desses navegadores.

- Google Chrome:
  - Toda as funcionalidades funcionaram corretamente, considerando que durante o processo de desenvolvimento a plataforma foi desenvolvida e testada utilizando esse navegador, é o comportamento esperado.
- Firefox e Internet Explorer:
  - Funcionalidades e elementos de layout tiveram o comportamento esperado.
  - Alguns elementos de CSS da página não ficaram iguais, como pode ser visto na Figura 4.63.

- Safari:
  - Sistema de autenticação de requests falhou no Safari, pois ele não enviava o campo *Authorization* no cabeçalho de suas requisições.
  - Alguns elementos de CSS da página não ficaram iguais como pode ser visto na Figura 4.63.

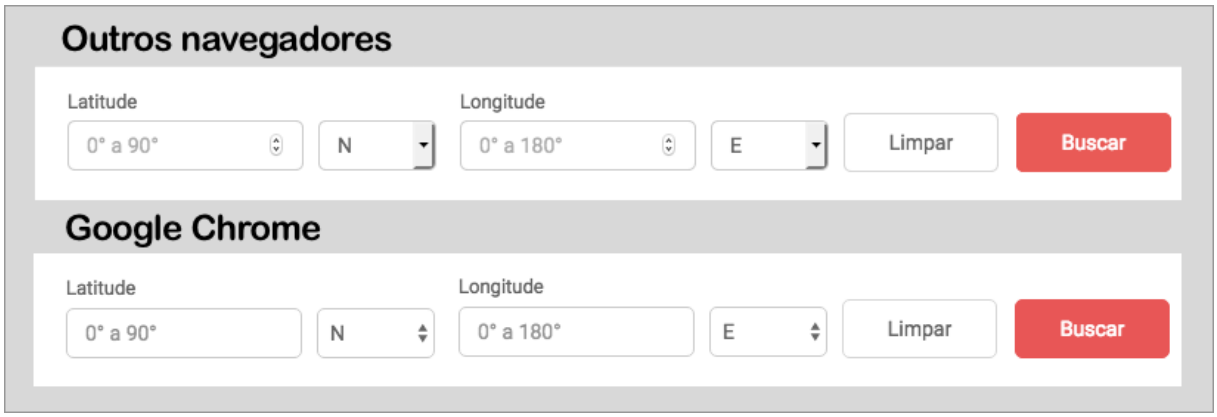


Figura 4.63: Comparação de navegadores.

Capítulo 5

Conclusão

O RockDroid é uma solução para facilitar o trabalho de geólogos por meio de uma interface Web, e um aplicativo que padroniza a coleta e facilita a análise dos dados. A união das ferramentas que podem ser providas por um aplicativo mobile (câmera fotográfica, GPS, mapa, bloco de notas, etc.) em conjunto com as ferramentas providas por um SIG Web permitiram criar uma solução completa para auxílio de profissionais e estudantes pesquisadores.

O SIG Web desenvolvido busca atender os três principais objetivos de um SIG, dando suporte aos profissionais da geociência para a análise espacial de fenômenos, a criação de um banco de dados geográfico e a produção de mapas a partir dos dados. O SIG Web facilita o acesso aos dados geográficos por qualquer dispositivo que tenha acesso a um navegador Web e auxilia em quase todas as etapas do processo de conversão de dados geográficos em informações úteis para análise de dados e tomadas de decisão futuras durante uma saída de campo por pesquisadores.

O desenvolvimento do SIG Web utilizando as tecnologias mais novas de desenvolvimento permitiu criar um sistema completo, fluido e seguro com um *design* agradável e intuitivo. Por utilizar *frameworks* consolidados no mercado para desenvolvimento do *Back-End* e *Front-End*, foi possível implementar rapidamente varias funcionalidades sem comprometer o padrão do projeto, e mantendo a qualidade do código da aplicação.

A ferramenta *Ruby On Rails* facilitou criar uma API que realiza todas as operações de (C_R_U_D) no banco de dados, assim como o uso do sistema de gemas da linguagem Ruby permitiu a utilização de bibliotecas que automatizam e facilitam o processo de integração do servidor com o banco de dados PostgreSQL. O *Ruby On Rails* contribui com a criação das tabelas do banco de dados, assim como a validação dos dados recebidos por formulários, a transferência de dados por JSON entre cliente e servidor e o controle da linha do tempo de alterações do banco por meio das *Migrations*.

A ferramenta *AngularJs* permitiu criar um projeto SPA utilizando a linguagem *javascript* operando sobre um servidor NodeJs que realiza requisições http para obter dados do servidor e exibir em formato de tabelas, gráficos e mapas para o usuário. O uso da biblioteca *javascript* Leaflet permitiu renderizar um mapa do OpenStreetMap iterativo que exibe os afloramentos criados pelos pesquisadores.

A construção da transmissão de dados entre aplicativo, *Back-End* e *Front-End* com base em um modelo RESTful, facilitou manter a mesma estrutura de comunicação entre os três elementos, garantindo a segurança dos dados e permitindo que o serviço criado

fosse simples de usar e manter, além de permitir sua ampliação e facilitar com que outros serviços que seguem a estrutura RESTful sejam futuramente integrados ao sistema sem grandes barreiras.

Dessa maneira, futuramente para melhoria do sistema faz-se necessária a implementação de um algoritmo de sincronização que atue em duas direções. A sincronização da maneira que opera no estado atual do projeto apenas envia dados do aplicativo RockDroid para o banco de dados central, caso algum dado seja inserido pelo painel Web no banco de dados, ao acessar o aplicativo e realizar a sincronização com o banco central, nenhum dado novo criado a partir do SIG será recebido pelo aplicativo.

Ampliar e melhorar a funcionalidade do gerenciamento de projetos e equipes também deve ser uma prioridade para trabalhos futuros. Criar um método de participação de projetos por meio de convites que possam ser enviados tanto pelo sistema Web, tanto quanto pelo aplicativo RockDroid. Os usuários do tipo aluno do sistema e do aplicativo procurariam os projetos e solicitariam participação. Os usuário com a conta do tipo professor poderiam aprovar, recusar participações ou enviar convites para os alunos por meio de um email, ou notificação *push* no aplicativo. Os alunos iriam ficar responsáveis apenas por enviar dados de afloramentos coletados, enquanto os administradores do projeto seriam os únicos com autorização para alterar e administrar os projetos e etapas.

Além disso, faz-se necessário discutir com os pesquisadores sobre novas formas de representar os dados coletados por meios de gráficos e filtros nas tabelas de dados do SIG com o objetivo de ampliar o *Dashboard* e facilitar a análise de cada etapa da saída de campo.

Referências

[1] Sobre o OpenStreetMap. Acessado em 01/8/2018. 5

[2] NodeJs - About. https://nodejs.org/en/about/, 2010. 20, 21

[3] Ruby Gems. http://guides.rubygems.org/, 2016. 17

[4] Angular Js. http://guides.rubyonrails.org/, Janeiro 2017. Acessado em 16/9/2017. 22, 23, 24, 25

[5] Ruby on Rails Guides. http://guides.rubyonrails.org/, Janeiro 2017. Acessado em 16/9/2017. 14, 15, 16, 17

[6] Estatísticas do uso de navegadores web em 2018. https://www.w3counter.com/globalstats.php, 2018. Acessado em 23/5/2018. 71

[7] Moment Js Documentation. https://momentjs.com/docs/, Janeiro 2018. Acessado em 19/05/2018. 44

[8] Wikipedia, 2018. Acessado em 01/8/2018. 5

[9] Renata Cristina Machado Nunes Bernardo Augusto Pereira. RockDroid - Uma Arquitetura para Coleta de Dados Geológicos . Julho 2016. 11, 12

[10] Gene Bylinsky. *Managing with electronic maps*. 1. edition, 1989. 3

[11] R. Fielding, UC Irvine, J. Gettys, and J. Mogul. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, The Internet Society, Julho 1999. 6

[12] Câmara G. *Representação computacional de dados geográficos*. 1. edition, 2007. 3, 4

[13] Erich Gamma. *Padrões de Projetos*. Bookman, 1. edition, 2000. 15

[14] Brad Green and Shyam Seshadri. *AngularJs*. O Reilly, 1. edition, Julho 2016. 22

[15] Shunfu H. *Web-based multimedia gis for the analysis and visualization of spatial environmental database.* ESRI Press, 1. edition, 2002. 3, 5

[16] David Heinemeier Hansson. The Rails Doctrine. http://rubyonrails.org/doctrine/, Janeiro 2016. Acessado em 16/9/2017. 14, 17

[17] James F. Kurose and Keith W. Ross. *COMPUTER NETWORKING - A Top-Down Approach*. Addison Wesley, 6. edition, 2000. vii, 7, 8, 9, 10

[18] DEEP MEDHI and KARTHIK RAMASAMY. *Network Routing - Algorithms, Protocols, and Architectures* . Morgan Kaufmann Publishers, 1. edition, 2007. 9

[19] Michael S. Mikowski and Josh C. Powell. *Single Page Web Applications.* Manning, 1. edition, Setembro 2013. 18, 19

[20] Sun J. Fu P. *Web GIS: Principles and Applications.* ESRI Press, 1. edition, 2010. 5, 6

[21] Ferreira K. R. Queiroz G. R. *Tutorial sobre bancos de dados geográficos.* 1. edition, 2006. 4

[22] Alex Rodriguez. RESTful Web services: The basics. *IBM developerWorks*, February 2015. 6, 7, 8

[23] Sam Ruby and David Bryant Copeland. *Agile Web Development with Rails 5.1.* Pragmatic Bookshelf, 2. edition, 2005. 14, 15, 16, 17

[24] Santos M. S. *Sistemas de informações geográficas: Elementos para o desenvolvimento de bibliotecas digitais geográficas distribuidas.* 1. edition, 2006. 4

[25] Steve Vinoski Stefan Tilkov. Node.js: Using JavaScript to Build High-Performance Network Programs. , Novembro 2010. 20, 21

[26] Bressan T. *Desenvolvimento e integração de um ambiente sigweb com ferramentas de software livre.* . 1. edition, 2010. 5

[27] Andrew S. Tanenbaum and Robbert Van Renesse. Distributed operating systems. *ACM Comput*, pages 24,25, Julho 1985. 8, 9, 10

[28] Dave Thomas. *Programming Ruby 2.0: The Pragmatic Programmers' Guide.* Pragmatic Bookshelf, 2. edition, julho 2013. 14

[29] Murilo Zaffalon. RockDroid - Rede Ad-Hoc . junho 2018. 11