UMA API PARA GERENCIAMENTO E NOTIFICAÇÃO DE EVENTOS EM APLICATIVOS.

L. R. A. Assis[1, *]; E.R. Zorzal[1]

1 Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo
Avenida Cesare Mansueto Giulio Lattes, 1201 – Eugênio de Melo, São José dos Campos/SP, CEP.: 12247-014, Brasil.
Telefone: (12) 3924-9500

* rocali@outlook.com

RESUMO: Este trabalho teve como objetivo apresentar um sistema colaborativo para gerenciamento de eventos, composto por uma *API* (*Application Programming Interface*) e um aplicativo móvel. A *API* é responsável por gerenciar os dados do sistema, registrar as localizações e preferências do usuários, receber os registros de eventos, gerenciar quais usuários serão notificados, receber registros que validam ou denunciam um evento criado e apresentar uma página *web* administrativa para gerenciamento dos dados e monitoramento dos eventos. O aplicativo móvel para gerenciamento de desastres naturais foi utilizado como estudo de caso para a criação do sistema e é um exemplo de aplicativo consumidor da *API* que permite ao usuário ver incidentes em um mapa, registrar um incidente e receber notificações baseado em sua localização ou áreas de interesse. A arquitetura da *API* permite gerenciar eventos de forma genérica sendo capaz de atender aplicativos com diferentes finalidades, mas que tenham a necessidade de utilizar o mesmo recurso para gerenciamento e notificação de eventos.

PALAVRAS-CHAVE: api; aplicativo; notificações; gerenciamento de eventos.

ABSTRACT: This project aimed to create a collaborative system for event management, composed of an API (Application Programming Interface) and a mobile application. The API is responsible for managing system data, registering user locations and preferences, receiving event registrations, managing which users will be notified, receiving requests that enforce or denouncing an event created, and rendering a admin web page for data management and monitoring events. The mobile application for natural disaster management was used as a case study for the creation of the system and is an example of an API consumer application that allows the user to view incidents on a map, record an incident and receive notifications based on their location or areas of interest. The API architecture allows to manage events in a generic way, being able to serve applications with different purposes but having the same need to use the resource for event management and notification.

KEYWORDS: api; application; notifications; event managemenent.

1. INTRODUÇÃO.

Cameron et al. (2012) definem que a coordenação de crise é o processo realizado por organizações e indivíduos com o intuito de reduzir a experiência dos impactos nas comunidades expostas ao perigo durante incidentes reais. Em grandes crises, entender o impacto do incidente é um fator crítico para o sucesso na recuperação de segurança e serviços essenciais. Segundo Mehta, Voisard e Müller (2013), os atuais métodos de alerta para eventos, como mensagens de texto ou

anúncios via televisão e rádio têm deficiências em definir a região ou grupos de pessoas a serem notificados.

Os *smartphones* têm tornado possível uma nova forma de acesso à informação sem esforço. Todos os dias estes dispositivos recebem uma variedade enorme de informação como e-mail, eventos de rede social e informações internas como alerta de bateria, alarmes, etc (MEHROTRA ET 6 AL., 2016). Sendo um importante recurso dos *smartphones*, as notificações têm tornado um novo canal de distribuição de informação, melhorando a cobertura e eficiência de alertas.

O objetivo desse trabalho foi desenvolver um sistema colaborativo genérico para gerenciamento de eventos e utilizar um aplicativo para alerta de desastres naturais como um exemplo de aplicativo consumidor da API (*Application Programming Interface*), que foi desenvolvida e apresentada neste trabalho. Na API foi definido um conjunto de padrões estabelecidos e documentados para que uma determinada aplicação ou sistema utilizem suas funcionalidades sem precisar saber detalhes da implementações de *software.* O aplicativo para desastre naturais é um estudo de caso para a criação do sistema e tem a finalidade de notificar usuários sobre desastres ou incidentes ocorridos. Com os dados de localização disponíveis dos usuários é possível identificar dinamicamente grupos de pessoas, e as notificações poderão ser direcionadas com informações personalizadas e instruções para cada grupo definido no sistema (MEHTA; VOISARD; MÜLLER, 2013).

Sendo um sistema cooperativo, ambas as partes terão uma informação mais precisa do ocorrido, onde as autoridades poderão notificar os usuários ou grupo em uma determinada área, bem como os usuários terão a possibilidade de notificar as autoridades para terem um melhor entendimento do ocorrido e o impacto do incidente.

2. ESTRATÉGIAS PARA DESENVOLVIMENTO.

Para o desenvolvimento das aplicações foi utilizado um *Macbook* Pro com sistema operacional Mac OSX Sierra e um dispositivo móvel iPhone 6. Os testes da aplicação Android foram realizados pelo emulador. Os aplicativos móveis foram desenvolvidos usando o *framework* para criação de aplicativos híbridos *React Native* (EISENMAN, 2015). Assim, foi possível desenvolver os aplicativos a partir de apenas um projeto em *Javascript*, agilizando o desenvolvimento, facilitando a manutenção e trazendo excelentes resultados de performance. Foi utilizado também o *framework Ruby on Rails* (BÄCHLE; KIRCHBERG, 2007) para a criação da API, o gerenciador de banco de dados MySQL e JSON (*JavaScript Object Notation*) para comunicação com os aplicativos. Além dessas tecnologias também foi usado o *framework One Signal* (OneSignal, 2018) para o envio de notificações da API para os aplicativos.

O sistema desenvolvido foi dividido em duas partes: a aplicação móvel e a API. O aplicativo requisita e fornece informações a API, que é responsável por armazenar os dados no banco de dados, requisitar envio de notificação por *push* via *OneSignal* e apresentar a página *web* com o mapa para o monitoramento dos eventos.

A Figura 1 mostra um exemplo de fluxo do registro de incidentes (em azul) e recebimento de notificações nas proximidades do usuário ao incidente (em vermelho). A ilustração apresenta também a preferência em receber incidentes de um determinado bairro (em amarelo) ou cidade (em roxo).

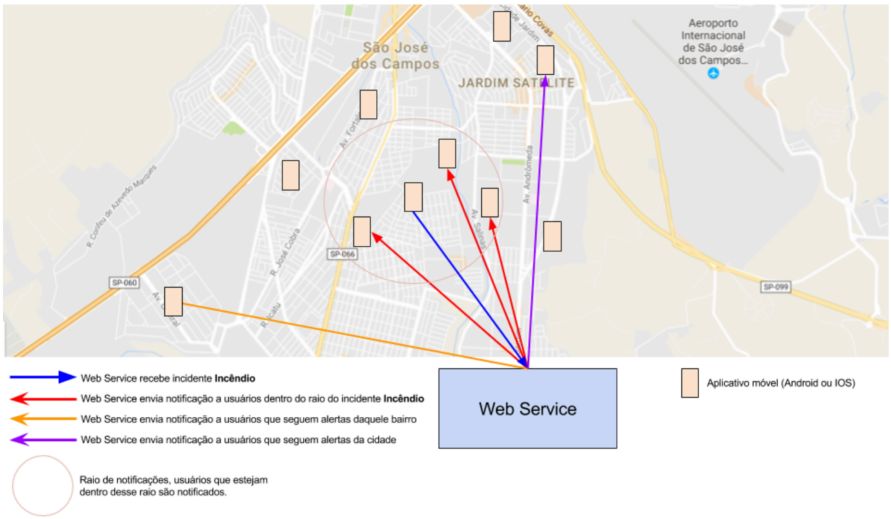


Figura 1. Exemplo do registro de incidentes e recebimento de notificações.

A Figura 2 apresenta de forma resumida como é a arquitetura do sistema, onde as requisições do aplicativo são direcionas a API, que é responsável por gerenciar os dados como banco de dados e retornar a informação ao aplicativo.

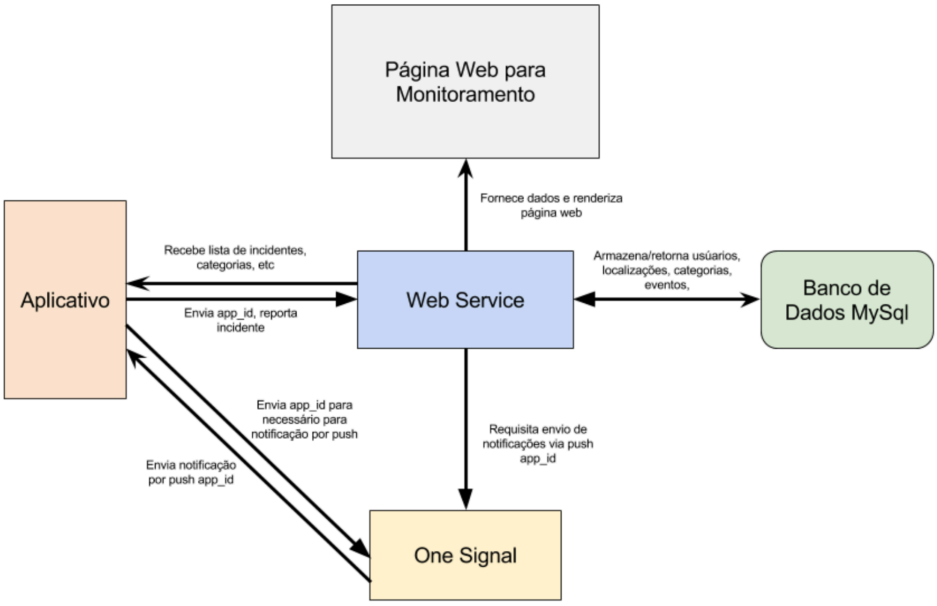


Figura 2. Arquitetura do sistema.

A aplicação móvel é a interface do usuário com o sistema e é responsável por fazer o *login* via o Facebook do usuário, apresentar a lista de incidentes e permitir que o usuário reporte os mesmos. Ao ver em detalhe um incidente o usuário pode denunciar aquele incidente como algo falso ou reforçar o relato, que trará maior credibilidade e o raio de usuários a serem notificados poderá ser incrementado.

O usuário tem também a opção de selecionar *tags* ou categorias de interesse para notificação. O sistema de *tags* é utilizado como uma estrutura em forma de árvore para gerenciar níveis de localizações. Por exemplo, uma *tag* do bairro Satélite tem como pai a *tag* São José dos Campos.

Portanto, ao notificar todas as pessoas que seguem a *tag* Satélite o sistema automaticamente notificará todos os usuário que seguem a *tag* pai São José dos Campos. Um exemplo do sistema de *tags* é apresentado na Figura 3 e tem como finalidade agrupar conjuntos de preferências por níveis.

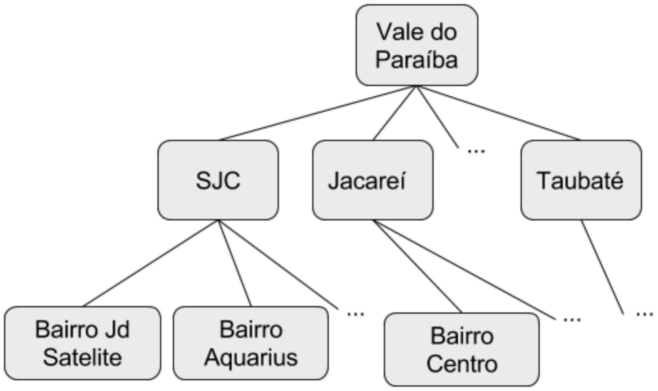


Figura 3. Exemplo do sistema de *tags* proposto.

Foi utilizado o *framework Ruby on Rails* para a criação da API, que é responsável por responder todas as requisições feitas pelo aplicativo, retornando dados ou registrando incidentes, bem como por gerenciar os usuários a serem notificados com base em suas localizações ou *tags* de interesse. A API também é responsável por requisitar o envio de notificação por *push* ao serviço *OneSignal*.

O sistema propõem uma página de monitoramento, onde é mostrado um mapa com todas os incidentes reportados, bem como a localização dos mesmos. A página tem uma lista com o histórico de notificações enviados aos usuários. Todas as requisições ao banco são feitas diretamente pela API que é responsável por administrar e repassar os dados.

3. RESULTADOS.

Foi desenvolvida uma API genérica para gerenciamento de eventos, capaz de atender as funcionalidades necessárias para o aplicativo de desastres naturais, desenvolvido como estudo de caso neste trabalho, bem como por atender outros tipos de aplicativos, com finalidades diferentes mas que tenham a necessidade de gerenciar eventos, registrando e notificando usuários.

A proposta da API é criar um servidor que fosse responsável por responder todas as requisições feitas pelo aplicativo, retornando dados ou registrando eventos, bem como por gerenciar os usuários a serem notificados com base em suas localizações ou *tags* de interesse. A API trata eventos de forma genérica, e notifica usuários baseado na proximidade ao evento ocorrido ou em locais de sua preferencia (como preferências por categoria ou *tag* que referencia uma determinada região).

Todas as funcionalidades relacionadas ao evento também são tratadas de forma genérica, qualquer sistema que utiliza a API tem as funcionalidades como registro de eventos associados a categorias e *tags*, registro de localizações do usuários, reforço ou denúncia de um evento de forma genérica.

Na Figura 4 é apresentado um exemplo que descreve o funcionamento da API, onde diferentes sistemas, tais como aplicativos para registros de desastres naturais, alertas de preços em supermercados e sistemas de segurança em condomínios consomem a API com diferentes finalidades mas utilizando a mesma estrutura.

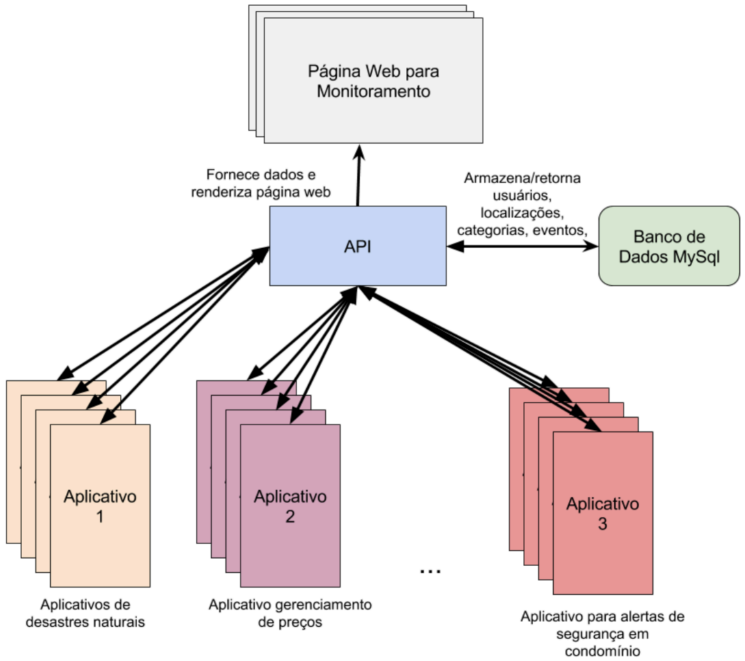


Figura 4. Arquitetura do sistema atendendo a múltiplos aplicativos.

Cada sistema tem um identificador único da API que é enviado na autenticação do usuário, com isso a API reconhece o usuário e o sistema específico que será autenticado.

No banco de dados (Figura 5) da API genérica os usuários (*User*), categorias (*Category*) e *tags* (*Tag*) são elementos únicos de cada sistema (*ApiUser*). Portanto, cada sistema que utiliza a API terá registros distintos de usuários (*User*), bem como todas as informações atreladas a ele, como eventos (Event), localizações (*Location*) e preferencias (*CategoryPreference* e *TagPrerence*). Para uma aplicação utilizar a API via painel administrativo pela *Web* é necessário estar previamente registrado no sistema, por meio de um usuário e senha. Cada aplicativo registrado pela API possui um *token* único (*api_token*), o qual é usado na identificação e na requisição de autenticação (*Login* e *Signup*) pela API.

Para garantir a segurança na utilização da API, as informações para autenticação deverão ser enviadas apenas no início de cada conexão. Assim, a API retorna um *token* de acesso para o usuário com validade de 24 horas e que deverá ser utilizado nas requisições posteriores no sistema. A autenticação utiliza o padrão JWT (*JSON Web Token*). JWT é um padrão (RFC 7519) que define de maneira compacta, autônoma e segura a transmissão de informações entre partes com um objeto JSON.

O método de codificação é responsável por criar *tokens* com base na identificação do usuário e no período de validade. O método de decodificação, por outro lado, aceita um *token* e tenta decodificá-lo usando a mesma chave usada na codificação.

O fluxo de requisições pode ser representado na Figura 6. Inicialmente, na autenticação com a API (*Login* e *Signup*) é necessário que seja incluído no cabeçalho da requisição o *token* do sistema para que cada usuário seja criado ou autenticado. Ao realizar a autenticação, a API retorna um *token* único (*auth_token*), válido por 24 horas, para o usuário. Esse *token* é a única identificação necessária que o sistema precisa para enviar as requisições para a API. Todas requisições requerem a adição do campo de autorização no cabeçalho para que a API autorize a requisição e retorne os dados. Por fim, com o *token* do usuário na requisição a API identifica o usuário do sistema e registra sua localização.

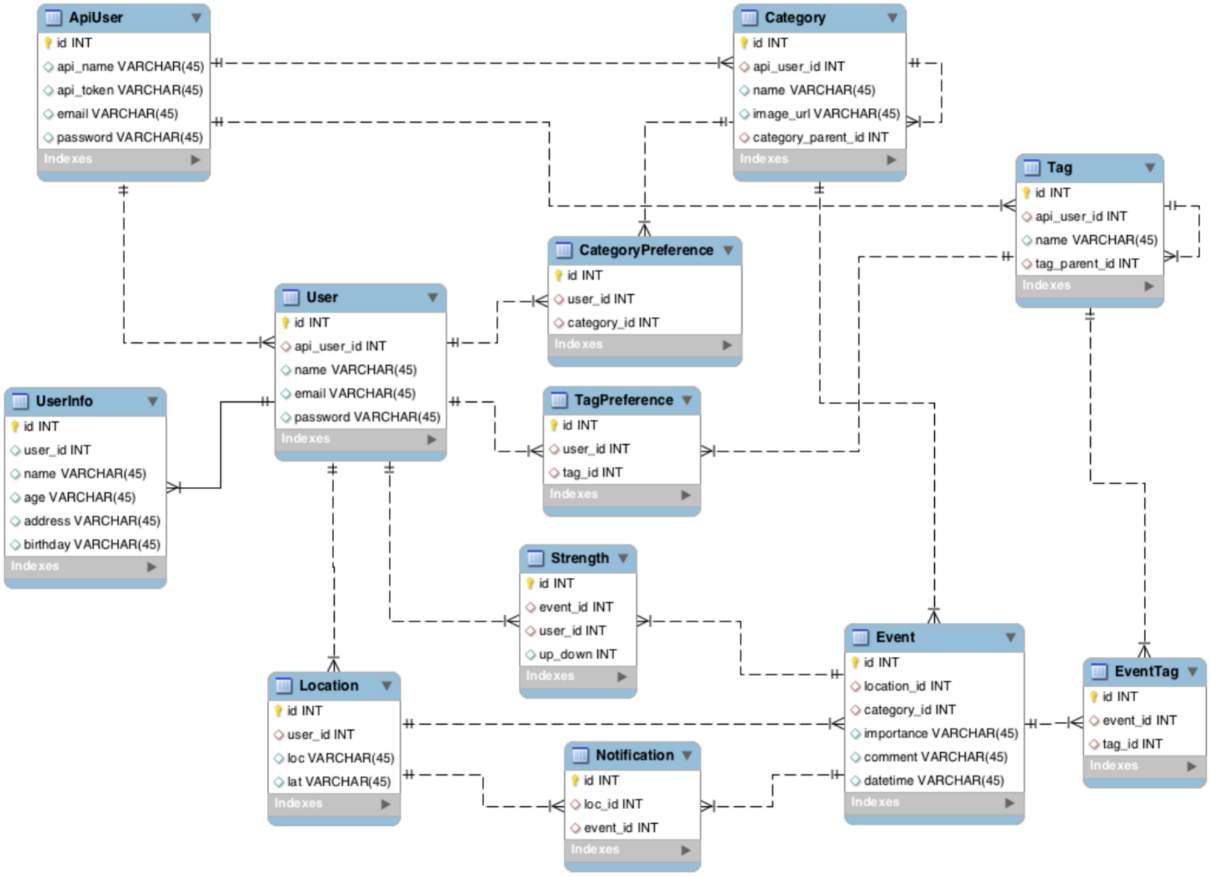


Figura 5. Estrutura do Banco de dados da API genérica.

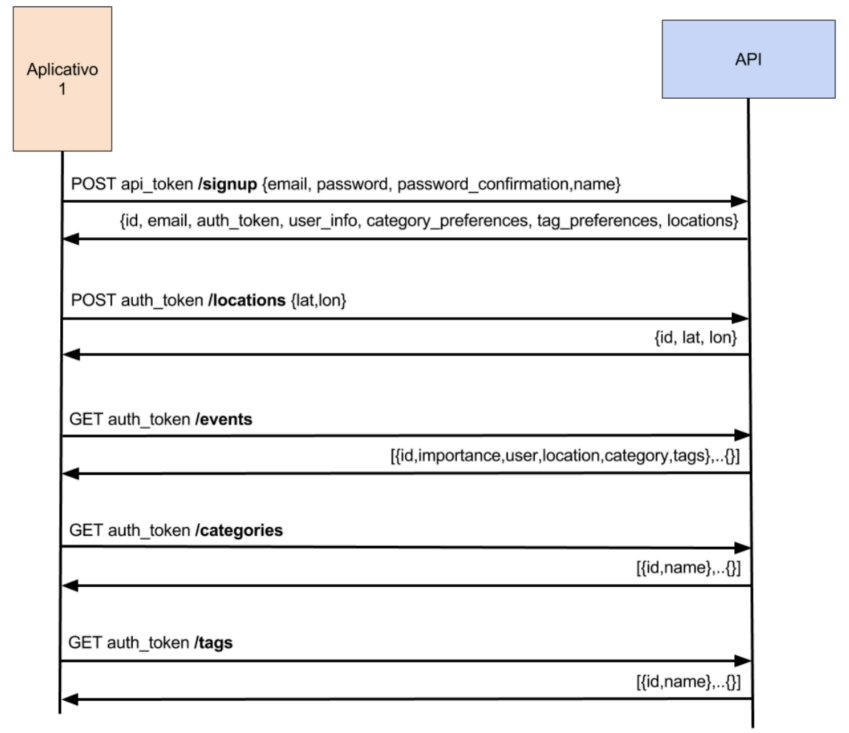


Figura 5. Exemplo de fluxo de utilização da API por um sistema.

A API garante acesso completo aos dados do sistema de gerenciamento. As requisições da API REST (*Representational State Transfer*) são apresentadas na Tabela 1. A API pode determinar se um sistema terá categorias fixas previamente definidas ou oferecer a possibilidade do usuário criá-las. Com esse formato, genérico é possível a utilização da API por diferentes tipos de aplicativos.

A seguir são apresentados alguns exemplos de sistemas/aplicativos que podem se beneficiar do uso da API.

a) Aplicativo para gerenciamento de desastres naturais: permite notificar a ocorrência de um desastre natural para pessoas que seguem uma determinada região ou estão próximas da ocorrência.
b) Aplicativo para alerta de preços: registrar e notificar ocorrências na mudança de preços dos produtos.
c) Aplicativo para segurança em condomínios: notificação de eventos, em tempo real e sua localização, aos moradores e interessados pela segurança do condomínio.

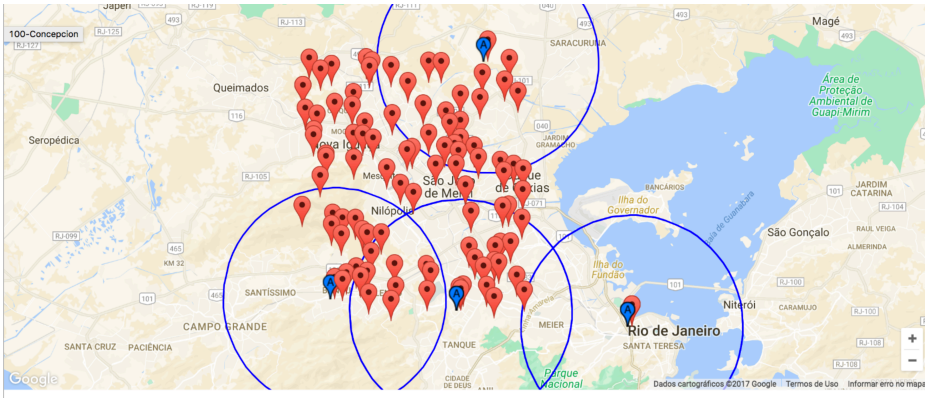
Tabela 1. Requisições da API REST.

Endpoint	Funcionalidade
POST /auth/login	Autentifica e retorna informações do usuário
POST /signup	Cadastra e retorna informações do usuário
POST /auth/logout	*Logout*
POST /strength	Reforça ou denuncia um evento
DEL /strength/:id	Deleta reforço ou denunca de um evento
POST /category_preferences	Adiciona categoria as preferências do usuário
DEL /category_preferences/:id	Remove categoria das preferências do usuário
POST /tag_preferences	Adiciona *tag* as preferências do usuário
DEL /tag_preferences/:id	Remove *tag* das preferências do usuário
GET /tags	Retorna *tags*
POST /tags	Cria uma nova *tag*
GET /tags/:id	Retorna *tag*
PUT /tags/:id	Edita uma *tag*
DEL /tags/:id	Deleta uma *tag*
GET /categories	Retorna categorias
POST /categories	Cria uma nova categoria
GET /categories/:id	Retorna categoria
PUT /categories/:id	Edita uma categoria
DEL /categories/:id	Deleta uma categoria
GET /locations	Retorna localizações do usuário
POST /locations	Cria uma nova localização do usuário
GET /locations/:id	Retorna localização do usuário
PUT /locations/:id	Edita uma localização do usuário
DEL /locations/:id	Deleta uma localização do usuário
GET /events	Retorna eventos
POST /events	Cria um novo evento
GET /events/:id	Retorna evento
PUT /events/:id	Edita um evento
DEL /events/:id	Deleta um evento

Para assegurar que os sistemas que utilizam a API não sejam prejudicados é extremamente importante manter uma estrutura de versionamento para gerenciar as mudanças da API ao longo do tempo. Foram definidos mecanismos no HTTP (*Hypertext Transfer Protocol*) que permitem servir diferentes versões de um recurso no mesmo URI (*Uniform Resource Identifier*). A API

desenvolvida implementa a negociação de conteúdo orientado por servidor onde o cliente define no cabeçalho HTTP qual versão da API deseja utilizar. Toda a estrutura do projeto foi desenvolvida pensando no versionamento para facilitar as atualizações e inclusão de novas versões.

O usuário monitor do sistema pode se autentificar pela API e acessar a página de monitoramento que mostra o mapa com todos os eventos próximos ocorridos (Figura 7).



Notificações

Notification Id	Event Id	Event Category	Event Created By	Notific Location Id	Alert to	Distance from event
60	15	Incendio	Gail	85	Pearl	5.32
61	15	Incendio	Gail	76	Caroline	5.43
62	15	Incendio	Gail	20	Willie	1.89
63	15	Incendio	Gail	1	Jackeline	2.45
64	15	Incendio	Gail	92	Kathleen	6.11

Figura 7. Página de monitoramento.

O mapa mostra todos os eventos que foram reportados na cor azul e todas as localizações dos usuários na cor vermelha. Dado um evento é mostrado o raio em azul onde todos os usuários nele incluídos foram notificados. Uma tabela é apresentada com todas as notificações que foram enviadas, mostrando qual evento gerou aquela notificação, qual usuário foi notificado e qual a distancia do usuário ao evento.

Foi criado também um painel administrativo (Figura 8), onde cada usuário da API pode acessar e gerenciar todos os dados do sistema como usuários, eventos, categorias, localizações, etc.

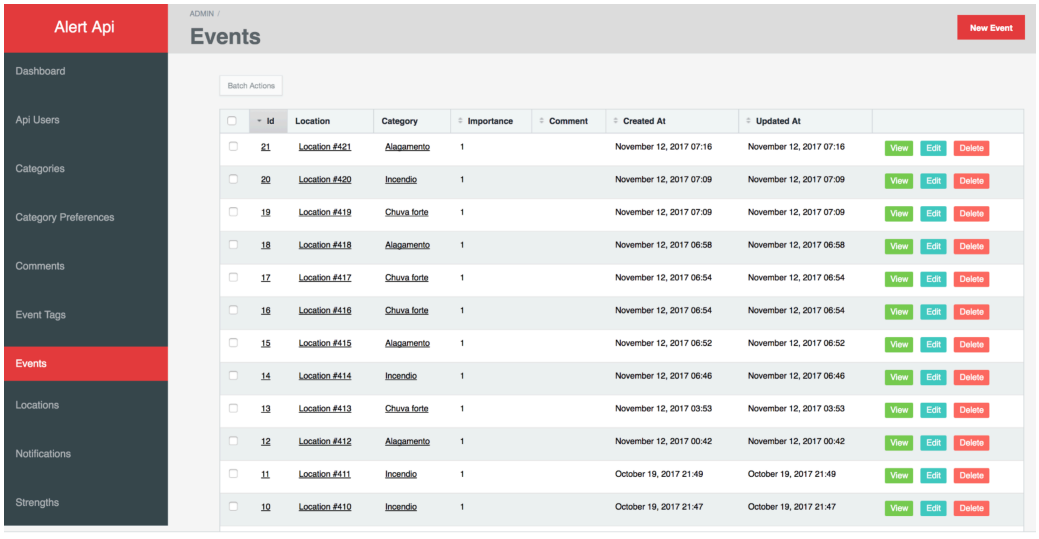


Figura 8. Página de monitoramento.

O aplicativo (Figura 9) implementado para exemplificar a utilização da API utiliza as funcionalidades de autenticação do usuário; visualização dos eventos em um mapa; visualização dos eventos em uma lista; visualização dos detalhes do evento e a localização em um mapa; possibilidade do usuário reforçar ou denunciar um incidente; possibilidade do usuário criar um evento a partir da categoria ou *tag* e; adicionar comentários.

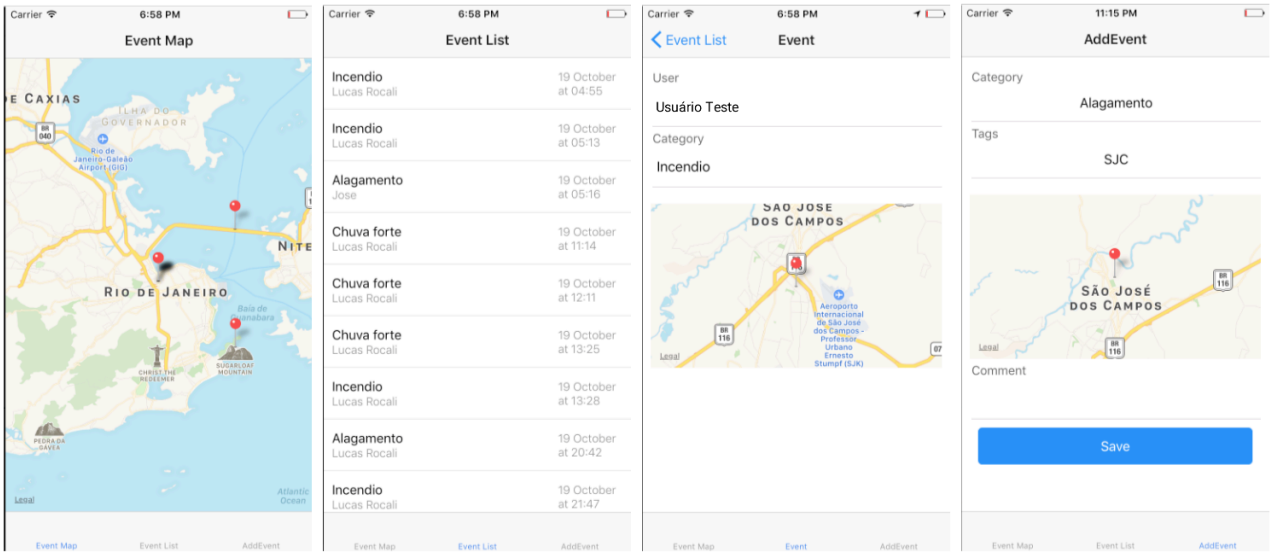


Figura 9. Página de monitoramento.

Foram executados testes de especificação para assegurar as relações entre as classes, bem como pela validação da presença de atributos. No exemplo apresentado pela Figura 9 é testado a classe do usuário do sistema (*User*), assegurando que ela pertence a classe de um sistema (*ApiUser*), verifica a validação do e-mail e senha, bem como localizações, preferências de categorias e *tags* do usuário. Também foram executados testes de requisição para assegurar o funcionamento da API, nestes testes foram verificados o retorno e status da requisição.

```
require 'rails_helper'

RSpec.describe User, type: :model do
  it { should belong_to(:api_user) }
  it { should have_one(:user_info).dependent(:destroy) }
  it { should validate_presence_of(:email) }
  it { should validate_presence_of(:password_digest) }
  it { should have_many(:category_preferences).dependent(:destroy) }
  it { should have_many(:tag_preferences).dependent(:destroy) }
  it { should have_many(:locations).dependent(:destroy) }
  it { should have_many(:strengths).dependent(:destroy) }
end
```

Figura 10. Exemplo de teste de especificação.

4. CONSIDERAÇÕES FINAIS.

Este trabalho apresentou a API genérica para gerenciamento e notificação de eventos, utilizando técnicas modernas de desenvolvimento e assegurando segurança, escalabilidade e versatilidade de uso por diversos sistemas. O trabalho apresenta um sistema robusto capaz de atender genericamente diversos tipos de aplicações/sistemas mantendo sua funcionalidade principal, que é o gerenciamento de eventos e notificação de usuários.

O trabalho também abordou a estratégia utilizada para o desenvolvimento da aplicação para desastres naturais que foi utilizada como exemplo de consumidor/usuário dessa API. A aplicação foi desenvolvida utilizando uma solução híbrida mas que não comprometeu a qualidade e usabilidade do aplicativo em ambas plataformas.

Todo o projeto da API, documentação das requisições, informações para uso da API e uma versão de demonstração do aplicativo se encontram no sítio eletrônico: (https://github.com/lucasrocali/Alert-App-2 / https://github.com/lucasrocali/Alert-API). Espera-se que seja possível disseminar o uso dessas tecnologias e técnicas de desenvolvimento usados nesse projeto.

5. REFERÊNCIAS

Bächle, M. and Kirchberg, P. "Ruby on rails." **IEEE software 24.6** (2007). Ruby on rails: https://rubyonrails.org/.

Cameron, M. A.; Power, R.; Robinson, B.; Yin, J. *Emergency situation awareness from twitter for crisis management*.In:Proceedings of the 21st International Conference on World Wide Web. New York, NY, USA:ACM, 2012. (WWW '12 Companion), p. 695–698. ISBN 978-1-4503-1230-1.

Eisenman, B. (2015). Learning React Native: Building Native Mobile Apps with JavaScript. "**O'Reilly Media, Inc.".** React Native: https://facebook.github.io/react-native/.

Mehrotra, A.; Pejovic, V.; Vermeulen, J.; Hendley, R.; Musolesi, M. *My phone and me: Understanding people's receptivity to mobilenotifications*. In:Proceedings of the 2016 CHI Conference on Human Factors in ComputingSystems. New York, NY, USA: ACM, 2016. (CHI '16), p. 1021–1032. ISBN 978-1-4503-3362-7.

Mehta, P.; Voisard, A.; Müller, S. *Clustering spatial data streams for targetedalerting in disaster response*. In:Proceedings of the 4th ACM SIGSPATIAL InternationalWorkshop on GeoStreaming. New York, NY, USA: ACM, 2013. (IWGS '13), p. 66–75. ISBN978-1-4503-2532-5.

OneSignal: High Volume Mobile and Web Push Notifications. Disponível em https://onesignal.com/. Acesso em: 10/05/2018.