Aplicação de uma solução REST para Filantropia com uso de geolocalização

Felipe de Oliveira Vianna Pinto[1], **Gustavo Bartz Guedes**[1], **André Constantino Silva**[1], **Daniela Marques**[1]

[1]Campus Hortolândia – Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP) 13183-250 – Hortolândia – SP – Brasil

fv.academico@gmail.com, gubartz@ifsp.edu.br

{andre.constantino,ifsp.daniela}@gmail.com

Abstract. *The interaction between people and services has changed with the large-scale introduction of mobile devices such as smartphones and tablets. This work presents a service-based architecture to support Ants, an application that connects volunteers and philanthropic institutions with the use of geolocation. Ants is particularly important considering that 25% of Brazilian population is supported by philanthropic institutions. The Web Server was developed using test-driven development that helped ensure compliance with the acceptance requirements.*

Resumo. *A interação entre pessoas e serviços mudou com a introdução em larga escala de dispositivos móveis, como smartphones e tablets. Este trabalho descreve o desenvolvimento de uma arquitetura baseada em serviço para suportar o Ants, um aplicativo que conecta voluntários e instituições filantrópicas com o uso de geolocalização. O Ants é particularmente importante considerando que 25% da população brasileira é atendida por instituições filantrópicas. O Web Server foi desenvolvido utilizando desenvolvimento orientado a testes que auxiliou a garantir o atendimento aos requisitos de aceitação.*

1. Introdução

A crescente popularidade e adoção de dispositivos móveis criaram inúmeras possibilidades de interação, como o uso de pontos de interesse baseados na localização do usuário [Capelas 2016]. Com o intuito de utilizar essa tecnologia em benefício da sociedade, o aplicativo *Ants* foi idealizado para auxiliar a interação entre voluntários e instituições filantrópicas, incentivando o aumento do número de pessoas que queiram contribuir com o terceiro setor. *Ants*, que significa "formigas" na língua inglesa, foi escolhido devido ao comportamento cooperativo das formigas em prol da colônia. O principal diferencial é o uso de geolocalização, apresentando aos voluntários oportunidades em potencial com base em sua localização.

O desenvolvimento do aplicativo *Ants* foi dividido em três partes, conforme mostra a Figura 1, realizadas por três analistas distintos, todos alunos do mesmo curso de Tecnologia em Análise e Desenvolvimento de Sistemas. Este trabalho apresenta exclusivamente o desenvolvimento do *Web Service* e o uso de testes unitários automatizados.

1. Desenvolvimento do *Web Service*: elaboração e integração de componentes de persistência e autenticação consistindo em uma arquitetura orientada a serviço.

2. Desenvolvimento do Aplicativo: elaboração de aplicativo para dispositivos móveis *Android* [Butler 2011].
3. Elaboração e execução de testes: criação de casos de teste com o intuito de reduzir defeitos e falhas no desenvolvimento do aplicativo.

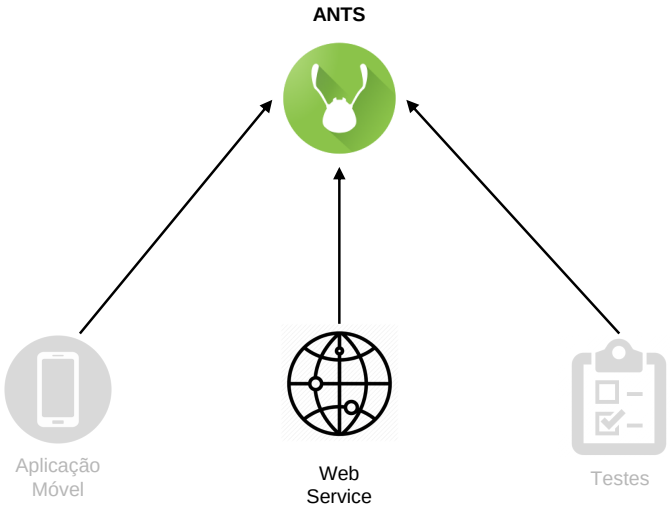

Figura 1. Divisão do Projeto *Ants*.

2. Motivação e Justificativa

O levantamento realizado pelo Instituto Superior de Estudos da Religião (ISER), em parceria com a Johns Hopkins University, mostrou que no Brasil existem em torno de 220 mil instituições beneficentes, sem fins lucrativos, somando 10 milhões de voluntários que auxiliam 40 milhões de pessoas, o que representa aproximadamente 25% da população brasileira [Rede Brasileira do Terceiro Setor 2017].

Com o intuito de analisar soluções tecnológicas de apoio à filantropia no Brasil, foi realizada uma pesquisa com o uso das palavras-chave "filantropia" e "organização não governamental" na ferramenta de busca da Google [Google 2017a] e na loja de aplicativos Play Store [Google 2017b] no período de janeiro de 2017 a março de 2017. Dezenove soluções foram escolhidas e analisadas de acordo com os atributos apresentados a seguir.

- Plataforma: indica a plataforma em que a aplicação é vinculada.
- Categoria:
  - Doação: objetivo da solução é a arrecadação de recursos financeiros.
  - Informativo: focada na disseminação de notícias sobre uma ou mais instituições.
  - Catálogo: apenas disponibiliza informações básicas de diversas instituições.
  - Voluntariado: divulga informações sobre oportunidades de voluntariado.
  - Adoção: divulga informações sobre adoção de animais.
- Multi-institucional: indica se aborda várias instituições.
- Colaborativa: indica se há interação entre a instituição e os voluntários.
- Geolocalização: indica se há utilização de geolocalização.

A Tabela 1 apresenta o resultado individual da análise realizada para cada solução.

Tabela 1. Resultado da análise de soluções tecnológicas para filantropia.

Aplicativo / Site	Categoria	Multi instituição	Geolocalização	Colaborativa	Plataforma
Joyz - Doação para caridade	Doação	Não	Não	Sim	Android
Coração de Davi	Informativo	Sim	Não	Não	
Revista Filantropia	Informativo	Sim	Não	Não	
DNF	Informativo	Sim	Não	Não	
Best of Fundraising	Informativo	Sim	Não	Não	
Casa de Espanha em Brasil	Informativo	Sim	Não	Não	
Socialmiix Busca de Interesses	Informativo	Não	Sim	Sim	
Well Done Good	Informativo	Não	Não	Sim	
Solidarity App	Doação	Não	Não	Não	
Philanthropy India	Informativo	Não	Não	Não	
JEDA	Doação	Sim	Não	Não	
Filantropia.org	Voluntáriado	Não	Não	Sim	WEB
Ongs Brasil	Catálogo	Não	Não	Não	
Sonhar Acordado	Voluntáriado	Sim	Não	Não	
Procure Um Amigo	Adoção	Sim	Não	Sim	
Instituto Agua Viva	Doação	Sim	Não	Não	
Instituto de Desenvolvimento Girassol	Informativo	Sim	Não	Não	
Juntos	Catálogo	Não	Não	Não	
AISEC	Informativo	Sim	Não	Não	

A Figura 2 apresenta um gráfico de setores em relação à categoria das soluções analisadas. Com 52,6% das instituições tendo como proposta a propagação de informações e notícias relacionadas ao terceiro setor ou às próprias instituições. Enquanto 10,5% é focada no incentivo ao voluntariado. Porém a intersecção entre as soluções que são focadas no incentivo ao voluntariado e que são multi institucionais representam menos de 6% das soluções analisadas.

Outro ponto importante é que apenas uma das soluções pesquisadas (Socialmiix) possui recurso de geolocalização. No período em que o levantamento foi feito constatou-se que o aplicativo não funcionava corretamente, em especial a função com geolocalização que deveria exibir apenas as instituições próximas a um determinado local, porém nenhuma era exibida. Primeiramente houve dificuldade para efetuar o cadastro no aplicativo, que é feito realizado com uma conta do Facebook [Facebook 2017], pois o aplicativo não respondia quando da solicitação do cadastro, entretanto ao reiniciá-lo o estado mudava subitamente para autenticado. Outro problema foi que ao solicitar a listagem das instituições próximas, o aplicativo não apresentava resposta e sua execução era encerrada.

O *Ants* mostra as oportunidades de voluntariado com base na localização, permitindo aos voluntários identificarem aquelas que estão nas proximidades. Finalmente, para evitar defeitos, como os notados no aplicativo Socialmiix, o *Ants* utilizou desenvolvimento orientado a testes.

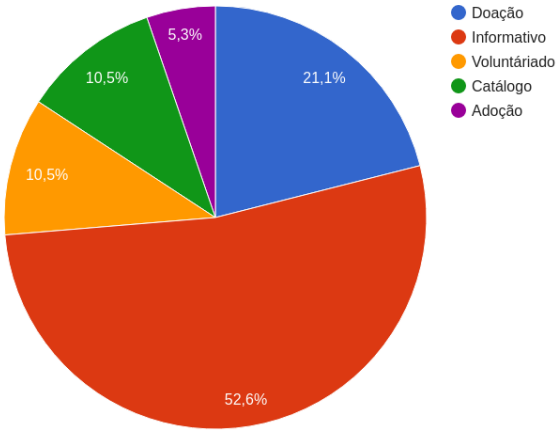

Figura 2. Gráfico de setores de soluções por categoria.

3. Referencial Teórico

Esta seção apresenta o referencial teórico estudado para elaboração e execução deste trabalho contendo conceitos importantes para compreensão da arquitetura e processos utilizados.

3.1. Representational State Transfer

O conceito de *Representational State Transfer* (REST) foi definido em 1993 por Roy Fielding como uma solução para a escalabilidade de rede para a *Internet* [Masse 2011].

REST é um conjunto de restrições na arquitetura que visa minimizar a latência e a sobrecarga de comunicação de rede. Ao mesmo tempo maximiza a independência dos componentes aumentando a escalabilidade de suas implementações em uma arquitetura cliente-servidor [Fielding 2000]. REST pode ser aplicada no desenvolvimento de arquiteturas baseadas em serviços com o uso de *Web Services*, por meio de uma interface uniforme de conexões entre cliente e servidor, isso permite que o mesmo serviço possa ser utilizado por diferentes plataformas.

3.2. OAUTH2

OAuth 2 é um *framework* de autenticação que utiliza *tokens*, que são códigos *hash* únicos para a validação da autenticidade de cada requisição. Permite que a autenticação de várias aplicações seja feita a partir de uma entidade autenticadora, além de autorizar uma aplicação a utilizar dados ou recursos da entidade autenticadora, como fotos e arquivos [Spasovski 2013].

3.3. Desenvolvimento Orientado a Testes

No Desenvolvimento Orientado a Testes (TDD), um caso de teste automatizado é inicialmente concebido e, na sequência, um teste é desenvolvido para validar o código a ser

implementado [Koskela 2007]. A implementação só é considerada exitosa quando o teste é executado com sucesso. A Figura 3 mostra o ciclo do TDD. A cada iteração o codigo é refatorado aprimorando sua qualidade, aumentando a confiança no código conforme mais funcionalidades e testes bem sucedidos se acumulam [Beck 2002].

Testes unitários automatizados podem ser utilizados para testes de regressão. Teste de regressão é o processo de testar programas de computador para garantir que defeitos não tenham sido introduzidos com alterações realizadas em iterações subsequentes [Rouse 2017].

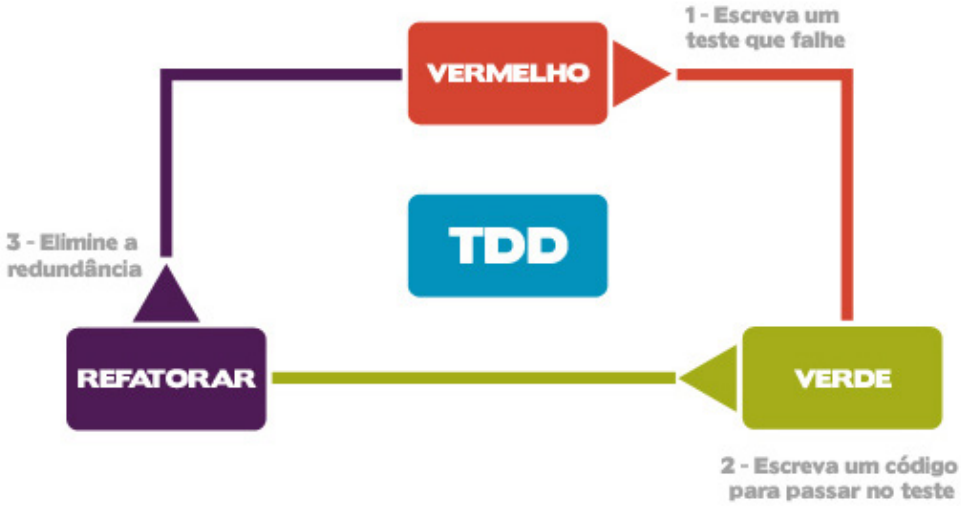

Figura 3. Ciclo do TDD.

3.4. SCRUM

O SCRUM é um *framework* de desenvolvimento pertencente à metodologia ágil em que o projeto é dividido em ciclos, denominados de *Sprints*, que geralmente compreendem um intervalo de tempo de uma semana a no máximo 15 dias [Schwaber and Sutherland 2012]. Nas *Sprints* são implementadas um subconjunto das funcionalidades totais do sistema, descritas em alto-nível em uma lista chamada *Backlog* do Produto. Essas funcionalidades são descritas em histórias que indicam como uma ação deve ser realizada [Schwaber and Sutherland 2012]. Durante a *Sprint* reuniões são realizadas diariamente para atualizar a equipe sobre a implementação dos requisitos a fim de garantir a entrega [Schwaber and Sutherland 2012].

No SCRUM a equipe do projeto é composta pelos papeis descritos a seguir.

1. *Product Owner*: responsável por definir os itens que compõem o *Backlog* do Produto e por estabelecer os itens a serem priorizados.
2. *Scrum Master*: responsável por gerenciar as práticas do SCRUM dentro da equipe.
3. Desenvolvedores: responsáveis pela implementação dos itens do *Backlog* do Produto.

4. Metodologia

Inicialmente foi realizada uma coleta e análise de requisitos para funções que auxiliassem a filantropia de algum modo. Esse levantamento foi feito junto a uma amostra composta por duas instituições, dos segmentos de auxílio à terceira idade e suporte a crianças com alguma deficiência (Cielo), e outra de cunho educacional (Associação Paideia). O levantamento foi utilizado para o desenvolvimento do aplicativo *Ants* que tem como intuito dar visibilidade e diminuir o desconhecimento das instituições nas proximidades.

Em seguida foram elaborados os requisitos do *Web Service* baseados nas necessidades do aplicativo como a utilização de geolocalização e os dados que devem ser persistidos. A Figura 4 mostra o fluxo de coleta de requisitos, desde a consulta às instituições, análise de requisitos, até a definição do que deveria ser implementado no *Web Server* para o funcionamento do aplicativo. Para o gerenciamento da implementação dos requisitos foi utilizado o *framework* de desenvolvimento SCRUM. Na organização das *Sprints* foram usados os princípios de Kanban [Hammarberg and Sunden 2014] com o uso da ferramenta Trello [Atlassian 2017] que permite a colaboração *on-line*.

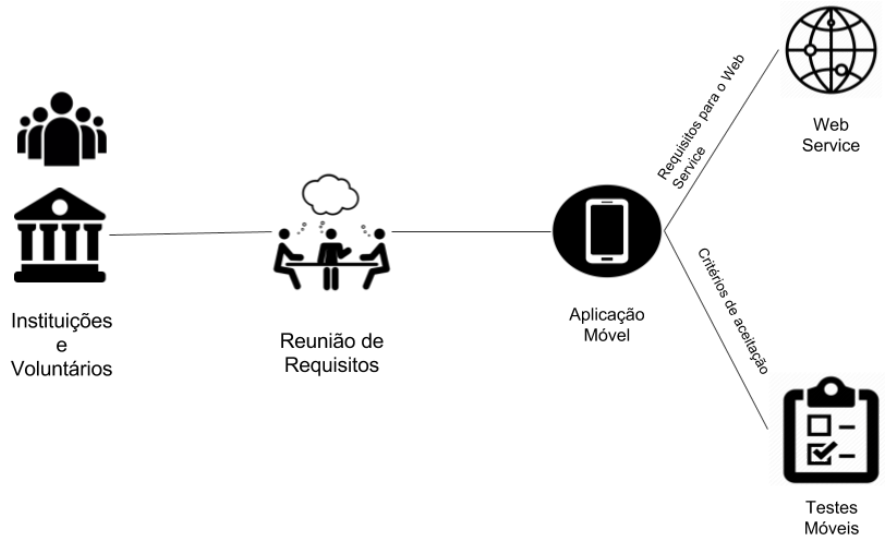

Figura 4. Fluxo de coleta de requisitos.

Foram definidas cinco *Sprints* a partir das histórias priorizadas, cada uma teve duração de duas semanas. Para cada história primeiramente foram elaborados os testes unitários baseados em seus critérios de aceitação. Quando algum dos testes unitários falhava, o código era refatorado até que todos os critérios de aceitação fossem atendidos. Esse processo se repetiu até a finalização do escopo do projeto. Com isso, o princípio do desenvolvimento orientado a testes foi seguido.

A elaboração do esquema do banco de dados foi feita de maneira incremental. As alterações no esquema relacional foram elaboradas conforme as necessidades com o objetivo de que os testes unitários obtivessem sucesso em sua execução.

5. Desenvolvimento do Ants e Arquitetura do Web Server

O principal diferencial do *Ants*, em relação às soluções apresentadas na seção 2 é o uso de geolocalização para aproximar entidades e voluntários, permitindo que o usuário en-

contre oportunidades de voluntariado com base em sua localização. A Figura 5 mostra um protótipo do *Ants*.

O *Web Service* elaborado neste trabalho dá suporte a uma aplicação da plataforma *Android*, porém como sua estrutura segue os princípios de REST qualquer aplicação capaz de efetuar requisições *Hypertext Transfer Protocol* (HTTP) pode consumir seus serviços. Assim, esta solução pode suportar uma gama maior de plataformas.

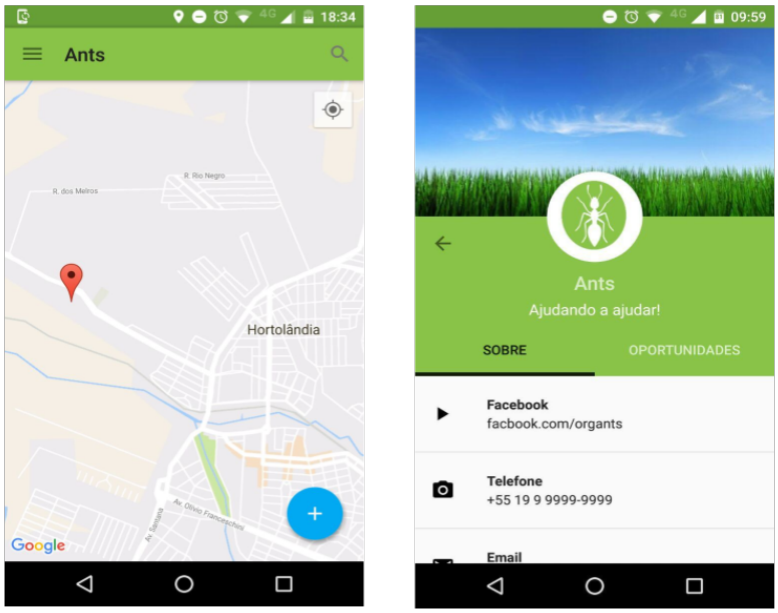

Figura 5. Protótipo da tela inicial do *Ants*.

5.1. Arquitetura

Este trabalho utiliza a arquitetura de *Web Service* projetado para suportar interoperabilidade entre aplicativos em uma rede [W3C 2004]. A arquitetura utilizada foi o REST, devido às informações serem transmitidas usando apenas o protocolo HTTP.

A arquitetura completa do *Web Service* é apresentada na Figura 6, composta pelos itens descritos a seguir.

1. Servidor: a fim de garantir alta disponibilidade, segurança e escalabilidade a hospedagem do sistema foi efetuada sobre uma plataforma de *Cloud* utilizando o sistema operacional Linux.
2. Sistema Gerenciador de Banco de Dados (SGBD): responsável por efetuar a persistência e recuperação dos dados. O PostgreSQL é uma solução de código aberto e possui a extensão PostGIS para dados espaciais, que provê recursos para o uso de geolocalização.
3. Plataformas Consumidoras: qualquer aplicativo, independentemente da plataforma, que efetue requisição HTTP para receber ou enviar dados ao *Web Service* em formato *JavaScript Object Notation* (JSON).
4. Servidor de Aplicação: contêiner no paradigma da linguagem Java, tem o papel de hospedar código compilado, prover e gerenciar todos os componentes dos quais o código faz uso como: acesso ao banco de dados, bibliotecas adicionais e gerenciamento de conexões.

5. Autenticação: para garantir a unicidade dos usuários, a integridade dos dados e o gerenciamento de acesso ao conteúdo, foram utilizadas entidades autenticadoras externas para identificação e autenticação dos usuários.

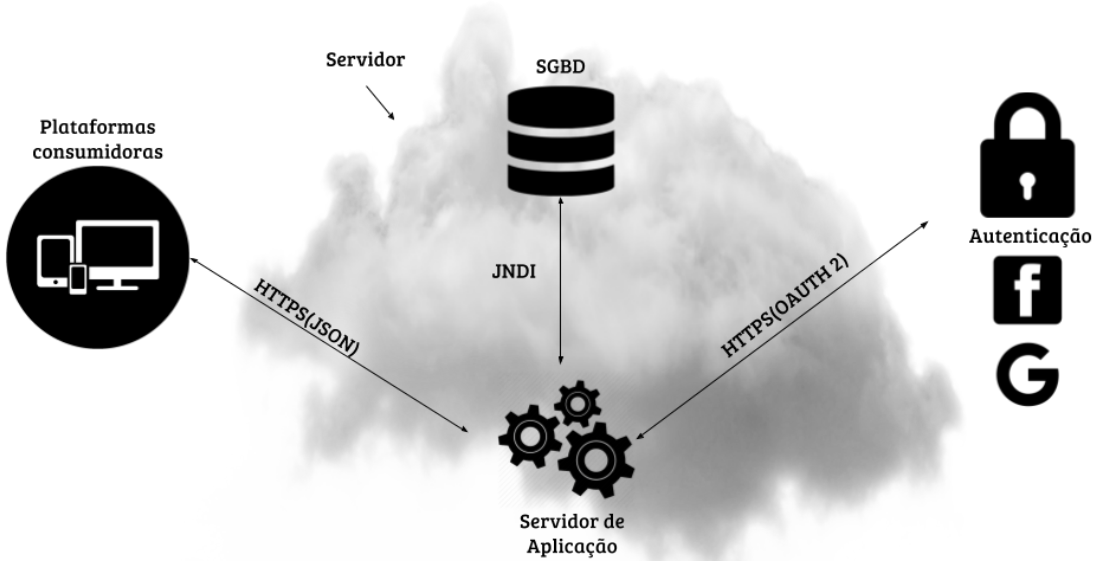

Figura 6. Arquitetura do sistema.

5.2. Banco de dados

O PostgreSQL é o mais avançado sistema de gerenciamento de banco de dados objeto-relacional de código aberto, pois possui uma comunidade ampla e ativa, suporte para ferramentas externas e pode ser expandido com a escrita de procedimentos [Tezer 2014].

Neste projeto em conjunto com o PostgreSQL foi utilizada a extensão PostGIS, que adiciona funcionalidades para geolocalização ao SGBD. A extensão PostGIS é a mais completa em termos de recursos para gerenciamento de dados espaciais [Boston Geographic Information Systems 2008]. Com a localização do usuário (latitude e longitude) é possível buscar todas as instituições e oportunidades dentro de um raio específico utilizando a linguagem de consulta estruturada SQL.

A Figura 7 apresenta o esquema conceitual do banco de dados elaborado para o *Web Service*. A multiplicidade um para muitos no conjunto de relacionamentos "Utiliza" se deve ao fato de que um usuário pode ter mais de um tipo de autenticação. Como seria o caso da autenticação na aplicação em si e outras como as permissões para uso dos recursos das entidades autenticadoras. Portanto, o conjunto de entidades "Autenticação" representa o tipo da autenticação e o *token* correspondente.

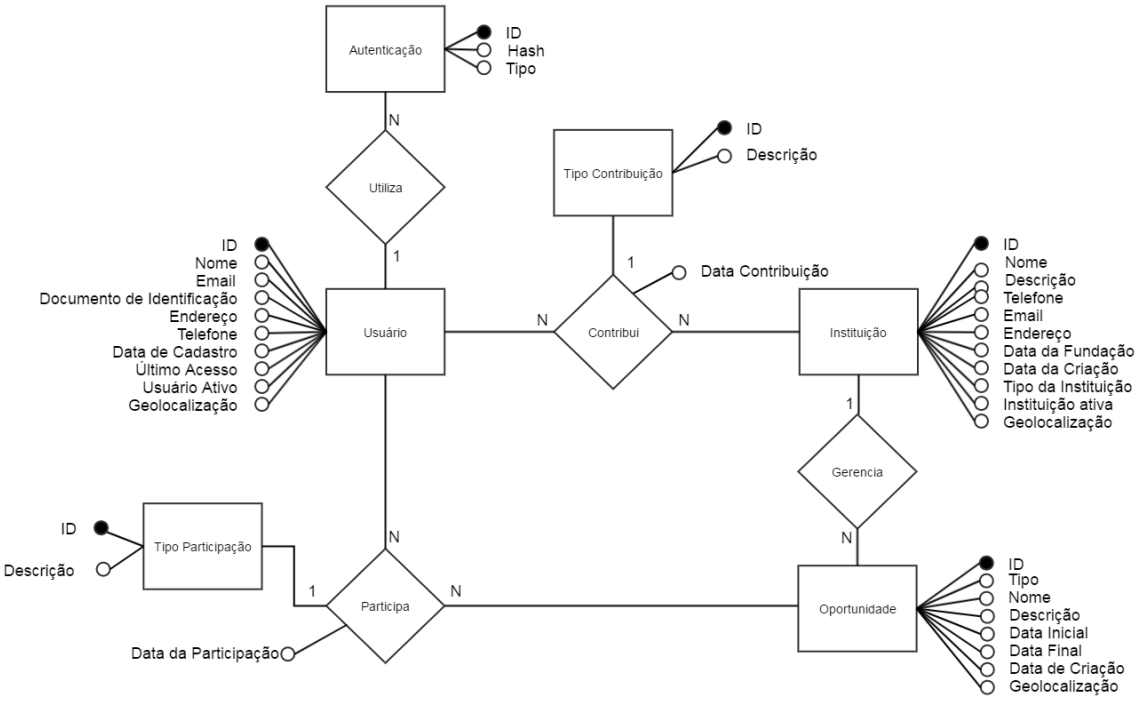

Figura 7. Diagrama Entidade Relacionamento.

O acesso ao banco de dados é realizado por meio de visões com o intuito de eliminar o acoplamento entre o esquema físico do banco de dados e o código do *Web Service*, além de simplificar consultas. Portanto, futuras alterações no esquema de dados não requerem alterações no código, sendo necessário apenas atualizar as respectivas visões.

5.3. Web Server e Servidor de Aplicação

O *Web Server* foi implementado na linguagem Java *Enterprise Edition* (Java EE) [Oracle 2017] com o *framework* RESTEasy [Jboss 2017a], utilizado na elaboração de *Web Services RESTFul*. O servidor de aplicação utilizado foi o WildFly [Jboss 2017b], que é um contêiner para aplicações Java que gerencia todas as requisições e recursos externos, conforme mostra a lista a seguir.

1. Intermedia comunicações entre o SGBD e a aplicação utilizando um *pool* de conexões utilizando o padrão *Java Naming and Directory Interface* (JNDI), mantendo um número determinado de conexões disponíveis para as solicitações da aplicação. Após finalizada, a conexão não é encerrada, mas retorna ao *pool* tornado-se disponível para uma próxima solicitação [Elias 2007].
2. Gerencia as conexões de rede aplicando restrições de acesso e sua quantidade, além de estabelecer conexões utilizando *Hyper Text Transfer Protocol Secure* (HTTPS), sobre uma camada adicional de segurança que utiliza o protocolo SSL/TLS.

As funcionalidades e papéis aos quais o *Web Server* atende estão representados no diagrama de caso de uso da Figura 8. Os serviços são acessados pelo aplicativo por meio de *Uniform Resource Identifier* (URI) de acordo com as premissas do REST, utilizando

métodos HTTP que determinam o tipo de ação será realizada sobre o recurso especificado. A Tabela 2 mostra uma relação dos URIs para ações relacionadas às instituições. A troca de informações feita entre cliente e servidor é realizada no formato JSON e enviada com o protocolo HTTP com uma camada de criptografia (HTTPS).

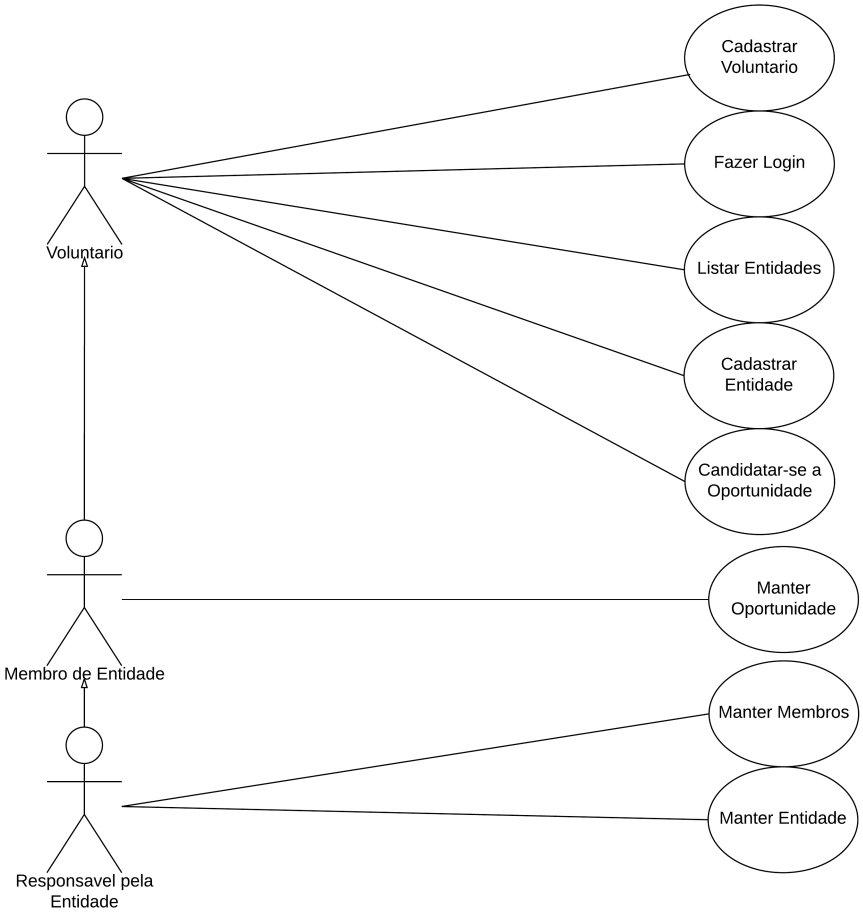

Figura 8. Diagrama de caso de uso.

Tabela 2. Exemplos de métodos HTTP e ações relacionadas.

Método	URI	Ação
POST	https://ants.com/api/institution/	Cadastrar Instituição
GET	https://ants.com/api/institution/	Listar Instituições Próximas
PUT	https://ants.com/api/institution/{id}	Atualizar Instituição
DELETE	https://ants.com/api/institution/{id}	Remover Instituição

5.4. Autenticação

A fim de evitar a elaboração e manutenção de um componente próprio de autenticação o sistema implementa o protocolo OAUTH2, padrão utilizado atualmente [Spasovski 2013]. Assim a aplicação faz uso de toda a infraestrutura e gerenciamento de senhas de uma entidade autenticadora.

Conforme mostra a Figura 9 o *Web Service* elaborado neste trabalho implementa dois fluxos de autenticação:

1. **Validação da identidade do usuário:** garante que o usuário existe e tem suas credenciais válidas perante uma entidade autenticadora.
   (a) O usuário é direcionado para o portal da entidade autenticadora pelo *Web Server*, como aponta o item 1 da Figura 9. Caso seja o primeiro acesso, informações adicionais para completar o perfil são solicitadas.
   (b) Uma vez que as credenciais são validadas, o *token* gerado pela entidade autenticadora é armazenado e contém as permissões necessárias para uso dos dados do usuário, conforme mostra o item 2 da Figura 9.
   (c) Para garantir a integridade do *token* gerado o *Web Server* efetua uma tentativa de renovar o *token* junto à entidade autenticadora. Essa renovação também acontece quando o *token* tem seu período de validade expirado.
2. **Autenticar todas as interações entre plataforma e *Web Server***: assegura que as mensagens trocadas são autênticas.
   (a) O processo é iniciado apenas após a validação da identidade do usuário.
   (b) O *Web Server* gera um *token* próprio que é utilizado pela plataforma durante todo o processo de iteração entre ambos.
   (c) O *token* tem um período de validade, que deve ser renovado para que a interação possa continuar.

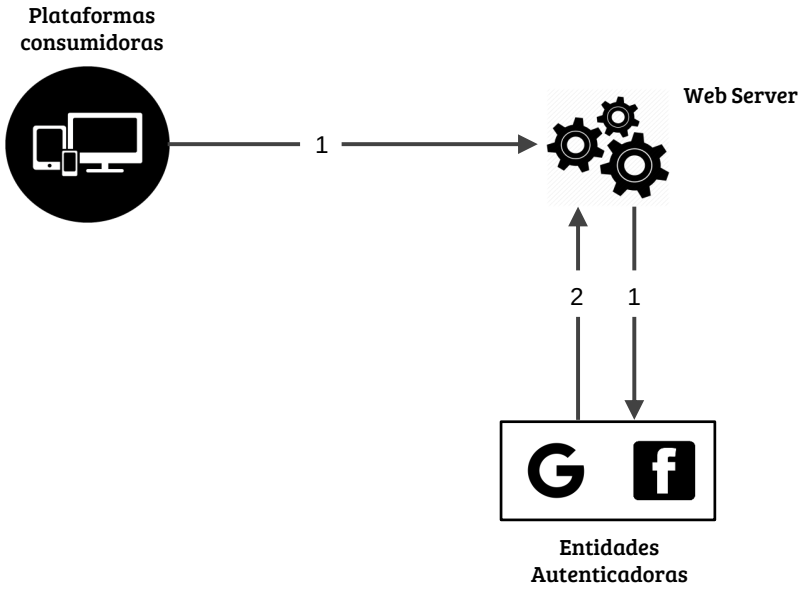

Figura 9. Fluxo de autenticação.

6. Aplicação do Desenvolvimento Orientado a Testes (TDD)

A fim de garantir a qualidade do sistema foi aplicado ao projeto a prática do TDD. Os casos de teste foram elaborados e implementados conforme os critérios de aceitação.

Os testes unitários foram elaborados no *framework* para desenvolvimento de testes unitários Junit [JUnit 2017] em conjunto com o REST Assured [Parkster 2017]. Esse último voltado para testes relacionados com *Web Services* REST.

A cada alteração relevante de código todo um teste de regressão, com o conjunto de testes já elaborados, era executado. O intuito foi garantir que defeitos não tivessem sido introduzidos após alterações. Após cada execução dos testes um resultado é gerado pelo JUnit. A Figura 10 mostra a saída com algumas marcações, que são detalhadas a seguir.

1. Lista dos casos de testes: cada caso de teste é composto por um conjunto de testes que contém os critérios de aceitação.
2. Um exemplo de teste em que houve um erro.
3. Tempo de execução do conjunto de casos de testes.
4. Quantidade de testes executados em relação à quantidade de testes existentes.
5. Quantidade de erros ocorridos. Erros são causados por sintaxe incorreta ou na compilação.
6. Quantidade de falhas ocorridas. Falhas são causadas quando o teste não é bem sucedido. Por exemplo, o valor esperado de retorno, por um teste, não é condizente com o resultado da execução do código.

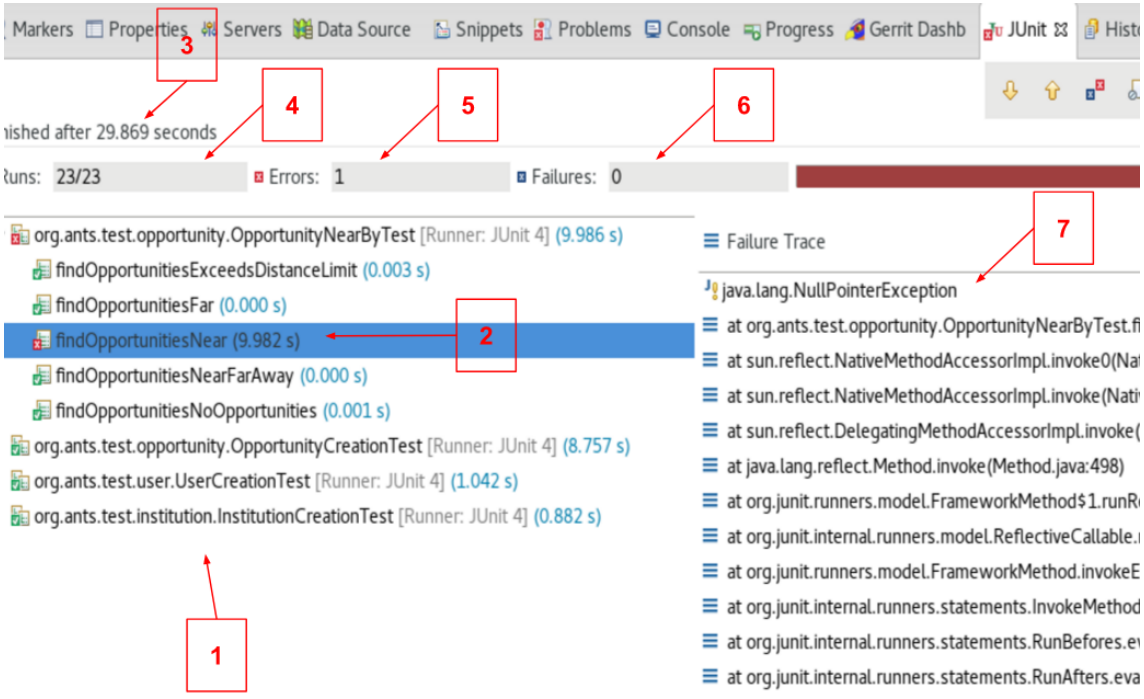

Figura 10. Exemplo de saída do Junit.

7. Conclusão

O *Web Service* foi finalizado e está disponível para uso do aplicativo. A utilização de métodos ágeis de desenvolvimento trouxe o ganho esperado com a geração de uma documentação concisa, porém houve alguns problemas quanto ao planejamento das *Sprints*, como a estimativa de tempo para implementação das histórias e assiduidade das entregas devido a pouca experiência da equipe com metodologia ágil de desenvolvimento impactando o prazo de entrega e a *backlog* do projeto.

Compor a arquitetura com tecnologias de código aberto facilitou a configuração e integração dos componentes devido a atividade das comunidades de código aberto, além

de aumentar a manutenibilidade do sistema como um todo pela garantia de atualizações sem necessidade de licenciamento. Porém, alguns erros não mapeados na documentação e falhas encontradas acrescentaram dificuldade na conclusão do projeto.

O suporte à geolocalização oferecido pelo PostGIS demonstrou-se muito útil pela facilidade que agregou às tarefas envolvendo a localização de entidades, oportunidades e voluntários. Sua configuração é custosa, mas uma vez terminada suas funções de inserção e busca de localização se mostraram simples e, por se tratarem de funções em PL/pgSQL, utilizadas em consultas SQL.

A exploração de tecnologias de geolocalização trouxe propostas para trabalhos futuros, como a utilização de dados demográficos para obter um mapeamento da relação entre as entidades e seus voluntários. Esses dados podem ser utilizados por entidades em campanhas de voluntariado ou até mesmo para ações do setor público.

Com a finalização do projeto espera-se que o aplicativo possa ser utilizado como um meio para entidades divulgarem seu trabalho e oportunidades de voluntariado, principalmente para comunidade próxima às instituições.

Referências

Beck (2002). *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Boston Geographic Information Systems (2008). Cross compare sql server 2008 spatial, postgresql/postgis 1.3-1.4, mysql 5-6. `http://www.bostongis.com/PrinterFriendly.aspx/content_name=sqlserver2008_postgis_mysql_compare`. [Online; acessado em 17 de janeiro de 2017].

Butler, M. (2011). Android: Changing the mobile landscape. *IEEE Pervasive Computing*, 10(1):4–7.

Capelas, B. (2016). Brasil chega a 168 milhões de smartphones em uso: Base instalada cresceu em 14 milhões de aparelhos. `http://link.estadao.com.br/noticias/gadget,brasil-chega-a-168-milhoes-de-smartphones-em-uso,10000047873`. [Online; acessado em 17 de fevereiro de 2017].

Elias, M. (2007). Connection pool: Como funciona, para que serve e como usar com o hibernate. `https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems`. [Online; acessado em 22 de janeiro de 2017].

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures.

Hammarberg, M. and Sunden, J. (2014). *Kanban in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition.

Koskela, L. (2007). *Test Driven: Practical Tdd and Acceptance Tdd for Java Developers*. Manning Publications Co., Greenwich, CT, USA.

Masse, M. (2011). *Rest API Design Rulebook*. O'Reilly Media, Inc, 1th edition.

Parkster (2017). Rest assured. http://rest-assured.io. [Online; acessado em 14 de março de 2017].

Atlassian (2017). Trello. https://trello.com. [Online; acessado em 22 de abril de 2017].

Facebook (2017). Facebook. https://www.facebook.com. [Online; acessado em 4 de maio de 2017].

Google (2017a). Google. https://www.google.com.br. [Online; acessado em 14 de março de 2017].

Google (2017b). Play store. https://play.google.com. [Online; acessado em 14 de março de 2017].

Jboss (2017a). Resteasy. http://resteasy.jboss.org. [Online; acessado em 14 de março de 2017].

Jboss (2017b). Wildfly. http://wildfly.org. [Online; acessado em 4 de maio de 2017].

JUnit (2017). Junit. http://junit.org/junit4. [Online; acessado em 22 de abril de 2017].

Oracle (2017). Java ee. http://www.oracle.com/technetwork/java/javaee/overview/index.html. [Online; acessado em 14 de março de 2017].

Rede Brasileira do Terceiro Setor (2017). O terceiro setor. http://www.terceirosetor.org.br/institucional/terceiro-setor. [Online; acessado em 14 de março de 2017].

Rouse, M. (2017). Regression testing. http://searchsoftwarequality.techtarget.com/definition/regression-testing. [Online; acessado em 22 de abril de 2017].

Schwaber, K. and Sutherland, J. (2012). *Software in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, and Leave Competitors in the Dust*. John Wiley & Sons, Inc., New York, NY, USA.

Spasovski, M. (2013). *OAuth 2.0 Identity and Access Management Patterns*. Packt Publishing.

Tezer, O. (2014). Sqlite vs mysql vs postgresql: A comparison of relational database management systems. https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Online; acessado em 17 de janeiro de 2017].

W3C (2004). Web services architecture. https://www.w3.org/TR/ws-arch. [Online; acessado em 3 de janeiro de 2017].