# Plataforma de colaboração para auxílio às administrações municipais

Aluno: João Paulo dos Santos Mota
Orientador: Prof. Dr. Roberto Hirata Jr.

9 de fevereiro de 2013

# Sumário

# Parte I

# Técnica

# 1 Introdução

Com o avanço no campo da computação móvel e crescente popularização de smartphones, tablets, PDAs e outros tipos de dispositivos, cresce o número de boas idéias para utilizar tais tecnologias. O intuito é explorar não somente a imensa quantidade de dados compartilhados diariamente por pessoas ao redor do mundo, mas também o crescente poder computacional presente em dispositivos de tamanho reduzido. Isso deve ser feito de forma eficiente, propondo soluções interessantes tanto do ponto de vista social quanto do ponto de vista computacional.

Vemos hoje grandes iniciativas feitas com o intuito de tirar o maior proveito possível do potencial de tais tecnologias, bem como das novas formas de comunicação em massa e compartilhamento de dados. Iniciativas tais como estratégias em diversos campos distintos como marketing, política, e muitos outros, visam tirar o maior proveito do poder das mídias sociais e softwares colaborativos.

Mais especificamente, do ponto de vista acadêmico, torna-se interessante utilizar esse potencial para construir soluções que ajudem a criar uma comunidade integrada, onde informações, de conhecimento específico ou de interesse geral, estariam disponíveis de modo simples e fácil. Grandes passos são dados nesse momento para que se possa alcançar tal objetivo, tais como a criação de redes sociais acadêmicas, como o STOA, integração das muitas bibliotecas da USP, com o SIBi, entre outros projetos.

Esses projetos são exemplos de como um bacharel em Ciências da Computação, juntamente com profissionais de outros ramos do conhecimento, pode unir todas as faculdades adquiridas durante o curso, de forma a construir tais soluções. Conhecimentos como construção e análise de algoritmos, técnicas de desenvolvimento de projetos de software, conceitos fundamentais a qualquer linguagem de programação, entre outros, são vitais para a criação de projetos de sucesso.

Esse trabalho visa contribuir com a comunidade viabilizando uma plataforma de colaboração, onde é possível compartilhar rapidamente informações sobre problemas existentes na cidade, facilitando a implementação de possíveis resoluções para os mesmos, bem como a criação de uma comunidade integrada. Tal plataforma consiste em uma aplicação web e uma aplicação móvel, esta última utilizando sistemas Android.

Nas próximas seções discutiremos alguns aspectos sobre sistemas colaborativos, tais como sua origem, evolução através dos anos e a classificação de sistemas dentro desse nicho. Também discutiremos detalhes da implementação da plataforma citada acima, seus desafios e algoritmos envolvidos.

## 1.1 Objetivos

É objetivo desse trabalho construir uma ferramenta de colaboração, onde é possível reportar problemas presentes dentro de uma instituição ou comunidade, de forma rápida e fácil, facilitando a comunicação entre as diversas partes que compõem uma comunidade, bem como obter um melhor entendimento acerca de softwares colaborativos, sua evolução através dos anos e como esse nicho de sistemas podem contribuir para melhorar a produtividade e a comunicação dentro de uma comunidade.

# 2 Aspectos de sistemas colaborativos

É fato que a colaboração é um aspecto intrínseco de comunidades de indivíduos onde os mesmos devem cooperar para solucionar eventuais problemas, ou ainda em comunidades onde os indivíduos tem responsabilidades definidas, em prol de alcançar um objetivo comum. Sabemos que esse aspecto esta presente não somente na espécie humana, como também em outras espécies presentes na natureza, nas quais a sobrevivência está diretamente relacionada com a eficiência com a qual a colaboração é realizada.

Em ciências da computação, torna-se interessante estudar aspectos de sistemas que facilitem a cooperação. Obviamente, para tal objetivo, é necessário definir o quê constitui colaboração em software e os parâmetros necessários para que a mesma seja realizada de forma eficiente.

Podemos considerar como sendo um software colaborativo, ou ainda groupware, qualquer tipo de software que auxilie na cooperação ou comunicação entre indivíduos, ou ainda na realização de um trabalho em grupo. Esse tipo de trabalho em conjunto é comumente citado na literatura como *trabalho cooperativo apoiado por computador*, ou ainda *CSCW*[1]. O auxílio fornecido por tal programa pode ser representado de várias formas: uma aplicação web que apresenta notícias aos usuários pode ser considerado um software colaborativo, assim como um servidor que entrega mensagens a pessoas em uma lista, como um fórum, ou ainda um software que auxilia um time de desenvolvimento de aplicativos a organizar, dividir e mensurar a qualidade do trabalho.

Apesar de todos os exemplos anteriores constituirem softwares colaborativos, nota-se claramente ampla distinção entre os mesmos, não somente em suas funcionalidades e objetivos, mas também no modo como as informações são compartilhadas, aspecto esse de central importância nesse caso. Por isso, tais sistemas recebem uma classificação baseada, entre outros fatores, na forma de interação e na distância geográfica entre os usuários.

Adiante é apresentada detalhes de tal classificação, bem como a origem de tais softwares e um pouco de sua história.

## 2.1 Origem e evolução

Embora o termo *CSCW* tenha sua origem datada na década de 80, é possível observar que as preocupações acerca de como utilizar tecnologia para desenvolver ferramentas em prol de melhorar a comunicação e ampliar os resultados do trabalho em grupo já existiam anteriormente. Em 1968, Douglas C. Engelbart publicou resultados de uma pesquisa[4] feita na Universidade de Stanford, que tinha com o objetivo desenvolver ferramentas para aumentar a produtividade dos pesquisadores em suas tarefas diárias. Esse processo permitiria entender

[1]Sigla para Computer-supported cooperative work

mais sobre os princípios do trabalho cooperativo e como este poderia ser usado para aumentar a capacidade intelectual de um ser humano. Entre as features apresentadas estavam implementação de um sistema de teleconferência, uma tela de trabalho compartilhada para edição, invenção do mouse, conceito de janelas, entre outros. Seu trabalho rendeu-lhe o nome de pai do software colaborativo.

O termo *groupware* surgiu anteriormente ao conceito *CSCW*, no ano de 1978. Seus inventores, Peter e Trudy Johnson-Lenz, fundadores da Awakening Technology, definiram groupware como software para apoiar processos em grupo[5]. Menos geral, portanto, que o subsequente termo *CSCW*, o qual engloba tanto a parte técnica como a parte humana.

Já na década de 80, nota-se a crescente popularização dos computadores pessoais. O acordo da Microsoft com a IBM garante o lançamento de computadores pessoais embarcados com o sistema operacional MS-DOS, o qual se tornaria extremamente bem sucedido nos anos seguintes. Temos também o lançamento do Apple Lisa, o primeiro computador pessoal com interface gráfica.

Com isso, o computador pessoal começa a fazer parte do dia-a-dia dos escritórios e, consequentemente, temos o lançamento de softwares para automatizar tarefas e ajudar o trabalho das pessoas. Porém, esses programas continham limitações e não poderiam satisfazer a demanda existente. Foi nesse contexto que, em 1984, em um esforço para ampliar os conhecimentos sobre trabalho cooperativo, Irene Greif e Paul M. Cashman apresentaram seminários a interessados em utilizar melhor a tecnologia disponível na época em seus trabalhos. Surgiu então o termo *Computer-sopported cooperative work*, que seria o estudo acerca de como atividades colaborativas e sua coordenação podem ser apoiadas em termos de sistemas computacionais[7].

## 2.2 Classificação

A classificação de elementos de um determinado campo de pesquisa visa agrupar os mesmos em categorias semelhantes com a finalidade de melhorar a organização e, por consequência, a eficiência do estudo. No caso de sistemas colaborativos não é diferente. Foram realizadas muitas tentativas de classificação de tais sistemas, porém trata-se de uma tarefa complexa devido a vários fatores como a dificuldade em definir formalmente colaboração, a constante evolução de tais sistemas e também sua própria natureza multi-tarefa.

Uma das primeiras tentativas de realizar tal classificação abordava aspectos temporais e físicos da interação entre os indivíduos envolvidos. A interação caracteriza-se como *síncrona* se as diferentes partes devem trocar informações de forma coordenada, sendo que uma parte é impossibilitada de prosseguir a tarefa antes da outra parte prover uma resposta. Um exemplo seria um serviço de chat, onde uma pessoa normalmente espera a resposta para prosseguir o diálogo. Se a sincronização não é fator relevante, a interação é *assíncrona*. Quanto ao aspecto

físico, é levado em conta a distância entre as diferentes partes em colaboração, podendo ser *local* ou *remota*. Johansen propôs em [6] a matriz de classificação contida na figura a seguir.

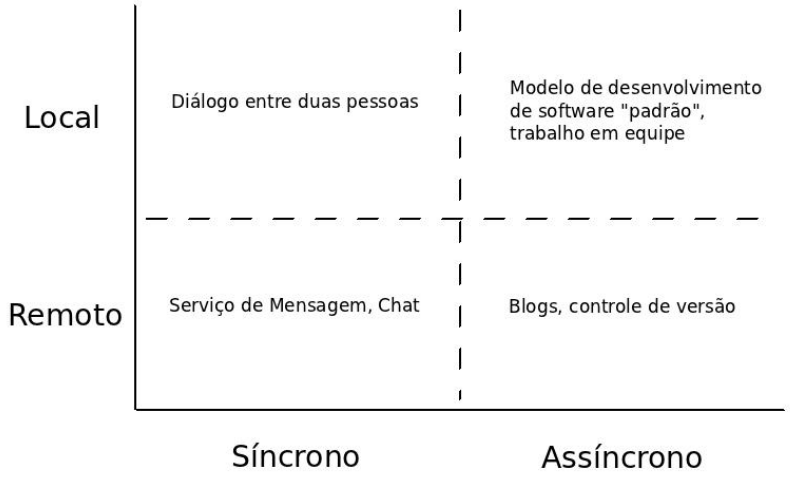


Figura 1: Matriz de classificação por tempo-espaço proposta por Johansen com alguns exemplos

Porém, não é necessário ser um especialista para observar que tal classificação é insuficiente pois aplicações modernas podem facilmente englobar todos os aspectos presentes na matriz, dado que um sistema um pouco mais complexo é capaz de englobar formas diversas de colaboração. Por exemplo, um sistema web de correio eletrônico provê serviço de email para o usuário, o qual na classificação de Johansen seria classificado como *assíncrono* e *remoto*. Porém, o mesmo sistema pode conter também um serviço de chat para usuários, o qual seria classificado como *síncrono* e *remoto*.

Devido a esse fator, é proposta de [1] apresentar uma nova forma de classificação que tem como objetivo minimizar os problemas acima. Essa nova classificação propõe a consideração de 3 novas características acerca da finalidade de tal sistema ou componente. São elas *coordenação*, *compartilhamento de informação* e *comunicação*. Por exemplo, um serviço de chat pode ser classificado em comunicação, já um serviço de "news feed" tem a finalidade de compartilhar informação e, por fim, um serviço de controle de versões de software pode ser considerado como um serviço de coordenação. Com isso, é possível classificar mais facilmente cada um dos componentes colaborativos que compõem um software moderno e assim, utilizar esse conjunto para classificar o software em si. Considere como exemplo a tabela a seguir:

| Características de Sistemas Colaborativos | | | |
|---|---|---|---|
| | Compartilhamento de Informação | Comunicação | Coordenação |
| A | 0 | 0 | 0 |
| B | 0 | 0 | 1 |
| C | 0 | 1 | 0 |
| D | 0 | 1 | 1 |
| E | 1 | 0 | 0 |
| F | 1 | 0 | 1 |
| G | 1 | 1 | 0 |
| H | 1 | 1 | 1 |

Na tabela acima vemos todas as possibilidades para classificação. Note, porém, que a primeira linha não representa um sistema colaborativo, dado que não apresenta nenhum aspecto relevante a colaboração.

Assim, se unirmos essa nova classificação com a matriz de Johansen teremos a tabela exemplo a seguir.

| Características de Sistemas Colaborativos | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Compart. de Informação | Comunicação | Coordenação | Síncrono | Assíncrono | Local | Remoto |
| Email | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| Chat | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| Telefone | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| Forum | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| Agenda | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| SVN | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| Blogs | 1 | 0 | 0 | 0 | 1 | 0 | 1 |

Note que as propriedades não são mutuamente exclusivas, ou seja, podemos, por exemplo, implementar uma agenda para uma sala de reuniões, onde as pessoas podem reservar a sala pessoalmente ou remotamente pela internet ou pelo telefone. Assim, a agenda, a qual é manipulada por muitos usuários que desejam reservar a sala, possue características de uma ferramenta colaborativa de acesso local ou remota. O mesmo pode ser dito a respeito de fórums. Os envolvidos podem se reunir em um local ou participar remotamente por tele ou vídeo conferência.

Alguns resultados interessantes foram obtidos de tal classificação. Vemos que, em geral, os sistemas colaborativos são classificados em padrões de valor ímpar, ou seja, apresentam valor 1 na casa menos significativa de sua representação binária. Isso pode ser explicado se

observarmos que a maioria das aplicações colaborativas atuais são ferramentas de uso remoto. Quanto as características relativas a tempo-espaço, o estudo conclue que mais de 85% das ferramentas analisadas podem ser classificadas como ferramentas de comunicação.

# 3 Plataforma de colaboração - Apresentação e Motivação

O sistema que será discutido nessa seção consiste em uma plataforma para facilitar a detecção de problemas na comunidade, agilizando assim a tomada de providências por parte de órgãos administrativos. Os problemas são reportados de forma rápida e fácil por qualquer indivíduo que possua acesso a um computador ou dispositivo móvel.

Seguindo modelos de plataformas conhecidas, um usuário detecta um problema, cadastra o problema no sistema, com sua localização, descrição e fotos do problema. Após esse processo, outros membros da comunidade podem acompanhar o desenrolar de uma possível solução para o problema, verificando se o mesmo ainda está em aberto, o tempo decorrido entre o cadastramento do problema e sua resolução, entre outros aspectos. Assim que um administrador da comunidade detecta um novo problema cadastrado, é iniciado o processo para validar e resolver tal problema. Esse processo ocorre fora do sistema, envolvendo medidas que dependem da natureza do problema em questão. Após anáĺise, o administrador poderá atualizar o status do problema, de forma a informar os usuários sobre as medidas tomadas. Então, o usuário pode finalmente finalizar o problema, fechando seu registro e marcando como resolvido. Abaixo, segue um caso de uso simplificado.

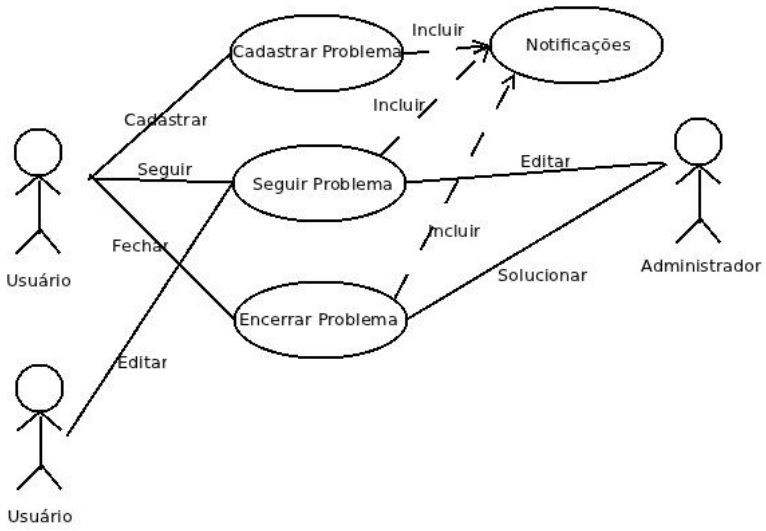


Figura 2: Caso de Uso do sistema. Atores são os usuários e os administradores

## 3.1 Motivação

Ferramentas colaborativas que ajudem a troca de informações e a organização de membros de uma certa comunidade em prol de um objetivo comum, ou simplesmente para ajudar o próximo, são extremamente úteis. Isso tornou-se evidente com a explosão das redes sociais. Modelos de sistemas como o Twitter, onde o objetivo é compartilhar informações de conteúdo sucinto, ou ainda Food.com, onde o foco é dividir experiências culinárias, são ex-

tremamente populares. Os objetivos são os mais diversos, entre eles estão entretenimento, artes ou política. Também não é incomum observar manifestações públicas organizadas pelas redes sociais, podendo ter objetivos relacionados somente com entretenimento[2], ou ainda manifestações de cunho social ou político[3][4].

Soma-se a tais fatores a dificuldade apresentada por órgãos tradicionais de administração em cobrir todas as necessidades de uma dada comunidade. Essa dificuldade está normalmente relacionada com uma alta demanda por serviços de administração e ouvidoria, procedimentos extremamente burocráticos de resolução de problemas e poucos profissionais com treinamento e incentivo para que o processo torne-se mais ágil. Segundo artigo publicado no blog da USP em 17/08/2011, a Ouvidoria da USP atendeu mais de 4 mil casos em 2010, sendo que para atender essa demanda possue apenas 2 funcionárias[5]. Considerando um pequeno orçamento, onde não é possível a contratação de muitos profissionais para atendimento de requisições feitas por uma comunidade numerosa, torna-se interessante utilizar sistemas colaborativos, como redes sociais, para monitorar o bem-estar comum e, dessa forma, conseguir realizar ações preventivas que acarretarão em diminuição do uso do serviço de administração ou ouvidoria. Esse modelo é adotado com sucesso em diversos sistemas, como por exemplo a Rádio Sulamérica Trânsito, que utiliza um sistema de monitoramento das vias em conjunto com a colaboração dos ouvintes para determinar e informar problemas nas vias da cidade de São Paulo.

Modelos propostos dessa forma dispensam a necessidade de moderadores para a comunidade, pois a própria comunidade é beneficiada pelo bom uso e manutenção do sistema. Outra vantagem seria a possibilidade de proporcionar a comunidade um meio para fiscalizar o bem-estar do ambiente comum e a eficiência dos órgãos administrativos em solucionar eventuais problemas.

[2]Flash Mob Pica pau

[3]Manifestação política em Brasília organizada através de redes sociais

[4]Flash Mob de incentivo a leitura

[5]Informação retirada de Blog da USP

# 4 Plataforma de Colaboração - Tecnologias utilizadas

A implementação do sistema foi dividida em duas partes, que consistem em uma aplicação web, e uma aplicação móvel. A aplicação móvel, foi implemendata utilizando a SDK[6]para Android. Já a aplicação web central foi implementada utilizando o framework para desenvolvimento web Ruby on Rails, o qual segue o modelo *MVC* de desenvolvimento.

## 4.1 Modelo MVC

O modelo MVC de desenvolvimento tem a origem de seu nome na lingua inglesa, sendo uma abreviação para *Model-View-Controller*. Trata-se de um modelo que tem como propósito organizar o desenvolvimento de uma aplicação em três camadas distintas. São elas: a modelagem de dados, a visualização desses dados e o controle das ações realizadas sobre tais dados.

A camada de modelagem reflete diretamente a criação de entidades de dados e suas relações, que normalmente tratam-se de uma abstração de relações entre objetos do mundo real. Por exemplo, na aplicação implementada nesse trabalho, temos uma entidade chamada "usuário", que representa uma abstração de uma pessoa usando o sistema, a qual se relaciona com outra entidade chamada de "problema", que por sua vez representa uma abstração de um possível problema na comunidade encontrado por uma pessoa.

A camada de controle defini as ações e regras com as quais torna-se possível manipular os dados definidos na camada anterior. Nessa camada é definido quais operações serão permitidas, quais operações serão proibidas, validações em operações de entrada e saída, entre outros. Pode-se dizer que trata-se do "cérebro"da aplicação.

Por fim, a camada de visualização é responsável por expôr os dados e suas relações ao usuário da aplicação. Também tem a função de prover uma interface para a entrada de dados e a posterior visualização do resultado do processamento de tal entrada pela camada controladora. Temos abaixo um esquema ilustrativo da interação entre as camadas.

[6]Sigla para Standard Development Kit

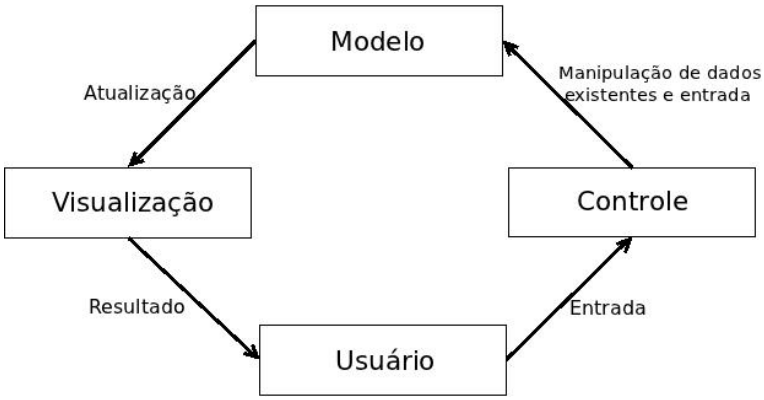


Figura 3: Fluxo de dados de acordo com modelo MVC

## 4.2 Ruby on Rails

*Ruby on Rails*[7] é um framework implementado para facilitar a construção de aplicações web. Utiliza como base a linguagem de programação *Ruby* e contém ferramentas para a geração e manutenção dos controles, modelos e visualizações que formam uma aplicação *MVC*. Também possui ferramentas para manipulação do banco de dados, como criação e alteração de tabelas e colunas. Para criar uma aplicação inicial, basta instalar *Rails* em um computador e utilizar o comando abaixo:

```
1 jpaulo@notebook:~$ rails new my_app
```

Esse comando irá criar um diretório chamado *my_app*, com uma estrutura definida de acordo com a figura abaixo.

[7] Ruby on Rails Site

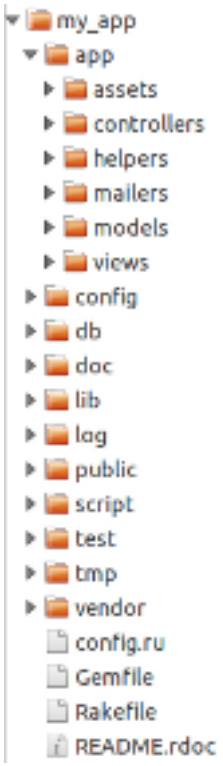


Figura 4: Estrutura de diretórios de uma aplicação gerada com *Rails*

Vemos que a estrutura gerada define alguns diretórios especiais para arquivos de configuração, como o diretório *config*, um diretório para armazenamento de arquivos de testes e um diretório chamado *app*, destinado a armazenar os arquivos que definem a implementação das funcionalidades do sistema. Nesse diretório localizam-se os arquivos que definem os modelos, as possíveis visualizações e os controles, formando assim a estrutura típica de um framework *MVC*. O diretório *db* contém arquivos que definem o esquema do banco de dados, bem como arquivos que, ao serem executados, realizam ações definidas para a modificação do banco.

## 4.3 PostgreSQL e integração da Base com Rails

A base de dados foi implementada utilizando o gerenciador de base de dados PostgreSQL. É licenciado através da PostgreSQL License, sendo essa uma licença de software livre. O framework *Ruby on Rails*, descrito anteriormente, tem total compatibilidade com bases de dados em *PostgreSQL*, simplificando a implementação e manutenção da mesma.

Para entender um pouco mais sobre a integração entre o banco de dados *PostgreSQL* e *Rails*, vejamos um exemplo: queremos realizar uma consulta de todos os usuários da aplicação que chamam-se Maria. Rails possui uma interface para realizar consultas ao banco de forma fácil, implementada por um de seus componentes, chamado *Active Record*. Esse componente é responsável por relacionar classes escritas em linguagem *ruby* e seus atributos com as tabelas presentes no banco de dados. Assim, o desenvolvedor escreverá a query da seguinte forma

```
1   @users = User.find(:name => 'Maria')
```

A classe *User* representa o modelo da entidade usuário, e herda os métodos da interface de consulta diretamente da classe *Base* do *Active Record*. Essa chamada do método *find* internamente é transformada em uma *query*, a qual é executada na base de dados. Uma vez completa, a consulta retorna uma lista de valores, os quais são transformados pelo *Active Record* em objetos da classe *User*.

Muitas são as vantagens proporcionadas pelo uso da interface de consultas descrita acima. Vejamos algumas delas:

- Clareza de código: é notável a clareza com a qual uma consulta é realizada, dado que a assinatura do método utilizado para a consulta deixa clara a sua função
- Concisão: o trabalho é realizado com clareza, utilizando pequenas linhas de código, evitando assim a presença de textos longos em linguagem *SQL*
- Portabilidade: caso surja a necessidade de realizar mudança no sistema gerenciador de banco de dados, não seria necessário mudanças no código da aplicação

## 4.4 Google Maps API

Ao cadastrar um problema no sistema, o usuário deve indicar o endereço do mesmo, o qual será marcado em um mapa. Dada a natureza geográfica da aplicação, fez-se necessário adotar uma solução de geolocalização eficiente. Para isso, a *API*[8]de geolocalização da empresa *Google* mostrou-se como a solução mais viável.[9]Através de chamadas assíncronas de procedimentos em linguagem *JavaScript*, é possível encontrar coordenadas geográficas para um endereço específico, ou ainda encontrar um endereço para um certo par de coordenadas. Esse processo é chamado de Geolocalização. Como trata-se de código *JavaScript*, o código é rodado pelo browser do cliente. Por isso, o processo de geolocalização não causa sobrecarga no servidor da aplicação.

Porém, ao utilizar a biblioteca de mapas do *Google*, é necessário adquirir uma licença de uso, com a qual é permitido uma certa quantidade de uso da biblioteca. Nesse trabalho, foi utilizada a licença de desenvolvedor, com a qual é possível realizar até 2500 requisições de geolocalização por dia, desde que a aplicação seja livremente e abertamente acessível a qualquer usuário.

[8]API: Application Programming Interface
[9]Mais sobre a API de geolicalização do Google

# 5 Plataforma de Colaboração - Implementação

As tecnologias utilizadas acima possuem diversas possibilidades de utilização. Um desenvolvedor interessado em implementar um sistema onde existe grande quantidade de operações de entrada e saída de uma base de dados, por exemplo, deve estar mais interessado em otimizar a busca em base quando comparado a um desenvolvedor interessado em implementar um jogo, onde o foco está em cálculos para a renderização de objetos 3D.

Com esse exemplo, vemos que diferentes aplicações requerem diferentes usos da tecnologia disponível. Cabe ao desenvolvedor analisar o problema em mãos e saber distinguir pontos críticos para a performance do sistema proposto.

Tendo em vista tal fato, iremos analisar os pontos principais do sistema *OuvidUSP*, discorrendo sobre as decisões de implementação de partes importantes da aplicação

## 5.1 Implementação da Base de Dados

A implementação da base de dados segue o modelo relacional, onde cada entidade representa uma tabela na base de dados, com relacionamentos representados por chaves estrangeiras. Abaixo, vemos o diagrama da base de dados do sistema implementado.

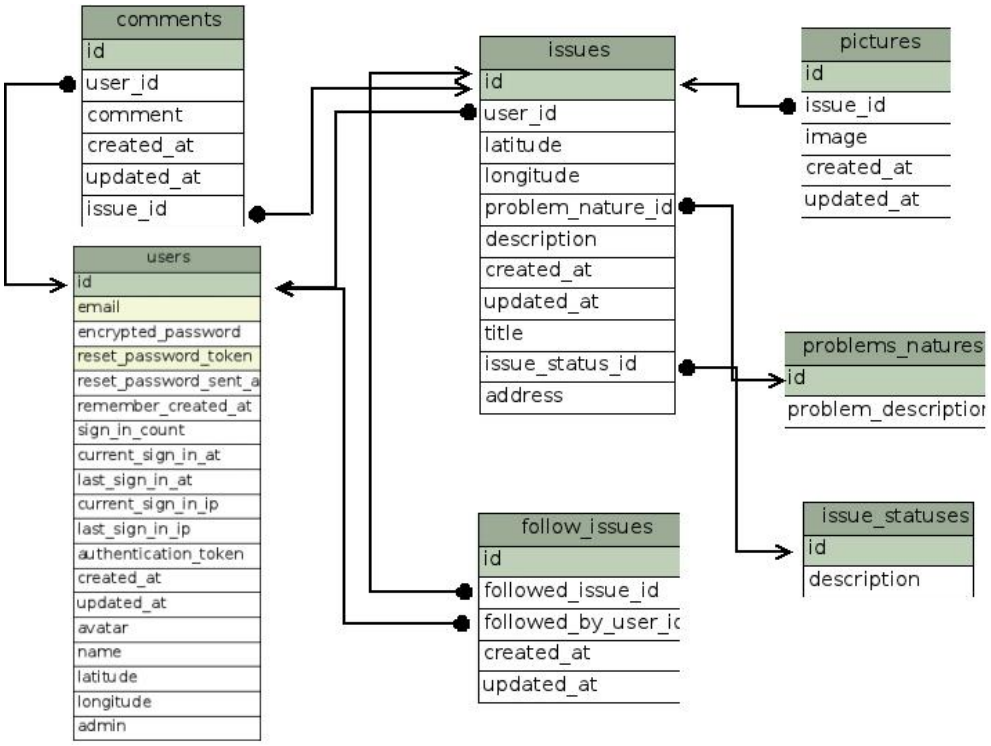


Figura 5: Diagrama representando a base de dados do sistema OuvidUSP

Quando um usuário cadastra um problema, é criada uma entrada na tabela *issue*, contendo os atributos pertencentes a tal entidade. O usuário também pode anexar fotos ao

problema ou decidir acompanhar o andamento de um problema, que gera entradas nas tabelas *pictures* e *follow_issues*, respectivamente. Por fim, o usuário também pode fazer comentários em um problema, gerando entradas na tabela *comments*. Todas as relações são implementadas por referências feitas através das chaves estrangeiras presentes nas tabelas do banco.

Ao desenvolver uma estrutura inicial para a base de dados, o desenvolvedor deve levar em conta aspectos como eficiência e escalabilidade, mas também é necessário estabelecer uma estrutura coerente com as escolhas feitas a cerca das tecnologias utilizadas no projeto. Como um exemplo, vemos o campo *address* da tabela *issue*. Trata-se de um campo texto responsável por guardar o endereço do problema. Então, uma busca por um problema em um endereço específico terá que analisar campos de texto de vários problemas contidos na base de dados para eventualmente devolver um resultado, o qual, no pior caso, será vazio. É fato que buscas em textos representam operações custosas, por isso poderiamos ter dividido o campo *address* em vários outros campos menores, como *street*, *number*, *neighborhood*, entre outros. Porém, ao fazer tal divisão, não estão sendo levados em conta endereços sem um bairro definido, ou locais onde não existem nomes de rua. Além disso, também deve-se levar em conta a presença dos campos *latitude* e *longitude*, que auxiliam na localização do problema, através da utilização da *API* de geolocalização da empresa Google. *API* essa que recebe strings de texto representando endereços ao redor do globo e devolve sua latitude e longitude, ou recebe um par de coordenadas e devolve seu endereço. Por isso, a escolha feita não é a mais eficiente do ponto de vista algorítmico, mas é plausível no contexto da aplicação.

## 5.2 Geolocalização

Na aplicação implementada, existem duas ações possíveis envolvendo a geolocalização de locais cadastrados. São elas:

- Ao entrar com um endereço para obter sua localização no mapa
- Ao ordenar problemas por critérios de distância

Para a primeira ação, a implementação segue uma abordagem direta. É recuperada a *string* que representa a entrada de um endereço pelo usuário, e essa *string* é fornecida a *API* do *MAPS*, para então encontrar as coordenadas corretas. Vemos a implementação abaixo.

```
1     var map = new google.maps.Map(document.getElementById('map_canvas'),
2             myOptions);
3     var geocoder = new google.maps.Geocoder();
4     var marker = new google.maps.Marker({
5      position: pos,
6      draggable: true,
```

```
7       title:""
8           });
9
10
11      function showAddress() {
12        var address = document.getElementById('issue_address').value
13        geocoder.geocode( { 'address': address}, function(results, status) {
14          if (status == google.maps.GeocoderStatus.OK) {
15            map.setCenter(results[0].geometry.location);
16            document.getElementById('issue_latitude').value = results[0].
                  geometry.location.lat();
17            document.getElementById('issue_longitude').value = results[0].
                  geometry.location.lng();
18            marker.setPosition(results[0].geometry.location);
19          } else {
20             alert();
21          }
22        });
23      }
```

Vemos que o objeto *geocoder* possui um método chamado *geocode*, que recebe como parâmetro um endereço e uma função de *callback*. Nessa função, temos uma condição que verifica se a *API* retornou com sucesso ou se houve erro ao tentar localizar um endereço. Caso a variável *status* sinalize sucesso, as coordenadas do endereço são resgatadas através do *array results*, um marcador, representado pela variável *marker*, é colocado na posição correta e o mapa é atualizado. Abaixo temos um problema cadastrado com seu endereço e mapa.

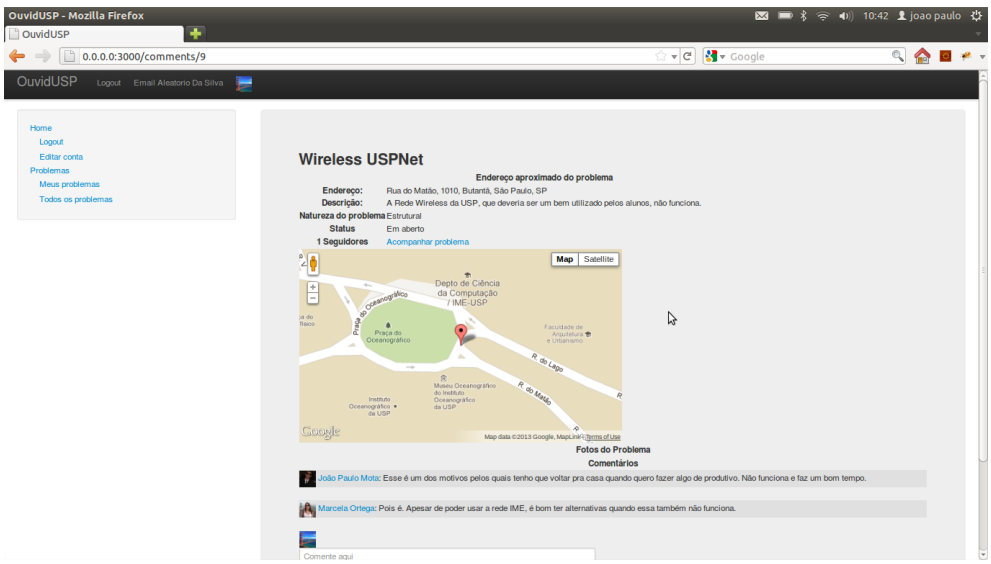


Figura 6: Exemplo de geolocalização onde o problema tem seu local marcado

A abordagem descrita acima resolve o primeiro problema encontrado de geolocalização, o qual refere-se a encontrar locais referentes a um endereço fornecido. É apresentado na seção seguinte a solução encontrada para o segundo problema de geolocalização encontrado, referente a ordenação de problemas pelo critério de distância até um ponto comum.

## 5.3 Cálculo de Distância entre Pontos: Fórmula de Haversine

Na tela de listagem de problemas cadastrados, temos filtros que auxiliam o usuário, seja ele um usuário comum ou administrador de comunidade, a organizar os problemas de acordo com sua necessidade. Entre os filtros implementados estão filtros por natureza do problema e por status dos problemas. Existe também a possibilidade de ordenação dos problemas por distância a um ponto comum. A esse ponto dá-se o nome de *local favorito*, o qual o usuário pode cadastrar ao editar seu perfil. Se o usuário não possuir um *local favorito* cadastrado, a ordenação por distância não tem efeito e a lista de problemas permanece inalterada. Abaixo vemos exemplos das funcionalidades de cadastro de local favorito e o filtro por distância.

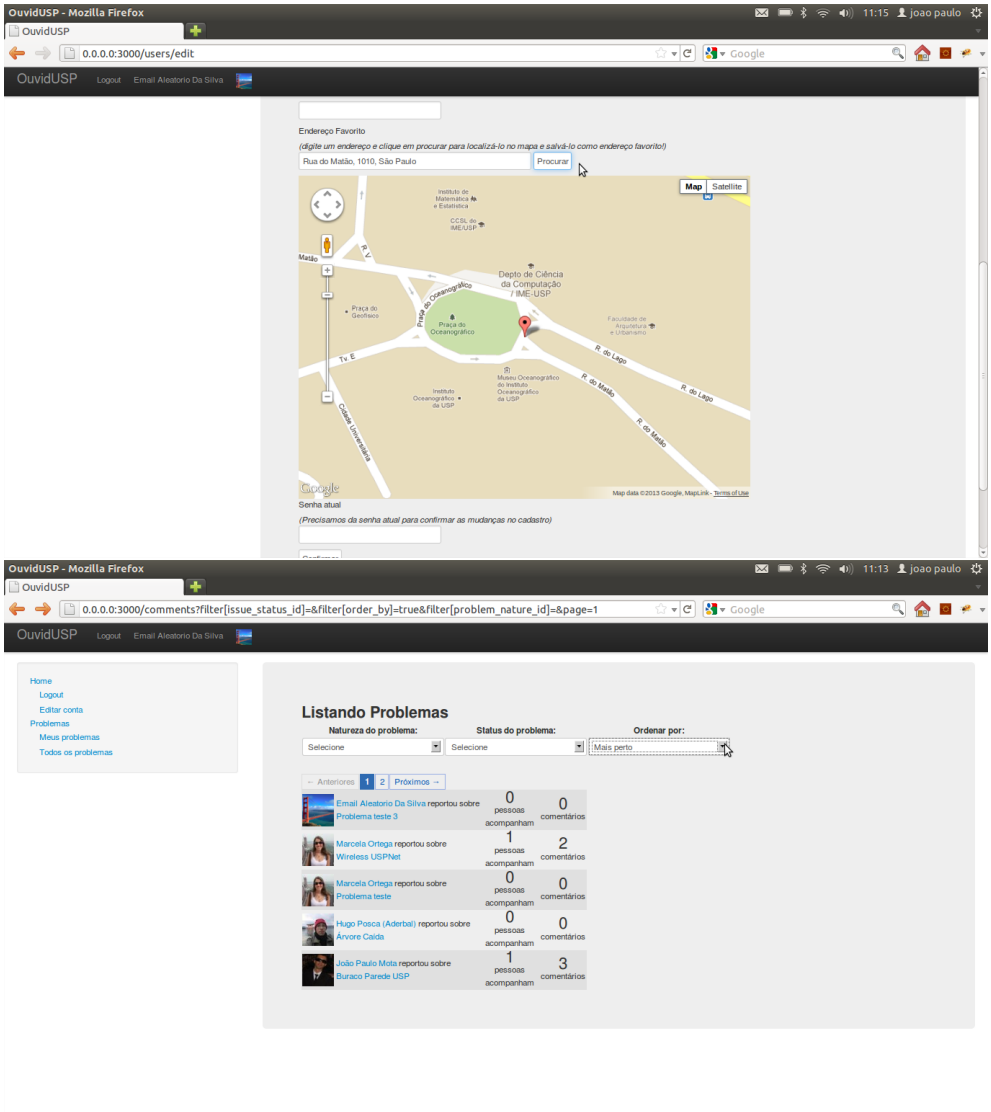


Figura 7: Cadastro de local favorito e ordenação de problemas por critério de distância

Para implementar tal funcionalidade, é necessário realizar uma consulta na base de dados e recuperar a lista de problemas. Cada problema possui dois atributos numéricos em ponto flutuante, representando a *latitude* e a *longitude*, os quais são preenchidos ao procurar por um endereço no campo texto ou mover o marcador no mapa, ambos na tela de cadastro de problemas. Analogamente, quando um usuário cadastra seu *local favorito*, são preenchidos dois atributos pertencentes a entidade *usuário*, também representando *latitude* e a *longitude*. Abaixo é possível observar a implementação da ordenação dos problemas.

```
1   @issues.sort_by!{|issue| haversine_distance(issue.latitude, issue.
        longitude, current_user.latitude, current_user.longitude) }
```

A ordenação é feita pela função *sort_by*!. Ela é aplicada ao objeto *@issues*, o qual representa a lista de problemas a ser mostrada na tela. A função *sort_by*! recebe como parâmetro um atributo, que será utilizado como valor comparativo no algoritmo de ordenação. Esse atri-

buto pode ser qualquer expressão, a qual será avaliada e transformada em uma chave para comparação. No caso em questão, essa expressão é uma função chamada *haversine_distance*, que recebe os valores de latitude e longitude de dois pontos e retorna um valor numérico, representando a distância entre os dois pontos. Esse valor é calculado para cada entrada da lista *@issues*, e associado a seu objeto, criando assim uma lista de tuplas. Após isso, essa lista de tuplas objeto-chave é ordenada de acordo com o valor da chave associada ao objeto. Esse processo faz com que a ordenação através da função *sort_by*! seja custosa para casos onde o atributo utilizado na ordenação é simples, pois deve-se iterar a lista duas vezes. Porém quando o critério de ordenação utilizado é complexo, esse método é mais eficiente.

A função *haversine_distance* calcula a distância entre dois pontos de uma esfera, passando por sua superfície, utilizando a *fórmula de haverseno*. Essa distância é na verdade o comprimento do segmento de *círculo máximo* formado pelos dois pontos. Em termos gerais, um *círculo máximo* pode ser definido como a intersecção entre a esfera e qualquer plano que contenha o centro da esfera.

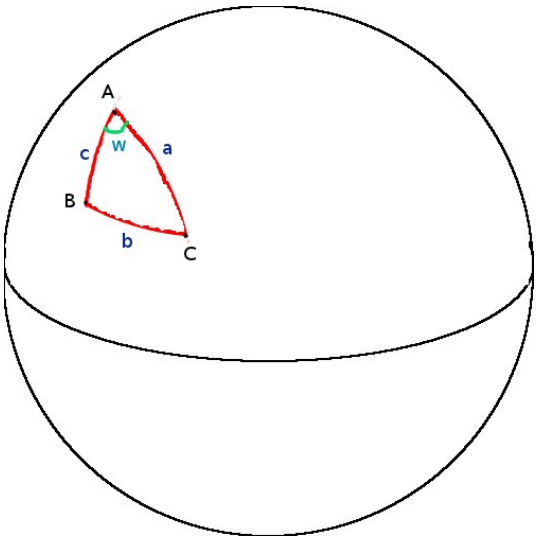


Figura 8: Lei de Haversenos na esfera

*Verseno* é uma função trigonométrica dada pela seguinte equação: $verseno(\theta) = 2 * \sin^2(\frac{1}{2} * \theta)$. Já a função *Haverseno* é definida como a metade do *verseno*: $haverseno(\theta) = \sin^2(\frac{1}{2} * \theta)$

Considerando a figura acima, podemos aplicar a *lei dos cossenos esféricos* no triângulo formado pela união dos pontos $A$, $B$ e $C$ em uma superfície esférica de raio unitário. Tem-se a seguinte fórmula:

$$\cos(b) = \cos(c) * \cos(a) + \sin(a) * \sin(c) * \cos(w)$$

. Sabemos também que são válidas as seguintes equações:

$$\cos(\theta) = 1 - 2 * haversine(\theta)$$

$$\cos(\theta - \omega) = \cos(\theta) * \cos(\omega) + \sin(\theta) * \sin(\omega)$$

Substituindo as equações acima na expressão da *lei dos cossenos esféricos*, é possível chegar na *lei dos haversenos*, dada pela seguinte expressão:

$$haverseno(b) = haverseno(c - a) + \sin(a) * \sin(c) * haverseno(w)$$

Sejam $P$ e $Q$ dois pontos na superfície de uma esfera, representados por pares ordenados $(\phi_1, \lambda_1)$ e $(\phi_2, \lambda_2)$, onde $\phi_i$ representa a latitude e $\lambda_i$ representa a longitude do ponto. Seja $d$ a distância entre $P$ e $Q$, e seja $R$ o raio da esfera, a *lei dos haversenos* pode ser reescrita da seguinte forma:

$$haverseno(d/R) = haverseno(\phi_2 - \phi_1) + \cos(\phi_1) * \cos(\phi_2) * haverseno(\lambda_2 - \lambda_1)$$

. Substituindo a função *haverseno* pela sua expressão e aplicando funções trigonométricas inversas como $\arcsin(\theta)$, é possível encontrar uma função para o cálculo da distância $d$. Temos a seguinte expressão:

$$d = 2 * R * \arcsin(\sqrt{\sin^2(\frac{\phi_2 - \phi_1}{2}) + \cos(\phi_1) * \cos(\phi_2) * \sin^2(\frac{\lambda_2 - \lambda_1}{2})})$$

Abaixo, a implementação do cálculo de $d$ em linguagem *Ruby*.

```ruby
EARTH_RADIUS = 3963.19
RADIAN_PER_DEGREE = Math::PI / 180.0

def haversine_distance(lat1, lng1, lat2, lng2)
  return 0 if lat1.nil? or lat2.nil? or lng1.nil? or lng2.nil?

  distlat_radians = (lat2-lat1) * RADIAN_PER_DEGREE
  distlng_radians = (lng2-lng1) * RADIAN_PER_DEGREE

  lat1_rad = lat1 * RADIAN_PER_DEGREE
  lon1_rad = lng1 * RADIAN_PER_DEGREE

  lat2_rad = lat2 * RADIAN_PER_DEGREE
  lon2_rad = lng2 * RADIAN_PER_DEGREE


  a = Math.sin(distlat_radians/2)**2 + Math.cos(lat1_rad) * Math.cos(
```

```
        lat2_rad) * Math.sin(distlng_radians/2) ** 2

    c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a))

    EARTH_RADIUS * c
  end
```

## 5.4 Plataforma Móvel

A plataforma móvel foi implementada utilizando o kit de desenvolvimento para sistemas *Android*. Trata-se de um sistema operacional de código aberto para dispositivos móveis, tais como celulares e tablets, desenvolvido pela empresa *Google*, com base em *kernel Linux*. Desenvolvedores podem implementar aplicativos que estendem a funcionalidade dos dispositivos e disponibilizá-los gratuitamente, ou mediante pagamento, para todos os usuários de dispositivos com *Android*. E para desenvolver tais aplicativos, basta adquirir o pacote *Android SDK*, configurar o ambiente de desenvolvimento e implementar as funcionalidades desejadas.

Na implementação da plataforma em questão, a aplicação realiza chamadas a um *serviço web*, que aceita requisições e devolve respostas em formato *JSON*. *JSON*, abreviação para *JavaScript Object Notation*, representa uma formatação para troca de dados, sendo uma alternativa ao famoso formato *XML*.

Ao requisitar alguma ação desejada na aplicação do dispositivo móvel, o usuário, o qual já possui uma conta no sistema, irá fornecer algum dado necessário para tal ação, como por exemplo, seu login e senha para autenticação. Esses valores são recuperados da interface e enviados para uma camada responsável por codificar os dados em *JSON*. Essa camada irá, por sua vez, enviar as informações para a camada responsável por realizar as chamadas de *webservice*. Essa camada, por fim, irá realizar uma chamada *HTTP*, abrindo uma conecção com o servidor. Ao receber a resposta, o processo inverso é realizado. Abaixo encontra-se um diagrama explicativo contendo a interação entre o aplicativo móvel, o browser em cliente fixo e o servidor *Rails*.

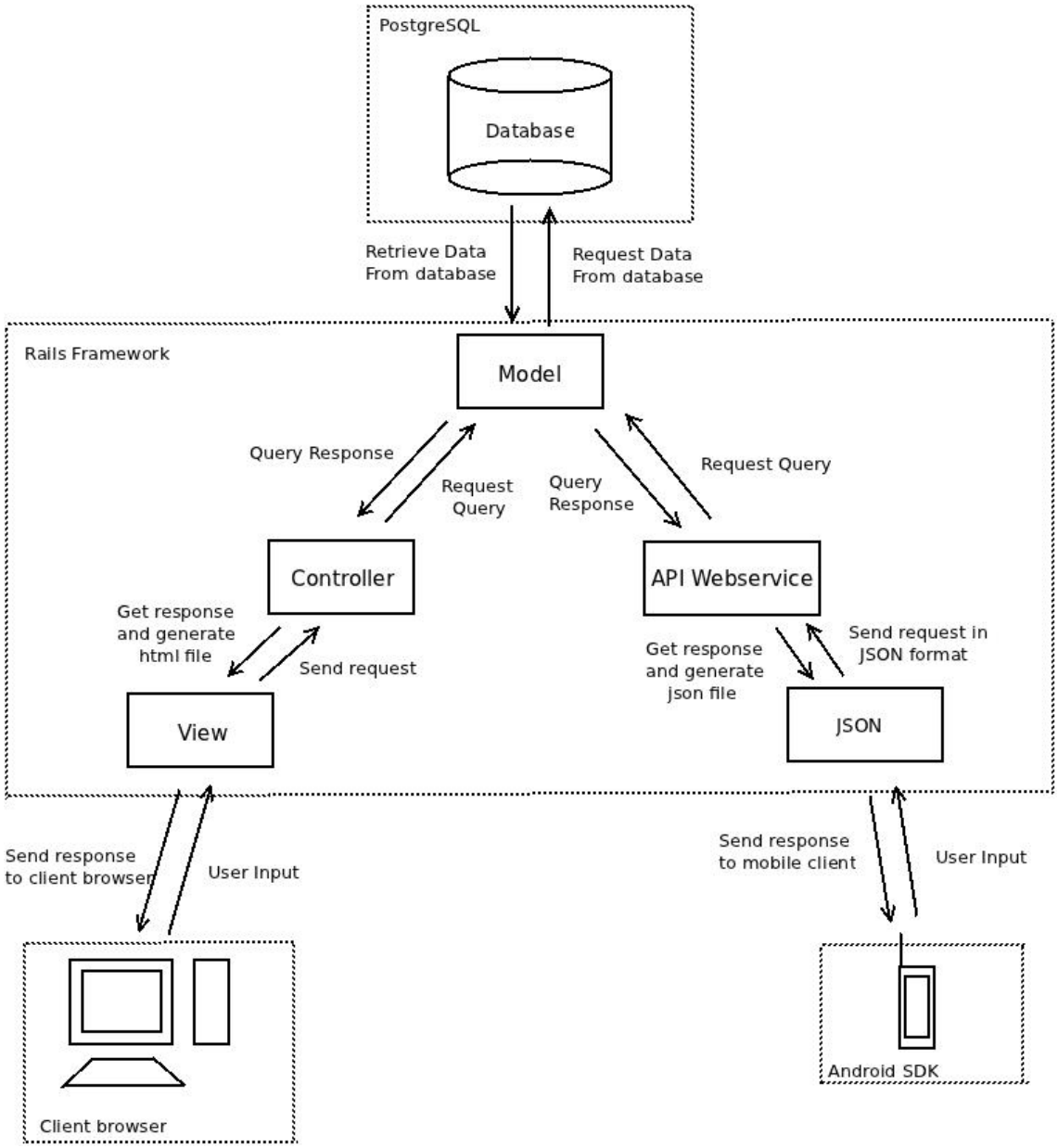


Figura 9: Diagrama mostrando integração entre as várias partes que compõem o sistema

Abaixo encontra-se um exemplo da dinâmica descrita, representando a implementação do sistema de autenticação de um usuário utilizando um dispositivo móvel.

```
1  Authorization auth = new Authorization(this.getApplicationContext());
2  String loginString = login.getText().toString();;
3  String passwordString = password.getText().toString();
4
5  new Thread(){
6   public void run() {
7    try {
8      auth.requestLogin(loginString, passwordString);
9    } catch (ParseException e) {
```

```
10        connectionError = e.getMessage();
11        mHandler.post(loginFailed);
12        return;
13      }
14      mHandler.post(loginSuccesful);
15    }
16  }.start();
```

O código acima resgata o login e senha entrados pelo usuário do sistema e guarda os valores nas variáveis *loginString* e *passwordString*. Essas variáveis são passadas com parâmetro para o método *requestLogin* da classe *Authorization*, onde será feita a requisição de autenticação ao *webservice*. Abaixo encontra-se o método responsável por receber tal requisição e processá-la.

```
1
2   def create
3     email = params[:email]
4     password = params[:password]
5
6     if request.format != :json
7       render :status => 406, :json => {:message=>"The request must be json
            "}
8       return
9     end
10
11    if email.nil? or password.nil?
12      render :status => 400, :json => {:message=>"The request must contain
            the user email and password."}
13      return
14    end
15
16    @user = User.find_by_email(email.downcase)
17
18    if @user.nil?
19      logger.info("User #{email} failed signin, user cannot be found.")
20      render :status => 401, :json => {:message => "Invalid email or
            passoword."}
21      return
22    end
23
24    if not @user.valid_password?(password)
25      logger.info("User #{email} failed signin, password \"#{password}\"
            is invalid")
26      render :status => 401, :json => {:message => "Invalid email or
            password."}
27    else
28      if @user.avatar.url != nil and @user.avatar.url != ''
```

```
29          @user.encoded_avatar = @user.avatar.url(:thumb)
30        end
31        render :status => 200, :json => {:user => @user.as_json}
32      end
33    end
```

O método *create* recebe como parâmetros strings representando um endereço de *email* e um *password*, e realiza os procedimentos necessários para efetuar a autenticação do usuário. Primeiro, é verificada a validade dos valores de *email* e *password*, pois nenhum dos dois pode ser vazio. Após isso, é realizada uma busca por usuários que possuam tal *email*. Uma vez verificada a existência de um cadastro para o *email* fornecido, compara-se o *password* fornecido com aquele contido em base de dados. Caso os valores estejam corretos, o método retorna o objeto *@user* em formato *JSON*, caso contrário, o método retorna uma mensagem de erro. Essa estrutura está presente não somente para autenticação, como também em outras partes importantes do aplicativo móvel, como a consulta de problemas e comentários de problemas,

A implementação de uma *API* foi feita em uma camada separada da aplicação web. Isso torna a aplicação mais eficiente pois separa o tratamento de requisições orinadas por disposivitos móveis e por *browsers* de clientes *desktops*, diminuindo assim o acoplamento entre os módulos e facilitando a manutenção dos mesmos.

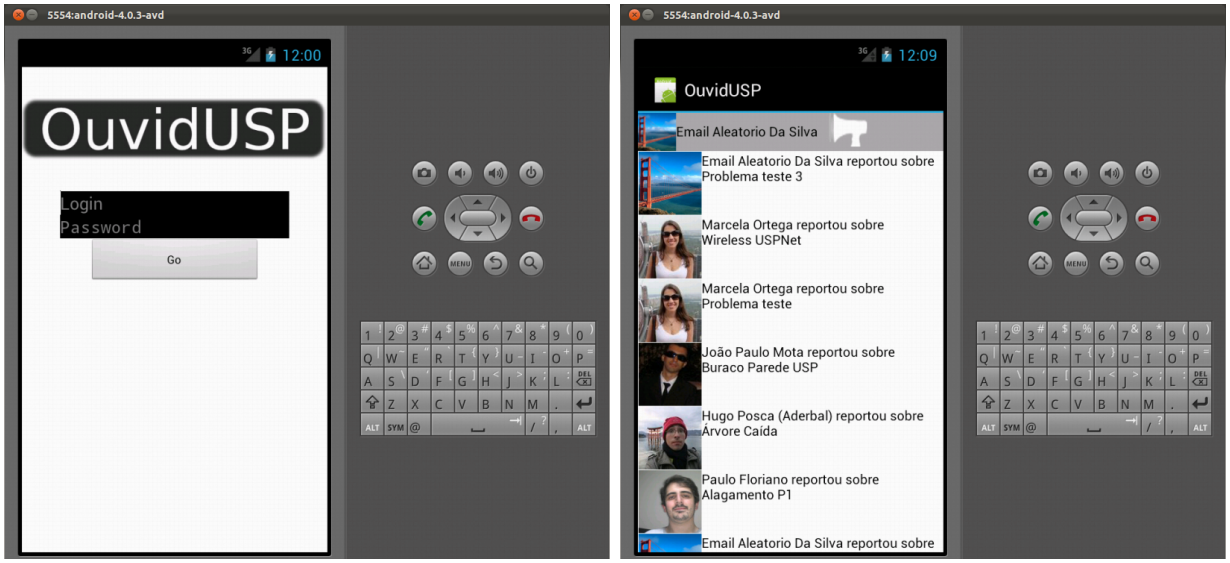


Figura 10: Telas de autenticação de usuários e lista de problemas

O aplicativo foi testado em emulação, onde é possível simular seu comportamento sem a necessidade de possuir um dispositivo móvel físico, e também em um celular da fabricante *Samsung*, em parceria com a empresa *Google*, modelo *Galaxy Nexus*. Em ambos os ambientes, o aplicativo mostrou bom desempenho em pontos críticos para aplicações móveis, como tempo de resposta.

# 6 Conclusão

A implementação do sistema descrito e as pesquisas sobre a origem e evolução de sistemas colaborativos permitem observar pontos importantes a cerca de colaboração através de programas de computador. Todo software colaborativo representa uma abstração de relações e atividades sociais, as quais ocorrem entre indivíduos de uma comunidade. Dada a subjetividade de tais relações, sua modelagem torna-se um processo complexo, e quando realizada de forma errônea pode prejudicar a aceitação do software. Devido a esse fato, alguns aspectos comuns em redes sociais atuais, como por exemplo a relação de amizade entre usuários, não foram implementados.

Outro aspecto importante a cerca de sistemas colaborativos é a importância de desenhar um sistema altamente escalável, aspecto esse o qual foi acentuado durante o desenvolvimento do sistema apresentado. Nota-se que sistemas colaborativos possuem alta demanda por novas funcionalidades, fazendo com que o processo de desenvolvimento seja rápido, com entregas constantes. Por isso, torna-se mais importante a implementação de boas práticas de programação, como a documentação do código e refatoração constante, para eliminar possíveis erros e garantir que o software permaneça sustentável. Não foi possível gerar uma documentação do sistema implementado, porém existem inúmeras ferramentas para documentação automática de código *ruby* e *java*, as quais são as linguagens utilizadas na implementação da ferramenta de colaboração.

Por fim, vale ressaltar a importância da implementação de um canal de comunicação com o usuário final da ferramenta, pois a melhor maneira de entender como a colaboração com auxílio de software pode ser melhorada em algum aspecto particular, é entender as necessidades reais das pessoas que a utilizam.

# 7 Referências


[1] V.M.R. Penichet, I. Martin, J.A. Gallud, M.D. Lozano, R. Tesoriero *A Classification Method for CSCW Systems*. Castilla-La Mancha, Albacete, Spain

[2] Tom Rodden, *A Survey of CSCW Systems*. Lancaster University, Lancaster, 2002.

[3] G. Henri Ter Hofte, *Working Apart Together: Foundations for Component Groupware*. Enschede, The Netherlands, 1998.

[4] Douglas C. Engelbart, William K. English, *A research center for augmenting human intellect*. Stanford Research Institute, Menlo Park, California 1968.

[5] Awakening Technology Groupware Explanation

[6] Johansen, R., *Groupware: Computer support for business teams*. New York: The Free Press 1988

[7] Jonathan Grudin, *Computer-Supported Cooperative Work: History and Focus*, University of California, Irvine, CA, 1994

[8] Developing for Android

## Parte II

# Subjetiva

# 8 Disciplinas relevantes ao trabalho

## 8.1 MAC0110, MAC0122 e MAC0323

As disciplinas de *Introdução a Computação*, *Princípios de Desenvolvimento de Algoritmos* e *Estrutura de Dados* são de vital importância para a formação de um Cientista da Computação e, por consequência, para a realização desse trabalho. Ao concluir as disciplinas citadas, o aluno começa a desenvolver a capacidade de resolução de problemas e o pensamento algorítmico que será de vital importância por toda a vida acadêmica e profissional. Tais disciplinas tiveram grande impacto na realização do trabalho.

## 8.2 MAC0211 e MAC0242

As disciplinas de *Laboratório de Programação I* e *Laboratório de Programação II* foram de grande importância para a realização desse trabalho pois possuem um caráter prático, onde o aluno, durante o período semestral das disciplinas, desenvolve um projeto por completo, passando por quase todas as fases de desenvolvimento de um software. Em particular, na disciplina de *Laboratório de Programação I*, a tarefa consistiu em implementar um jogo utilizando bibliotecas gráficas em linguagem C, o quê foi muito divertido e motivador.

## 8.3 MAC0342

A disciplina de *Laboratório de Programação Extrema* foi aquela com maior impacto para a realização desse trabalho. Com um caráter extremamente prático, a turma é dividida em grupos grandes e cada grupo desenvolve um software utilizando técnicas de desenvolvimento ágil. Essa disciplina é muito interessante pois, pela minha experiência de mercado, é aquela que mais se parece com a dinâmica típica de trabalho presente na maioria das empresas atualmente, dado que todos os aspectos envolvendo desenvolvimento ágil tornaram-se tendência em um mundo tecnológico onde as startups e empresas ditas jovens ditam as regras.

# 9 Desafios e frustrações

Uma vez que a idéia para o sistema e sua função foram definidas, acredito que o grande desafio desse trabalho consistiu em encontrar a maneira adequada de dispôr as informações para o usuário de forma clara, com o dinamismo e praticidade típicos das aplicações atuais. Para realizar tal tarefa, é necessário dominar algumas habilidades que, embora o aluno de Ciências da Computação tenha contato informalmente ao longo do curso, não costumam ser treinadas exaustivamente. São elas: planejamento de interfaces e design. Existem algumas ferramentas prontas de código aberto para auxiliar nessa parte, mas a prática ideal seria

trabalhar juntamente com um profissional especialista nessa área, o quê infelizmente não foi possível.

# 10 Futuro

Para o futuro, seria interessante implantar o sistema em uma comunidade e utilizar técnicas de análise de dados e recuperação de informação para extrair dados relevantes, como quais locais são mais problemáticos, qual a natureza mais comum dos problemas, e outros aspectos que ajudem a tomada de medidas preventivas. Com o sistema ativo em produção, provavelmente surgirão demandas e mais features para serem implementadas, o quê proporciona dinamismo ao projeto e manutenções constantes. Isso também seria muito divertido de observar, pois infelizmente, apesar de parecer um longo período de tempo para um projeto, um ano foi um período curto e algumas funcionalidades que são interessantes para o sistema não puderam ser implementadas.