

**UNIVERSIDADE ESTADUAL PAULISTA  
“JÚLIO DE MESQUITA FILHO”  
FACULDADE DE CIÊNCIAS  
DEPARTAMENTO DE COMPUTAÇÃO  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**LUCAS MORENO SILVA**

**SISTEMA ONLINE DE SUGESTÃO DE FILMES BASEADO NA  
ASSOCIAÇÃO ENTRE PERFIS DE USUÁRIOS**

**BAURU - SP  
2015**

LUCAS MORENO SILVA

**SISTEMA ONLINE DE SUGESTÃO DE FILMES BASEADO NA  
ASSOCIAÇÃO ENTRE PERFIS DE USUÁRIOS**

Trabalho de Conclusão de Curso do Curso  
de Bacharelado em Ciência da Computação  
da Faculdade de Ciências, Universidade  
Estadual Paulista “Júlio de Mesquita Filho”,  
campus Bauru.

Orientador: Prof. Dr. João Paulo Papa  
Coorientador: Luiz Miguel Axcar

BAURU - SP  
2015

# **SISTEMA ONLINE DE SUGESTÃO DE FILMES BASEADO NA ASSOCIAÇÃO ENTRE PERFIS DE USUÁRIOS**

Trabalho de Conclusão de Curso do Curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, campus Bauru.

Orientador: Prof. Dr. João Paulo Papa

Coorientador: Luiz Miguel Axcar

## **BANCA EXAMINADORA**

Prof. Dr. João Paulo Papa

UNESP - Bauru

Orientador

Prof. Dr. Aparecido Nilceu Marana

UNESP - Bauru

Prof. Dr. Humberto Ferasoli Filho

UNESP - Bauru

Dedico esse trabalho à meus pais, que sempre me incentivaram a escolher uma área que gostasse e sempre se esforçaram para que eu tivesse a melhor qualidade de vida possível

## **AGRADECIMENTOS**

Agradeço primeiramente à meus pais, que concordaram com minha decisão de morar fora da minha cidade natal para cursar Ciência da Computação em Bauru. Além de sempre me indicarem o caminho certo de se fazer as coisas: com muito esforço e estudo.

Também agradeço a meus colegas mais próximos que embarcaram comigo nessa aventura de morar numa cidade nova, organizando de grupos de estudos a festas, dividindo experiências ótimas nesses cinco anos.

*“Fracassar não é cair. Fracassar é continuar no chão.”*

(Anônimo)

## RESUMO

Devido à grande quantidade de filmes de diversos gêneros, atores e diretores nos dias atuais, fica cada vez mais difícil encontrar algum filme que realmente agrade uma pessoa. Muitas vezes, a pessoa define um filme com determinadas características como favorito e acredita que não há nenhum outro filme similar, quando na verdade, isso é muito improvável. O grande problema em encontrar filmes que parecem únicos, se deve ao fato do grande número de opções, como citado anteriormente. Para ajudar os fãs de filmes a encontrarem uma agulha nesse palheiro, algumas empresas desenvolveram sistemas de recomendação de filmes. O problema existe porque esses sistemas de recomendação tendem a sugerir filmes aclamados pela crítica e em alguns casos sugerir filmes que somente estão em seu catálogo para visualização (no caso de sistemas que reproduzem filmes), o que limita absurdamente as opções de sugestão. Com um banco de dados relacional, o sistema descrito nesse documento busca sugerir filmes através da associação entre perfis de usuário, utilizando como base dessa associação o novo conceito de *tags* quantitativas. Esse conceito também é utilizado para medir a intensidade com que cada *tag* aparece em cada filme. Com essa modelagem de dados, diversas formas de sugestões são possíveis, como as quatro descritas nesse documento.

**Palavras-chave:** sistemas de recomendação, banco de dados relacional, *tags* quantitativas.

## **ABSTRACT**

Due to the great amount of films of various genres, actors and directors today, it is increasingly difficult to find a movie that really pleases a person. Very often, the person defines a film with certain characteristics as favorite and believes that there is no other similar film, when in fact, this is very unlikely. The big problem in finding films that seem unique is because of the large number of options, as mentioned above. To help movie fans to find a needle in that haystack, some companies have developed recommender systems. The problem exists because these recommender systems tend to suggest critically acclaimed films, and in some cases suggest films that are only in their catalog for watching (for systems that play movies), which absurdly limit the options. With a relational database, the system described in this paper aims to suggest movies through the association between user profiles, using as the basis of this association the new concept of quantitative tags. This concept is also used to measure the extent which each tag appears in each film. With this data modeling, various forms of suggestions are possible, as the four described in this document.

**Keywords:** recommender systems, relational database, quantitative tags.



## LISTA DE FIGURAS

Figura 1 - Comunicação entre cliente e servidor web.....	25
Figura 2 - Componentes de uma requisição HTTP.....	26
Figura 3 - Rails MVC.....	31
Figura 4 - Dinamicidade.....	35
Figura 5 - Tipagem Forte.....	35
Figura 6 - Pivotal Tracker.....	40
Figura 7 - Formulário de cadastro de usuário.....	42
Figura 8 - Formulário de login de usuário.....	43
Figura 9 - Cadastros de <i>tags</i> de um filme.....	44
Figura 10 - Geenciamento de filmes.....	44
Figura 11 - Tela de detalhes de um filme.....	45
Figura 12 - Avaliação de comentários de um filme.....	46
Figura 13 - Exemplo da associação entre usuários e <i>tags</i> .....	49
Figura 14 - Lista de filmes na <i>home</i> .....	50

## LISTA DE TABELAS

Tabela 1 - Métodos do protocolo HTTP .....	27
Tabela 2 - Códigos de resposta do HTTP .....	28
Tabela 3 - URL's do padrão REST.....	29

## LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
HTTPS	<i>Hyper Text Transfer Protocol Secure</i>
IP	<i>Internet Protocol</i>
JSON	<i>JavaScript Object Notation</i>
SSL	<i>Secure Sockets Layer</i>
TCP	<i>Transmission Control Protocol</i>
URI	<i>Uniform Resource Identifier</i>
URL	<i>Uniform Resource Locator</i>
XML	<i>eXtensible Markup Language</i>

<b>1 INTRODUÇÃO.....</b>	<b>13</b>
1.1 Objetivos do trabalho.....	14
1.2 Justificativa.....	15
<b>2 FUNDAMENTAÇÃO TEÓRICA.....</b>	<b>16</b>
2.1 Sistemas de recomendação.....	16
2.2 Tags quantitativas.....	18
2.3 Programação orientada à objetos.....	20
2.3.1 ABSTRAÇÃO.....	21
2.3.2 ENCAPSULAMENTO.....	21
2.3.3 HERANÇA.....	22
2.3.4 POLIMORFISMO.....	22
2.4 Aplicações web.....	23
2.4.1 HTTP - HyperText Transfer Protocol.....	24
2.4.2 REST - REpresentational State Transfer.....	28
2.5 Modelo MVC.....	30
2.5.1 MODELS.....	32
2.5.2 VIEWS.....	33
2.5.3 CONTROLLERS.....	33
<b>3 MATERIAIS E MÉTODOS.....</b>	<b>34</b>
3.1 Ruby - A linguagem.....	34
3.2 Rails - O <i>framework</i> .....	37
3.3 Gems.....	38

<b>4 DESENVOLVIMENTO.....</b>	<b>40</b>
4.1 Cadastro e login de usuários.....	41
4.2 Gerenciamento de filmes.....	43
4.3 Tela de detalhes de um filme.....	44
4.4 Comentários.....	45
4.5 Avaliação de filmes.....	46
4.6 Sugestão de filmes.....	49
4.6.1 <i>USUÁRIOS</i> <i>COM</i> <i>USER_GROUPED_TAGS</i> <i>PARECIDAS</i> .....	50
4.6.2 <i>USUÁRIOS</i> <i>COM</i> <i>USER_TAGS</i> <i>PARECIDAS</i> .....	51
4.6.3 <i>FILMES</i> <i>COM</i> <i>TAGS</i> <i>DA</i> <i>TABELA</i> <i>USER_GROUPED_TAGS</i> .....	51
4.6.4 <i>FILMES</i> <i>COM</i> <i>TAGS</i> <i>DA</i> <i>TABELA</i> <i>USER_TAGS</i> .....	51
<b>5 CONSIDERAÇÕES FINAIS.....</b>	<b>52</b>
<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>53</b>
<b>APÊNDICES.....</b>	<b>55</b>
Apêndice A - Entradas do cabeçalho de uma requisição HTTP.....	55
Apêndice B - Modelagem final do banco de dados.....	56

## 1 INTRODUÇÃO

Em um mundo onde a quantidade de informações vem aumentando muito, achar algo de real interesse acaba se tornando uma tarefa complexa devido à grande diversidade e quantidade de dados em qualquer área da cultura humana.

Célia Turino (2006) define cultura como “uma composição de coisas, um pouco de entretenimento, de belas artes, de alta cultura, erudita, hermética, como também, a história, costumes, condutas, desejos, reflexões”. Dessa forma, fica clara a enorme diversidade em que um certo usuário pode buscar um dado específico. Além de todas essas vertentes que a palavra cultura atinge, vale lembrar que esses dados são repassados de geração a geração, de uma forma que só aumentam cada vez mais e mais. A cultura em si só existe porque antepassadas encontraram formas de imortalizar informações que considerassem valiosas para o “desenvolvimento da espécie”. A começar por conversas entre pai e filho e criação de desenhos em pedra até a criação de livros com milhares de exemplares.

Hoje em dia, no milênio da Tecnologia da Informação, existem outras formas de passar informações tanto técnicas quanto culturais. Se destacam principalmente os filmes e até mesmo os jogos que tentam retratar períodos históricos de grande influência para nossa sociedade, bem como a criação de metáforas que se aplicam perfeitamente a nosso mundo. Além daqueles que se arriscam a reescrever toda nossa história de uma maneira que cientistas tentam de todas as formas provar que tal ideia é ou não pura ficção.

Atualmente, por conta da facilidade da transmissão de informação, as opções do que ler, assistir e jogar são inúmeras. Hoje em dia, se um jogo é lançado do outro lado do mundo em uma língua pouco popular e chama a atenção dentre os demais, em questão de semanas ele terá seu lugar de destaque na internet, o que fará o mundo todo saber de sua existência. A questão é: nem sempre o que se destaca em um país do outro lado do mundo ou algo que é aclamado pelos críticos irá agradar a todos os consumidores de filmes, livros e jogos da mesma forma. Gosto se trata de uma característica completamente pessoal, e portanto, não é possível emitir uma lista dos melhores filmes que todos concordem. Cada um terá seu *ranking* baseado em métricas completamente pessoais.

Portanto, a sugestão de filmes, jogos e livros acaba se tornando algo mais complexo do que aparenta ser. Não basta sugerir somente os que aparecem nas listas dos considerados melhores, isso retornará os filmes que os críticos acham melhores, não o que a pessoa que espera sugestões acha melhor. Por conta disso, um bom sistema de sugestões deve se basear quase que completamente na criação de um perfil de usuário e buscar filmes, jogos ou livros que tenham pontos em comum com esse perfil, sem estabelecer que necessariamente os filmes que se encontram no *top* dos *rankings* irão agradar o usuário em questão. O algoritmo de sugestão de filmes deve se basear muito mais na associação entre características dos mesmos que numa ordenação por notas que outras pessoas deram.

Além disso, uma forma eficiente de sugerir dados para um usuário é procurar outro usuário com perfil parecido com o primeiro e sugerir dados que esse segundo possa gostar. Dessa forma, a busca não é por filmes parecidos com os o que o usuário gosta, mas de usuários com perfis parecidos. Apesar da implementação ser mais complexa, é provável que essa segunda opção seja mais satisfatória tendo em vista que é difícil encontrar uma boa métrica para medir todas as características dos filmes (mesmo com as *tags* quantitativas que serão usadas).

No meio desse oceano de dados, encontrar algo que interesse a quem procura acaba se tornando uma tarefa complexa de se executar, que é o objetivo desse sistema como um todo.

## **1.1 Objetivos do trabalho**

Desenvolver um sistema onde seja possível cadastrar filmes e usuários e, a partir da avaliação que um usuário fizer desses filmes, sugerir outros filmes baseados em seu gosto pessoal.

É possível destacar os seguintes objetivos específicos do trabalho:

- Modelar um banco de dados que seja possível relacionar filmes com *tags* de forma que um peso seja atribuído a essa associação;
- Cadastrar filmes, inserido as *tags* do mesmo com seus respectivos pesos, bem como seu ano de lançamento e descrição;
- Permitir que usuários se cadastrem no sistema;
- Criar um sistema de comentários em que o usuário possa fazer vários

comentários por filme bem como avaliar positivamente ou negativamente os comentários de outros usuários; e

- Sugerir filmes de diferentes formas baseado no histórico de avaliação do usuário. Tais sugestões devem partir, predominantemente, da associação entre perfis de usuário.

## **1.2 Justificativa**

Como a quantidade de dados em nosso mundo vem aumentando absurdamente, sugerir um filme a um usuário acaba se tornando uma tarefa difícil devido ao grande número de opções disponíveis. Os usuários acabam se contentando com sugestões dadas por sistemas que possuem um catálogo reduzido ou que envolvem algum tipo de estratégia de venda por se tratarem, algumas vezes, de sistemas de *e-commerce*. Assim, o sistema que foi desenvolvido busca sugerir filmes numa base sempre crescente (com sugestões dos usuários) sem qualquer interesse em vender algo, de forma que o objetivo será sempre focado em sugerir da forma mais precisa possível filmes que o usuário possa gostar.



## 2 FUNDAMENTAÇÃO TEÓRICA

Um algoritmo de sugestão de qualquer tipo de informação nos dias de hoje apresenta um certo grau de dificuldade devido à enorme quantidade e diversidade de dados encontrados em qualquer base de dados que se espera uma sugestão. Dessa forma, visando melhorar tanto a *performance* quanto a precisão das sugestões, são construídos sistemas de sugestão ou de recomendação.

### 2.1 Sistemas de recomendação

Segundo CORUMBRA (2010):

Um Sistema de Recomendação é um sistema que sugere recomendações ao usuário baseado em suas preferências. Estes sistemas desempenham a mesma função que um conhecido, o qual recomenda um restaurante ou um filme. As técnicas usadas em Sistemas de Recomendação podem ser classificadas em basicamente três tipos: (i) filtragem colaborativa, (ii) baseada em conteúdo e (iii) híbridos.

A recomendação baseada em conteúdo sugere itens similares aos que usuário já utilizou anteriormente. Este tipo de recomendação baseia-se na recuperação de informação e faz uso de várias técnicas de extração de informação. Documentos textuais são recomendados baseados na comparação entre seu conteúdo e o perfil do usuário. Estruturas de dados podem ser criadas extraindo características dos textos dos documentos. Geralmente são utilizados métodos para associar pesos às palavras presentes na coleção dos documentos. Existem vários métodos alternativos para criar pesos às palavras, como por exemplo, a técnica Tf-idf. Um Sistema de Recomendação é considerado puramente baseado em conteúdo quando as recomendações que são feitas ao usuário são baseadas apenas em conteúdo dos itens que já foram classificados pelo usuário anteriormente.

A filtragem colaborativa é bastante diferente da técnica de recomendação baseada em conteúdo. Ao invés de recomendar itens porque eles são similares aos itens que o usuário utilizou

anteriormente, a filtragem colaborativa recomenda itens que usuários similares utilizaram anteriormente. Dessa forma, é encontrado um conjunto de usuários cujos gostos sejam similares aos gostos do usuário em questão, os quais são denominados de vizinhos mais próximos. A pontuação dos itens que não foram utilizados pelo usuário é dada pela combinação das pontuações conhecidas do conjunto de vizinhos mais próximos. Em um Sistema de Recomendação puramente colaborativo, as recomendações para os usuários são feitas baseadas nas similaridades com outros usuários. Fazendo uso das recomendações de outros usuários, é possível lidar com qualquer tipo de conteúdo e receber itens com conteúdo diferente daqueles vistos anteriormente.

Diversos sistemas de recomendação usam a abordagem híbrida combinando técnicas dos métodos colaborativo e baseado em conteúdo, o que ajuda a melhorar o desempenho da filtragem colaborativa ou da técnica baseada em conteúdo quando são utilizados isoladamente. Existem diferentes formas para combinar as duas técnicas e gerar o método híbrido: (1) implementar o método colaborativo e o método baseado em conteúdo separadamente e depois comparar predições, (2) incorporar algumas características do método colaborativo no método baseado em conteúdo e vice-versa, e (3) construir um modelo unificado geral que incorpora características dos métodos colaborativo e baseado em conteúdo.

Dessa forma, podemos compreender que não há somente uma forma de sugerir dados a um usuário. Independente da forma com que o algoritmo de sugestão percorra, será necessária uma grande quantidade de dados para termos sugestões reais, já que em uma quantidade pequena de dados a sugestão em si se torna ou muito simples ou impossível.

AKITA (2013) diz que nos dias atuais, uma busca **nunca** deve retornar uma mensagem do tipo “nenhum dado foi encontrado”. Mesmo que não exista no banco dados registros com os filtros passados para busca, é necessário desenvolver uma espécie de *ranking* que liste os dados ordenados por grau de relevância que possa satisfazer a busca de alguma forma. No contexto de Akita, ele se referia ao método de busca chamado Elastic Search.

Independente do contexto, nos dias de hoje, buscas que não retornam nada devem ser evitadas ao máximo. Para isso, é necessário organizar os dados de uma forma que mais de um tipo de associação seja possível, para que, caso uma forma de sugestão, via algumas das associações, não retorne dados (ou um número muito pequeno de dados), outras entrem em ação, mostrando ao usuário um número considerável de dados.

Mais do que isso, como será detalhado no capítulo de desenvolvimento, é interessante sugerir dados ao usuário de diversas formas, bem como popular o banco com a origem dessas sugestões e se o usuário aprovou ou reprovou a sugestão. Com o passar do tempo, ao analisar esses dados, será possível notar qual método de sugestão foi mais eficiente e, a partir daí, adaptar o algoritmo de forma que se torne cada vez mais preciso.

## **2.2 Tags quantitativas**

Mesmo tendo em mente os métodos de sugestão disponíveis, é necessário definir a estrutura de como a aplicação irá funcionar, seja com um simples banco de dados, arquivos, inteligência artificial ou qualquer ferramenta necessária para implementar o método desejado.

Para esse trabalho, visando armazenar informações relevantes sobre as características dos filmes, foi-se utilizado um novo conceito, denominado **tags quantitativas**.

Assis (2009) faz a seguinte observação em relação à tags:

A forma mais comum de utilizar as tags é relacionar textos na internet. Em um mesmo site, blog ou serviço de blogs (como o Wordpress.com), ao escrever um texto e relacioná-lo a uma tag, faz-se com que todos os textos que são relacionados a essa tag possam ser encontrados mais facilmente.

Imagine que você encontrou um texto que fala sobre criação de gatos. Ao final do texto, você encontra a palavra chave “gato”. Ao clicar sobre essa palavra, você encontra outros textos relacionados à essa palavra, como uma resenha do filme Garfield, ou ainda um artigo sobre diferentes raças de gatos.

Como citado anteriormente, *tags* são palavras que servem para interligar interesses de forma muito simples: criando palavras-chave. A ideia de aplicar um número para representar a intensidade dessas palavras-chave foi um conceito sugerido pelo coorientador do autor desse trabalho que torna o banco uma fonte com informações mais precisas e com representação numérica referentes aos filmes.

Normalmente, os filmes são classificados por seu gênero/categoria. Realmente é uma forma muito funcional e intuitiva de classificar tipos de filmes. De fato, todo filme acaba pertencendo a um gênero principal. A proposta desse novo conceito é explicitar, mesmo estando dentro de um grupo de um gênero específico, o quanto outros gêneros/categorias estão presentes nesse filme. Mesmo que um filme seja definido como gênero ação, ele pode apresentar características de filmes de comédia, ficção ou qualquer outro gênero.

Um exemplo ótimo para esclarecer esse conceito é o filme “The Matrix” lançado em 1999. No site do IMDb<sup>1</sup>, ele é definido como um filme de ação e ficção científica. De fato, o filme possui características desses gêneros muito aparentes, porém, ao colocá-lo junto a outros filmes de ação, pode-se notar que esses não possuem quase nenhuma relação com o filme além de cenas de ação, tornando essa relação entre os filmes um elo muito fraco de associação de filmes “parecidos”.

Caso utilizássemos *tags* quantitativas para caracterizar esse filme, atribuiríamos um número para a intensidade do gênero ação presente no filme, bem como a intensidade do gênero ficção científica. Se definirmos esse intervalo de intensidade indo de 0 a 3, por exemplo, poderíamos dizer que o filme The Matrix tem intensidade 3 de ação e intensidade 3 de ficção científica. Dessa forma, mesmo sabendo que o filme realmente tem associação com filmes de ação, ele tem uma associação muito mais forte com filmes que possuam as mesmas intensidades de *tags* quantitativas. É muito mais provável que um filme com intensidade 3 de ficção científica e intensidade 3 de ação seja mais parecido com o filme do que outro de ação qualquer. Além disso, esse conceito serve não somente para filmes, mas também para definir o gosto do usuário do sistema de acordo com as avaliações dos

---

<sup>1</sup> IMDb (Internet Movie Database) - Site com informações sobre diversos filmes, seriados e todos profissionais envolvidos diretamente com a elaboração dos mesmos

filmes.

As *tags* quantitativas do usuário são representadas pela soma das *tags* dos filmes que ele avaliou positivamente e pela subtração das *tags* dos filmes que ele avaliou negativamente. A implementação do método de sugestão, bem como a organização do banco de dados será detalhado no capítulo de desenvolvimento.

### 2.3 Programação orientada à objetos

Nos dias de hoje, o paradigma de computação mais usado para desenvolvimento de softwares é o de programação orientada à objetos. Tal paradigma busca aproximar o mundo real do mundo virtual, tentando simular o mundo real dentro do computador. Para isso, nada mais natural do que utilizar os chamados objetos, que são as coisas que compõem nosso mundo real como diz DAVID (2007).

Na programação orientada à objetos, a chamada **classe** é um arquivo que descreve como os objetos, que são **instâncias** da mesma, que irão se “comportar”, ou seja, descreve a quais métodos esses objetos irão responder bem como os atributos que eles terão. No conceito de programação linear, um determinado código principal simplesmente liga as funções de acordo com a necessidade do usuário ou do próprio algoritmo em si. Na programação orientada à objetos, esse processo é muito mais trabalhado para que se torne mais natural a manipulação desses dados.

Como citado anteriormente, objetos podem ter métodos e atributos determinados na classe que os descrevem. Métodos são as funções relativas ao objeto em si, ou seja, algo que o objeto deveria fazer por si só, como um objeto do tipo humano, falar. Já os atributos são as características dos objetos, mais simples do que os métodos, somente guardam valores, sem chamar linhas de código. Como o nome de um objeto do tipo humano.

Além dos objetos, as classes também podem ter métodos e atributos. Esses são chamados de atributos e métodos **estáticos**, que são carregados junto com a classe, e não após a instanciação de um objeto de tal classe. No caso da classe Humano, um método estático coerente seria algo como “todos”, que retorna um vetor com todos os humanos.

Dentre os diversos métodos que objetos e classes podem ter, um apresenta características muito específicas: o construtor. Esse método é o responsável por alocar

uma parte da memória da máquina para um determinado objeto, ou seja, **instanciar** o objeto. Esse método pode ser reescrito, adicionando funcionalidades que façam sentido para o determinado objeto, como definir 0 para idade de um Humano recém instanciado.

A programação orientada à objetos possui diversos conceitos e definições que acabam a tornando uma área muito interessante de estudo. Dentre todos esses, vale citar os considerados por GASPAROTTO (2015) os quatro pilares da programação orientada à objetos: *abstração*, *encapsulamento*, *herança* e *polimorfismo*.

### 2.3.1 ABSTRAÇÃO

Para uma correta utilização da programação orientada à objetos, é preciso abstrair, imaginar como os objetos que estamos criando realmente irão se portar no sistema. Nessa parte do desenvolvimento é que devemos nos questionar quais métodos tais objetos e classes deverão responder, de que forma, e chamados por quem. Os atributos também acabam entrando nesse pilar da abstração por tratarem de características que são atribuídos a simples *bytes* de memória alocada numa máquina.

A abstração de um problema acaba se tornando uma das partes mais importantes no desenvolvimento de um sistema orientado a objetos, pois é necessário saber aplicar o grau correto de abstração, uma vez que, como classes podem representar qualquer coisa, um sistema pode chegar a modelar classes que acabem se tornando desnecessárias por não apresentarem um grau de importância expressivo comparadas às demais.

### 2.3.2 ENCAPSULAMENTO

O conceito de encapsulamento faz com que detalhes internos do funcionamento de métodos, tanto de objetos quanto de classes, seja oculto para o programador que os utiliza. A ideia é saber **o que** o método faz, e não **como** faz. Dessa forma, a implementação acaba se tornando modular, de forma que os métodos devam cumprir seus objetivos sem influência de códigos externos.

Além de modularizar a funcionalidade dos métodos, o encapsulamento

também define se os métodos e atributos serão públicos, privados ou protegidos. Essa separação é uma forma que o programador tem de garantir que nenhum método seja chamado onde não deve. Os métodos e atributos públicos, como o próprio nome diz, são aqueles que podem ser acessados de qualquer outra classe do sistema. Os privados são aqueles que só podem ser acessados de dentro da classe em questão e os protegidos aqueles que só podem ser acessados de dentro da classe ou por qualquer um de seus descendentes.

### 2.3.3 HERANÇA

A ideia de herança foi criada para relacionar classes que tenham uma forma hierárquica de associação. A ideia, baseada também no encapsulamento, é modularizar ainda mais os trabalhos dos métodos de objetos e classes.

Um exemplo de herança de uma classe Humano poderia ser uma classe Estudante, afinal de contas, todo estudante é humano e possui todas as características humanas. Nesse caso, característica como nome, idade e altura poderiam ser considerados atributos de um Humano enquanto que Curso e RA seriam atributos específicos de um estudante.

Na parte prática, um objeto da classe humano terá acesso a todos atributos e métodos protegidos de sua classe pai além de, obviamente, poder acessar seus atributos e métodos públicos.

### 2.3.4 POLIMORFISMO

O quarto e último pilar da programação orientada a objetos é referente a uma consequência da utilização de herança no desenvolvimento das classes. Na natureza, vemos animais que são capazes de alterar sua forma conforme a necessidade. É dessa ideia que vem o polimorfismo na orientação a objetos.

Um objeto da classe Estudante citada no tópico anterior, também pode ser tratado como um Humano, afinal de contas, sua classe é filha da classe Humano. Um trecho de código que espera um objeto do tipo Humano, também irá se contentar com um objeto do tipo Estudante, afinal de contas um estudante é um Humano.

O polimorfismo visa simplificar o desenvolvimento de algoritmos,

exigindo que o objeto não seja especificamente de certo tipo, mas que seja herdeiro de certo tipo. Dessa forma, pode-se garantir que mesmo objetos de classes diferentes respondam a métodos e atributos em comum (da classe pai). Assim, um objeto da classe Estudante é considerado polimórfico por poder ter forma tanto de Humano como de Estudante.

## 2.4 Aplicações web

Nos dias atuais, cada vez mais as pessoas estão ficando conectadas a internet. Hoje é comum pessoas acessarem a internet por seus computadores, *notebooks* e até mesmo celulares. Com a evolução da tecnologia e a chegada da internet das coisas<sup>2</sup>, algo que até então parecia ser um item de luxo ou entretenimento está se tornando uma ferramenta indispensável no cotidiano das pessoas.

Por conta disso, as aplicações de internet estão tomando o lugar das aplicações *desktop*, já que precisar instalar um *software* em sua máquina está se tornando um empecilho para uma sociedade que está se acostumando a fazer tudo que é necessário e acessando os mais diversos sistemas por um simples navegador em qualquer computador, *notebook* ou celular do mundo.

A dependência de internet já não é considerada um problema hoje em dia, já que é possível se conectar a ela muito facilmente via conexões sem fios de roteadores ou redes oferecidas por operadoras de telefonia. Outro fator que justifica esse aumento na construção de aplicações para internet é o aumento de velocidade tanto de *download* quanto de *upload* das conexões. Fator que até então era limitador para essas aplicações, pois não era possível transmitir uma quantidade muito grande de dados porque a velocidade era baixa e comprometia a experiência de uso do usuário.

Outra vantagem que esse aumento de velocidade trouxe é o da terceirização de serviços, os chamados *web services*. Como a velocidade só tende a aumentar, transmitir dados entre servidores está se tornando um problema irrelevante.

---

<sup>2</sup> ZAMBARD (2014): A “Internet das Coisas” se refere a uma revolução tecnológica que tem como objetivo conectar os itens usados do dia a dia à rede mundial de computadores. Cada vez mais surgem eletrodomésticos, meios de transporte e até mesmo tênis, roupas e maçanetas conectadas à Internet e a outros dispositivos, como computadores e smartphones.



Dessa forma, ao invés de concentrar todo processamento de uma requisição em somente um servidor, é possível se comunicar com outro servidor, “pedindo” um cálculo qualquer, fazendo o trabalho em paralelo. No final, a resposta chegará ao usuário final mais rapidamente.

A terceirização de serviços não se trata necessariamente de quebrar todo o processamento necessário em várias pequenas tarefas iguais, mas definir tarefas diferentes em que se conhece algum serviço que as execute com maior estabilidade e precisão.

Por exemplo, imaginando um caso onde se deseja implementar um sistema que envie SMS<sup>3</sup> para um determinado grupo de usuários cadastrados. Ao invés de compreender todo protocolo que existe no envio de SMS e comprar todos equipamentos e tecnologias necessárias para o envio dessas mensagens, é muito mais conveniente se comunicar com um serviço que receba requisições de envio e faça todo esse serviço, devolvendo ao requisitante se o processo ocorreu corretamente ou não.

Mesmo que tais serviços sejam, algumas vezes, pagos, eles apresentam uma alternativa muito mais conveniente e segura do que a implantação de todos equipamentos e tecnologias necessárias. Como um *web service* tem uma especialidade, é natural que sua qualidade tenda a aumentar, bem como sua variedade de funcionalidades. Para a comunicação entre estes serviços, bem como a navegação comum em navegadores de internet, utiliza-se o protocolo HTTP, conforme descrito na próxima seção.

#### 2.4.1 HTTP - HyperText Transfer Protocol

De acordo com CCM (2016), HTTP é o protocolo mais utilizado na internet desde 1990. Até a versão 0.9, o protocolo destinava-se à transmissão unicamente de dados pela internet (páginas escritas em HTML<sup>4</sup>). A partir da versão

---

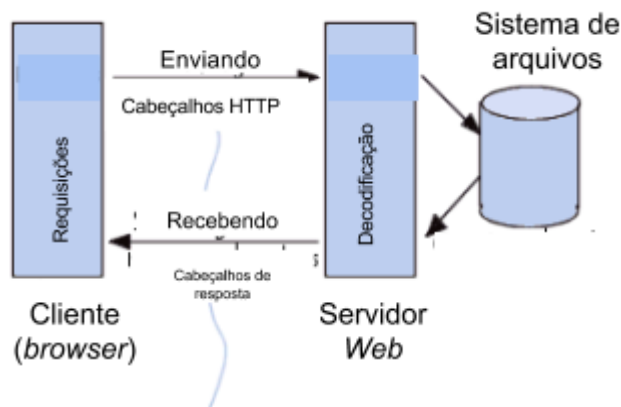
<sup>3</sup> SMS (*Short Message System*) - Conhecido também como mensagem de texto, é um serviço oferecido pelas operadoras que permite, a partir de um aparelho celular, o envio de uma mensagem de até 160 caracteres para outros aparelhos.

<sup>4</sup> De acordo com EIS (2011): HTML é uma das utilizadas para desenvolver *websites*. O acrônimo HTML vem do inglês e significa Hypertext Markup Language ou em português Linguagem de Marcação de Hipertexto. O HTML é a linguagem base da internet. Foi criada para ser de fácil entendimento por seres humanos e também por máquinas, como por exemplo o Google ou outros

1.0, o protocolo começou a adotar um cabeçalho que especifica o tipo de dado que está sendo transmitido.

O objetivo do protocolo é a transferir dados entre um cliente e um servidor. Essa transferência acontece em dois tempos como mostrado na Figura 1.

Figura 1 - Comunicação entre cliente e servidor web.



Fonte: <http://br.ccm.net/contents/266-o-protocolo-http>

1. O cliente faz um pedido HTTP, e
2. O servidor trata o pedido e seguidamente envia uma resposta HTTP.

Um pedido HTTP, como mostrado na Figura 2, é formado, basicamente, por 3 partes:

1. Linha de pedido - formada pelo método, a URL<sup>5</sup> acessada pelo cliente e a versão de protocolo que está sendo utilizada pelo cliente;
2. Cabeçalho - Conjunto de informações suplementares sobre a informação que está sendo enviada; e
3. Corpo - Opcional, é onde as “exigências” do pedido são informadas e organizadas de uma forma que o servidor que receberá o pedido as compreenda.

---

sistemas que percorrem a internet capturando informação.

<sup>5</sup> URL (*Uniform Resource Locator*) - URL se refere ao endereço de rede no qual se encontra algum recurso informático, como por exemplo um arquivo de computador ou um dispositivo periférico. É o endereço de *websites* que acessamos por navegadores de internet

Figura 2 - Componentes de uma requisição HTTP.



Fonte: [https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP\\_Basics.html](https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

Cada método HTTP tem uma funcionalidade definida como é mostrado na Tabela 1:

Tabela 1 - Métodos HTTP.

<b>Método</b>	<b>Descrição</b>
GET	Requisita um representação do recurso especificado (O mesmo recurso pode ter várias representações, ao exemplo de serviços que retornam XML e JSON)
HEAD	Retorna os cabeçalhos de uma resposta (sem o corpo contendo o recurso)
PUT	Requisita que um entidade seja armazenada embaixo da URI fornecida. Se a URI se refere a um recurso que já existe, ele é modificado; se a URI não aponta para um recurso existente, então o servidor pode criar o recurso com essa URI
POST	Envia uma entidade e requisita que o servidor aceite-a como subordinada do recurso identificado pela URI
DELETE	Apaga o recurso especificado
TRACE	Ecoa de volta a requisição recebida para que o cliente veja se houveram mudanças e adições feitas por servidores intermediários
CONNECT	Converte a requisição de conexão para um canal TCP/IP transparente, usualmente para facilitar comunicação criptografada com SSL (HTTPS) através de um proxy HTTP não criptografado
OPTIONS	Retorna os métodos HTTP que o servidor suporta para a URL especificada
PATCH	Usado para aplicar modificações parciais a um recurso

Fonte:

<http://pt.stackoverflow.com/questions/9419/quais-s%C3%A3o-os-m%C3%A9todos-de-requisi%C3%A7%C3%A3o-http-e-quais-s%C3%A3o-as-diferen%C3%A7as-entre-eles>

Assim como os métodos, as entradas do cabeçalho também são definidas claramente por esse protocolo, como mostrado no apêndice A desse

documento.

Após uma requisição ser enviada e processada, o servidor devolve ao cliente uma resposta HTTP, que também segue uma lista de itens que podem ser especificados em seu cabeçalho, assim como uma lista de códigos de resposta, que permite com que a aplicação que obteve tal resposta consiga identificar o que aconteceu a partir de um simples número. Esses códigos são representados por 3 dígitos, onde o primeiro indica a que tipo de resposta esse código pertence:

Tabela 2 - Códigos de resposta do HTTP.

<b>Código</b>	<b>Mensagem</b>	<b>Descrição</b>
10X	Mensagem de informação	Estes códigos não são utilizados na versão 1.0 do protocolo
20X	Sucesso	Estes códigos indicam o bom desenrolar da transação
30X	Redireção	Estes códigos indicam que o recurso já não está no lugar indicado
40X	Erro devido ao cliente	Estes códigos indicam que o pedido está incorreto
50X	Erro devido ao servidor	Estes códigos indicam que houve um erro interno do servidor

Fonte: <http://br.ccm.net/contents/266-o-protocolo-http>

#### 2.4.2 *REST - REpresentational State Transfer*

Seguindo os padrões definidos pelo protocolo HTTP, diversas formas de comunicação (seguindo essa regras) foram criadas para facilitar a comunicação entre diferentes servidores, além de padronizar a forma como os dados devem ser manipulados seguindo simplesmente o formato de sua URL e o tipo de método utilizado.

REST é um estilo de arquitetura de *software* desenvolvido para aplicações *web* que vem se destacando pela sua simplicidade e sua funcionalidade. Foi criado em 2000 por Roy Thomas Fielding em sua dissertação "Architectural Styles and the Design of Network-based Software Architectures".

Diversas API's da internet seguem esse padrão, além de diversos *frameworks*<sup>6</sup> facilitarem a criação desses padrões.

O REST, basicamente, define um padrão de URLs que deve ser seguido para manipular um determinado tipo de dado. Via essas URLs, é possível executar todos os comandos necessários para controle de dados de qualquer informação. O padrão para o caso de uma coleção de filmes é mostrado na Tabela 3.

Tabela 3 - URLs do padrão REST.

URL	Método HTTP	Ação	Descrição
/movies	GET	index	Retorna a coleção de todos filmes
/movies/new	GET	new	Retorna um formulário para a criação de um novo filme
/movies	POST	create	Cria um novo filme com os parâmetros enviados via POST
/movies/:id	GET	show	Retorna o filme com id especificado na URL
/movies/:id/edit	GET	edit	Retorna um formulário para edição do filme com id especificado na URL
/movies/:id	PATCH	update	Atualiza o filme com id especificado na URL com os parâmetros enviados via PATCH
/movies/:id	DELETE	destroy	Apaga o filme com id especificado na URL

Fonte: Autor

Por padrão se utiliza o inglês em palavras-chave das URLs (*new* e *edit*), assim como no nome das ações. Além disso, como será explicado no tópico a seguir, é preferível que se utilize também essa linguagem para nomear as coleções do sistema, para que, nesse caso, o *framework* utilizado nesse projeto identifique os

---

<sup>6</sup> *Frameworks* - Em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Um *framework* pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Ao contrário das bibliotecas, é o *framework* quem dita o fluxo de controle da aplicação, chamado de Inversão de Controle - WIKIPEDIA (2015b)

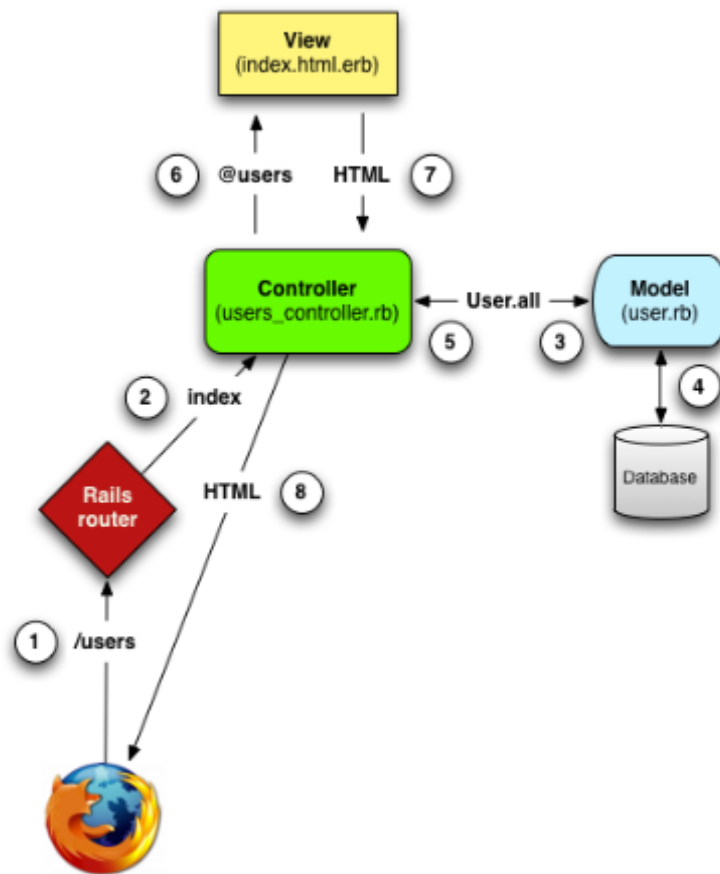
arquivos necessários naturalmente, sem a necessidade de qualquer tipo de configuração.

## **2.5 Modelo MVC**

Em 1979, Trygve Reenskaug apresentou uma nova arquitetura de desenvolvimento para aplicações interativas. Nessa arquitetura, existem três elementos principais: Models, Views e Controllers.

A Figura 3 representa como o modelo MVC funciona em uma aplicação Rails a partir de uma requisição via um navegador de internet. Além dos 3 componentes do MVC, a imagem também retrata um componente muito importante numa aplicação baseada em Ruby on Rails, o roteador, que define qual *controller* e qual ação irá responder a requisição recebida.

Figura 3 - Rails MVC.



Fonte: <http://i.stack.imgur.com/IOJBS.png>

O passo 1 ocorre quando o usuário acessa um endereço qualquer via um navegador (como /users) executando uma requisição HTTP. A aplicação, seguindo o padrão REST, recebe essa requisição e o roteador define qual *controller* irá tratá-la. Rails segue uma cultura de *convention over configuration*, como será explicado no capítulo de materiais. Essa é uma aplicação dessa cultura. Se a aplicação recebeu uma requisição para acessar a página *users*, por padrão, um *controller* chamado UsersController definido em um arquivo *users\_controller.rb* será responsável por responder a essa requisição. Olhando a tabela das URL's do REST, sabemos que quando o usuário acessa a /users pelo método GET, a ação *index* do *controller* responsável deve ser chamada. Seguindo os padrões de nomenclatura, esse passo será feito automaticamente (passo 2).

A ação *index* do *controller* UsersController, por sua vez, pede ao *model*



User, que tem acesso ao banco, a coleção de todos os usuários do sistema (passo 3). O *model* acessa o banco, obtendo a coleção de usuários (passo 4) e devolve para o *controller* (passo 5). Novamente no *controller*, com os dados dos usuários em mãos, uma *view* é chamada para mostrar as informações e a coleção de usuários é passada para ela (passo 6).

Na *view*, a forma como os usuários serão listados é definida (geralmente via uma página HTML), e essa forma é passada novamente para o *controller* (passo 7), que finalmente devolve os dados formatados para o usuário (passo 8) via uma resposta HTTP. Cada um dos itens do MVC será explicado mais detalhadamente nos tópicos a seguir.

### 2.5.1 MODELS

Os *models* são basicamente as partes da aplicação que representam os dados em si, geralmente sendo associados a tabelas do banco de dados. Porém, além de serem responsáveis pela interface de armazenamento e obtenção de dados no banco, os *models* também são os responsáveis por aplicar as chamadas regras de negócio nos dados. Como diz RUBY (2013, p. 29), os *models* se comportam como porteiros e também como armazenadores de dados. São os únicos, em teoria, que devem acessar diretamente os dados do banco e aplicar quaisquer regras impostas pela aplicação antes de devolvê-los ao *controller*.

Além disso, as validações dos dados também devem ser mostradas nos *model*, afinal de contas, só algo que cumpre as regras de negócio deve ser salvo no banco de dados. As validações geralmente determinam quais campos são obrigatórios bem como se estão no formato desejado ou, em alguns casos, verificam se o dado que está sendo inserido já não existe no sistema (unicidade).

Mesmo já tendo a funcionalidade de ser o porteiro dos dados, os *models* também concentram a lógica mais “pesada” da aplicação. Um projeto Rails bem estruturado, de acordo com a maioria da comunidade, apresenta *models* com muitas linhas, chamados informalmente de *models* gordos (*fat models*) quando necessário. A qualidade de código geralmente é inversamente proporcional à complexidade da lógica. Ninguém se orgulha de funções com muitas linhas ou muito complexas, mas, se

elas realmente precisam existir, que fiquem nos *models*.

### 2.5.2 VIEWS

Essas são as partes da arquitetura responsáveis por mostrar as funcionalidades do sistema ao usuário. Elas geralmente exibem informações dos mais diversos tipos, que geralmente estão armazenadas nas tabelas relativas aos *Models*. Porém, seu objetivo é focar somente na listagem dessas informações, sem se preocupar com qualquer forma de validação ou formatação de dados para salvar no banco.

A lógica presente nessa camada deve ser quase nula, apenas implementações como a verificação se um dado deve ser ou não exibido naquele momento, ou a exibição dos dados de um vetor recebido pelo seu controller correspondente. As *views* são, basicamente, formadas por HTML, o mínimo necessário de código Ruby e importações de arquivos CSS e JavaScript, que devem ficar em seus respectivos diretórios.

### 2.5.3 CONTROLLERS

Os *controllers* são os responsáveis por orquestrar a aplicação. Respondem a comandos exigidos pelos usuários nas *views*, e os passa para os *models*. É o intermediário entre as outras duas camadas da aplicação. A característica mais importante de um modelo MVC bem estruturado é a presença de *controllers* magros, ou seja, com pouco código.

A parte lógica, de cálculos e acesso de dados, deve se concentrar sempre nos *models*, os *controllers* apenas se comunicam com esses e repassam as informações para *view*, de forma que a quantidade de código presente nestes deve ser mínima.

### 3 MATERIAIS E MÉTODOS

Dentre os materiais e métodos utilizados, os mais relevantes para realização desse trabalho serão descritos nos tópicos a seguir.

#### 3.1 Ruby - A linguagem

A linguagem utilizada no desenvolvimento do sistema foi a Ruby, criada por Yukihiro Matsumoto no Japão e publicada em 1995 de acordo com a WIKIPEDIA (2015c).

É considerada uma linguagem de alto nível por ter um bom nível de abstração, além de ter características que diminuem o tempo de desenvolvimento ao mesmo tempo que não permite com que o usuário fuja das melhores práticas de desenvolvimento da linguagem:

- **Dinamicidade:** Termo usado para descrever linguagens que executam diversas tarefas em tempo de execução que outras linguagens executam na hora da compilação. Um exemplo seria atribuição de um valor a uma variável. Em Ruby, não é necessário especificar o tipo das variáveis. Elas terão seu tipo modificado de acordo com o valor que é atribuído à elas. No exemplo mostrado na Figura 4, notamos que, quando a variável *ano* recebe um valor com aspas ('2016'), ela é definida como uma variável do tipo "String" e, quando recebe um número sem aspas (2016), ela é transformada numa variável do tipo "Fixnum". Apesar de ser dinâmica, Ruby é uma linguagem de tipagem forte, como será esclarecido abaixo:

Figura 4 - Dinamicidade.

```
ano = '2016'  
=> "2016"  
ano.class  
=> String  
ano = 2016  
=> 2016  
ano.class  
=> Fixnum
```

Fonte: Autor

- Tipagem forte: Apesar de ser dinâmica, Ruby possui tipagem forte, ou seja, é necessário fazer transformações na hora de trabalhar com operadores ou funções que esperam um determinado tipo de variável. Se uma String for somada com um Fixnum, a linguagem levantará um erro, pois realmente não é algo natural de se fazer. Porém, se um Fixnum for transformado em String e “somado” com outra String, ocorrerá uma concatenação; No caso inverso, ocorrerá uma soma, como mostrado na Figura 5.

Figura 5 - Tipagem forte.

```
numero_1 = '123'  
=> "123"  
numero_2 = 321  
=> 321  
numero_1 + numero_2.to_s  
=> "123321"  
numero_1.to_i + numero_2  
=> 444
```

Fonte: Autor

- Suporte a metaprogramação; e
- Sobreescrever classes nativas facilmente.

Além das características supracitadas, é interessante citar que

Matsumoto afirmou que a linguagem foi desenvolvida tanto para a produtividade do programador quanto para sua diversão, sempre reforçando que *design de software* deve ser focado nas necessidades das pessoas, e não dos computadores.

Em 2008, na Google Tech Talk<sup>7</sup>, Matsumoto disse - WIKIPEDIA (2015b):

“Muitas vezes as pessoas, especialmente os engenheiros de computação, concentram-se nas máquinas. Eles pensam: "Ao fazer isso, a máquina irá rodar mais rápido. Ao fazer isso, a máquina irá rodar de forma mais eficaz. Ao fazer isso, a máquina alguma coisa alguma coisa alguma coisa." Eles estão se concentrando em máquinas. Mas, na verdade temos de nos concentrar nos seres humanos, sobre a forma como os seres humanos se preocupam em programar ou operar a aplicação nas máquinas. Nós somos os mestres. Eles são os escravos.”

Com essa afirmação, podemos entender que toda argumentação sobre a facilidade do desenvolvimento de programas com essa linguagem tem o objetivo de focar nas relações entre as pessoas, ao invés da relação do programador com seu computador. Os problemas a serem resolvidos precisam ser relacionados ao andamento do projeto, e não à algum problema específico da linguagem que requer tempo para ser analisado e que, conseqüentemente, acabaria atrasando ainda mais o andamento do projeto.

Seguindo essa linha de raciocínio, Ruby também é conhecido por adotar o POLA (Principle Of Least Astonishment), que é definido, basicamente, pela seguinte frase: “Se um recurso necessário tem um alto fator surpresa, pode ser necessário redesenhar o recurso.” (WIKIPEDIA, 2015d).

### **3.2 Rails - O Framework**

Quando se fala de desenvolvimento para web utilizando Ruby, a utilização do *framework* Rails está quase sempre associada. Juntas, essas tecnologias formam a famosa Ruby on Rails, que vem ganhando espaço cada vez mais no mercado de trabalho por apresentar um tempo de desenvolvimento geralmente menor

---

<sup>7</sup> *Google Tech Talk* - Evento patrocinado pela Google que traz diversos convidados de diversas áreas para falar sobre sua área de atuação nos diversos escritórios da empresa ao redor do mundo.

do que seus concorrentes além de ser uma linguagem de alto nível que evita que o programador perca tempo repetindo algoritmos ou lidando com problemas pequenos.

Rails segue à risca o padrão de nomenclatura definido pela linguagem, além disso, adiciona duas novas culturas ao desenvolvimento de *software*: *DRY* (*Don't Repeat Yourself*) e *CoC* (*Convention over Configuration*). O primeiro, que em português significa “não repita você mesmo”, prega a ideia exata que essa frase representa. Basicamente, se você está replicando um trecho de código, há algo de errado com a estrutura da sua aplicação.

Para criar um projeto, basta executar o comando “rails new <nome\_do\_projeto>” numa máquina que tenha tanto o Ruby como o Rails instalados. Quando um novo projeto é criado, diversas pastas são criadas, como *models*, *views* e *controllers*, de uma forma que cada arquivo tenha um local específico para ser colocado, evitando assim, a replicação de código.

O segundo conceito (CoC), que significa “convenção sobre configuração”, atua junto com o DRY para facilitar o desenvolvimento de projetos. Esse conceito define que, pelo menos no Rails, é mais fácil e produtivo seguir os padrões estabelecidos pelo *framework* do que inventar um novo padrão.

Se os padrões de nomenclaturas para nomes de arquivos, pastas, nomes de métodos e todas as estruturas de programação forem os mesmos do *framework*, pouco código adicional será gerado para um funcionamento básico.

Um exemplo disso pode ser observado na Figura 3, onde pode ser notado que, com quase nenhuma configuração, caso uma URL “/users” for requerida por um usuário via GET, assume-se que, seguindo o padrão REST (Tabela 3), todos os usuários do banco de dados devem ser retornados. Para tal, um *controller* denominado “UserController” e sua ação *index* será chamada obtendo dados de uma tabela denominada “users” via um *model* denominado “User” para depois carregar uma *view* localizada em “/users/index.html.erb”.

Percebe-se que o *framework* trabalha com o nome da classe “User”, pluralizando-o, deixando-o com caixa baixa ou concatenando o sufixo *Controller*. Todos os arquivos e classes citados no parágrafo acima são gerados simplesmente

se o comando “rails generate scaffold User name:string” for executado. Isso mostra que o conceito de CoC realmente é funcional e também segue o DRY, facilitando muito o desenvolvimento do projeto.

### 3.3 *Gems*

Além de todas vantagens que a linguagem Ruby já apresenta por sua facilidade de codificação, outra particularidade que envolve a utilização da mesma é a disponibilidade de utilizar as chamadas *gems*.

*Gems* são diversas bibliotecas *opensource*<sup>8</sup> que podem ser encontradas em [www.rubygems.org](http://www.rubygems.org). Essas *gems* oferecem uma infinidade de funcionalidades que são facilmente integradas com a aplicação desejada por se tratarem de códigos sempre em manutenção e evolução pelas mãos da comunidade.

Essa é uma grande vantagem dos códigos abertos pois, se alguém já implementou uma funcionalidade de forma modularizada, previu vários possíveis problemas que poderiam ocorrer e otimizou o código de tal funcionalidade, não é necessário que um programador que necessite da mesma, escreva outro código fazendo a mesma coisa (possivelmente de forma pior). O mais prático é reutilizar o código aberto e, se possível, colaborar com o mesmo, propondo soluções para problemas encontrados e sugestões para melhorar as funcionalidades e a *performance*.

A evolução da computação nos dias de hoje está muito mais associada em ligar diversos módulos para propor soluções inovadoras do que propor coisas que façam as mesmas coisas de formas diferentes. Utilizando um código em manutenção constante, em que um time já gastou um certo tempo ajustando o código e corrigindo erros, há uma certa garantia que o código em questão possui uma qualidade maior do que um código escrito do zero, sem qualquer tipo de teste ou apoio da comunidade *opensource*.

No capítulo de desenvolvimento serão descritos os passos em que foram

---

<sup>8</sup> *Opensource* - Open source é um termo em inglês que significa código aberto. Isso diz respeito ao código-fonte de um software, que pode ser adaptado para diferentes fins. O termo foi criado pela OSI (Open Source Initiative) que o utiliza sob um ponto de vista essencialmente técnico - CANALTECH

utilizadas *gems* para o desenvolvimento da aplicação descrita nesse documento, de forma que fique claro a facilidade da utilização das mesmas, bem como a obtenção rápida dos resultados esperados.

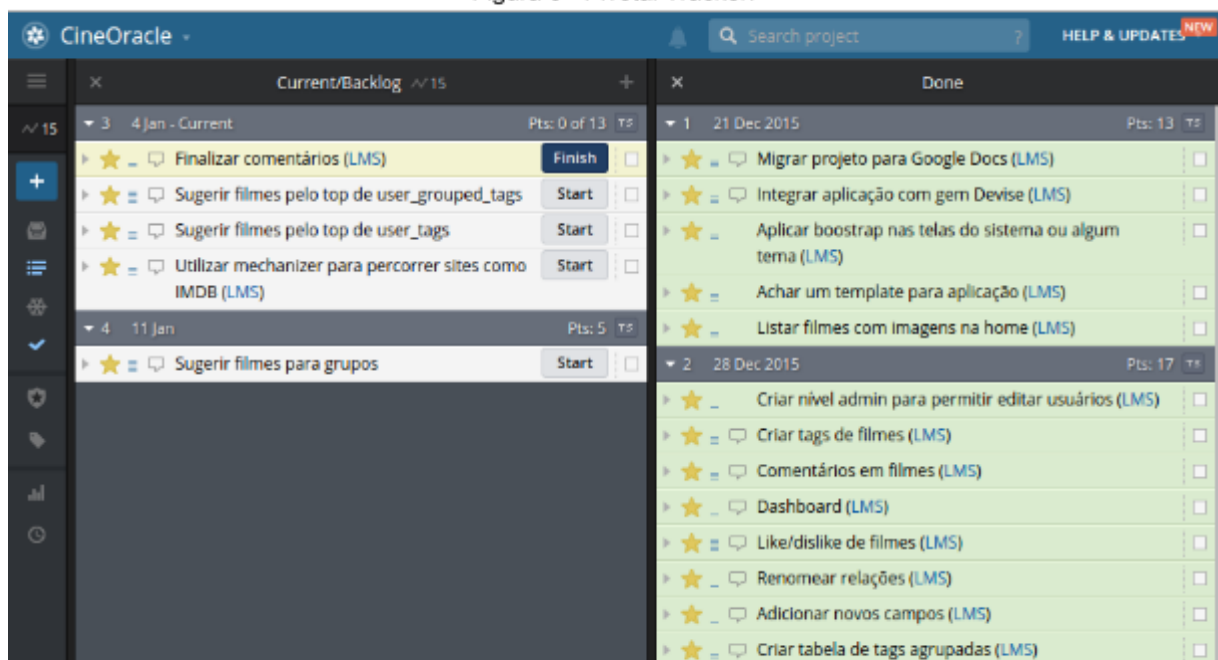


## 4 DESENVOLVIMENTO

Para o acompanhamento do desenvolvimento e verificação de viabilidade do projeto, a ferramenta Pivotal Tracker (Figura 6) foi utilizada ([www.pivotaltracker.com](http://www.pivotaltracker.com)). Essa ferramenta conta com diversas funcionalidades que desburocratizam a criação e associação de tarefas. Apesar de ter sido desenvolvida para equipes, a ferramenta cumpriu perfeitamente as exigências necessárias por um projeto desenvolvido somente por uma pessoa. Com uma interface simples, limpa e com a exata quantidade necessária de informações, é possível observar com exatidão o andamento do projeto, tendo a opção de redefinir a fila de prioridades de forma rápida, bem como a criação e remoção de tarefas a serem desenvolvidas.

Além da definição da fila de prioridades e o que deve ser feito em cada tarefa, essa ferramenta também dá a opção de definir quantos pontos cada tarefa irá levar. Esses pontos são referentes ao esforço necessário para executar cada tarefa.

Figura 6 - Pivotal Tracker.



Fonte: Autor

Nas configurações do projeto, há a opção de definir um limite de esforço semanal, de forma que, com as tarefas criadas e seus pontos associados, é possível prever em qual semana as tarefas serão concluídas se seguirem a fila de prioridades.

Tendo tal ferramenta para o acompanhamento das tarefas, os próximos subitens irão descrever os principais passos que foram tomados no decorrer do projeto em ordem cronológica.

#### 4.1 Cadastro e login de usuários

Com a definição dos materiais e métodos, o projeto se iniciou pelo desenvolvimento da lógica de autenticação do usuário. Para tal funcionalidade, foi utilizada uma *gem* (biblioteca) da comunidade Ruby chamada ***devise***. Essa é uma das *gems* mais baixadas de toda comunidade e apresenta uma ótima solução para autenticações de usuários nos mais diversos sistemas. Seguindo as instruções mostradas na página do Github<sup>9</sup> da *gem* (<http://github.com/plataformatec/devise>), a integração se apresentou fácil e rápida.

As *gems* do Ruby, como será mostrado mais algumas vezes nesse documento, apresentam uma característica que demonstra muito bem a forma como camadas devem funcionar dentro de uma aplicação: não é necessário saber exatamente como todas bibliotecas utilizadas foram desenvolvidas, tampouco seus mínimos detalhes. É preciso saber como configurá-la e como utilizá-la. Dessa forma, o desenvolvimento da aplicação fica muito mais rápida, já que, como citado por Matsumoto no item 3.1 desse documento, o foco no desenvolvimento de software devem ser as pessoas, as negociações, e não nos problemas que integrações de bibliotecas possam trazer ao código que está em desenvolvimento.

Como uma ótima *gem*, a *devise* funcionou muito bem, não gerando nenhuma problema adicional e funcionando exatamente como esperado. Dentre as funcionalidades que essa *gem* oferece, é interessante ressaltar as principais:

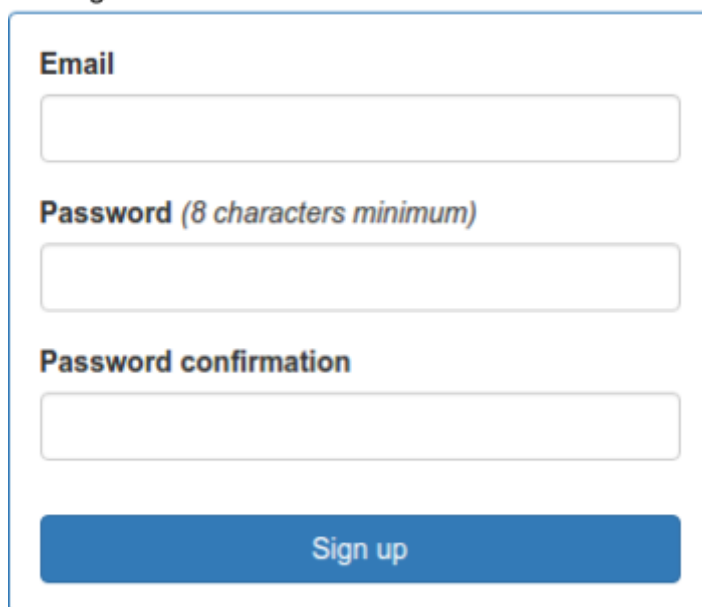
- Login de usuário (obviamente) - Por padrão, exigindo e-mail cadastrado e senha;
- Cadastro de usuário - Já cadastrando a senha do usuário encriptada no banco de dados;

---

<sup>9</sup> GitHub é um Serviço de Web Hosting Compartilhado para projetos que usam o controle de versionamento Git. É escrito em Ruby on Rails pelos desenvolvedores da Logical Awesome (Chris Wanstrath, PJ Hyett e Tom Preston - Wemder). O GitHub possui planos comerciais e gratuitos para projetos de código aberto (WIKIPEDIA, 2015c).

- “Esqueci minha senha” - Como é padrão em quase todos sistemas atualmente, essa gem também traz pronta a implementação da funcionalidade de redefinição de senha;
- Funções que informam se usuário está logado, para uma construção dinâmica das telas (Figuras 7 e 8).

Figura 7 - Formulário de cadastro de usuário.



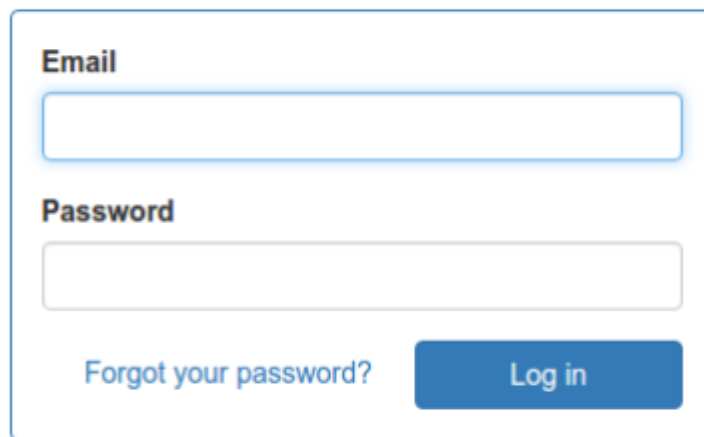
O formulário de cadastro de usuário é composto por três campos de entrada e um botão de ação. O primeiro campo é rotulado "Email" e é um retângulo branco com uma borda cinza. O segundo campo é rotulado "Password (8 characters minimum)" e também é um retângulo branco com uma borda cinza. O terceiro campo é rotulado "Password confirmation" e é um retângulo branco com uma borda cinza. Abaixo dos campos, há um botão azul com o texto "Sign up" em branco.

Fonte: Autor

Além das funcionalidades já disponibilizadas pela *gem*, um controle que diferencia administradores de usuários comuns foi criado. A implementação é muito simples, somente adicionando um campo a mais na tabela de usuários informando se o mesmo é um administrador ou não.

Os administradores, diferente dos usuários comuns, tem acesso à tela de gerenciamento de filmes, onde podem adicionar, editar e excluir filmes presentes no sistema.

Figura 8 - Formulário de login de usuário.

A imagem mostra um formulário de login de usuário. No topo, há o rótulo "Email" em negrito, seguido por um campo de entrada retangular. Abaixo dele, há o rótulo "Password" em negrito, seguido por outro campo de entrada retangular. Na base do formulário, há um link "Forgot your password?" em azul e um botão "Log in" em azul com o texto em branco.

Fonte: Autor

Os usuários comuns, por sua vez, podem criar comentários e, obviamente, esperar sugestões de filmes para ver.

#### 4.2 Gerenciamento de filmes

Tendo implementado o nível de acesso dos usuários, o passo seguinte desse projeto foi relativo ao gerenciamento de filmes, um dos elementos mais importantes no sistema descrito nesse documento. Para tal, foi necessário o estudo de uma modelagem de banco de dados que suportasse a ideia de *tags* quantitativas. A versão final de tal modelagem pode ser encontrada no apêndice B.

Para o cadastro de filmes, foi utilizada novamente outra *gem* chamada Paperclip. Tem a função de facilitar o cadastro de fotos no sistemas, já gerando os arquivos redimensionados de acordo com as configurações definidas em seu respectivo *model*. Esse gerenciamento de filmes inclui todas funcionalidades necessárias para um controle total da coleção desses dados.

No caso do formulário de cadastro, vale ressaltar como ficou o cadastro das *tags* de um filme, conforme ilustrado na Figura 9.

Figura 9 - Cadastro de *tags* de um filme.

Action	<input type="radio"/> ★ ★ ★ ★	Adventure	<input type="radio"/> ★ ★ ★ ★
Biography	<input type="radio"/> ★ ★ ★ ★	Comedy	<input type="radio"/> ★ ★ ★ ★
Documentary	<input type="radio"/> ★ ★ ★ ★	Drama	<input type="radio"/> ★ ★ ★ ★
History	<input type="radio"/> ★ ★ ★ ★	Horror	<input type="radio"/> ★ ★ ★ ★
Romance	<input type="radio"/> ★ ★ ★ ★	Sci-fi	<input type="radio"/> ★ ★ ★ ★
Thriller	<input type="radio"/> ★ ★ ★ ★	War	<input type="radio"/> ★ ★ ★ ★

Fonte: Autor

Como a Figura 9 mostra, o cadastro de *tags* de um filme no formulário de cadastro do próprio filme se torna algo muito intuitivo, onde o administrador escolhe a intensidade em estrelas de (0 a 3) que cada uma das *tags* aparece no filme, gerando uma característica relevante para o mesmo.

Além da tela de cadastro, o gerenciamento de filmes possui uma tela de listagem de todos os filmes, bem como botões de edição e remoção de cada um deles. Essa tela, obviamente, só é acessada pelos administradores do sistema (Figura 10).

Figura 10 - Gerenciamento de filmes.

## Movies

Id	Name	Duration	Imdb rating	Created at	Added by	Actions
1	The Matrix	136		Wed, 30 Dec 2015 16:44:30 +0000	lucasmorenosilva@gmail.com	<input type="button" value="Edit"/> <input type="button" value="Destroy"/>
2	The Butterfly Effect	114		Sun, 03 Jan 2016 04:27:08 +0000	lucasmorenosilva@gmail.com	<input type="button" value="Edit"/> <input type="button" value="Destroy"/>

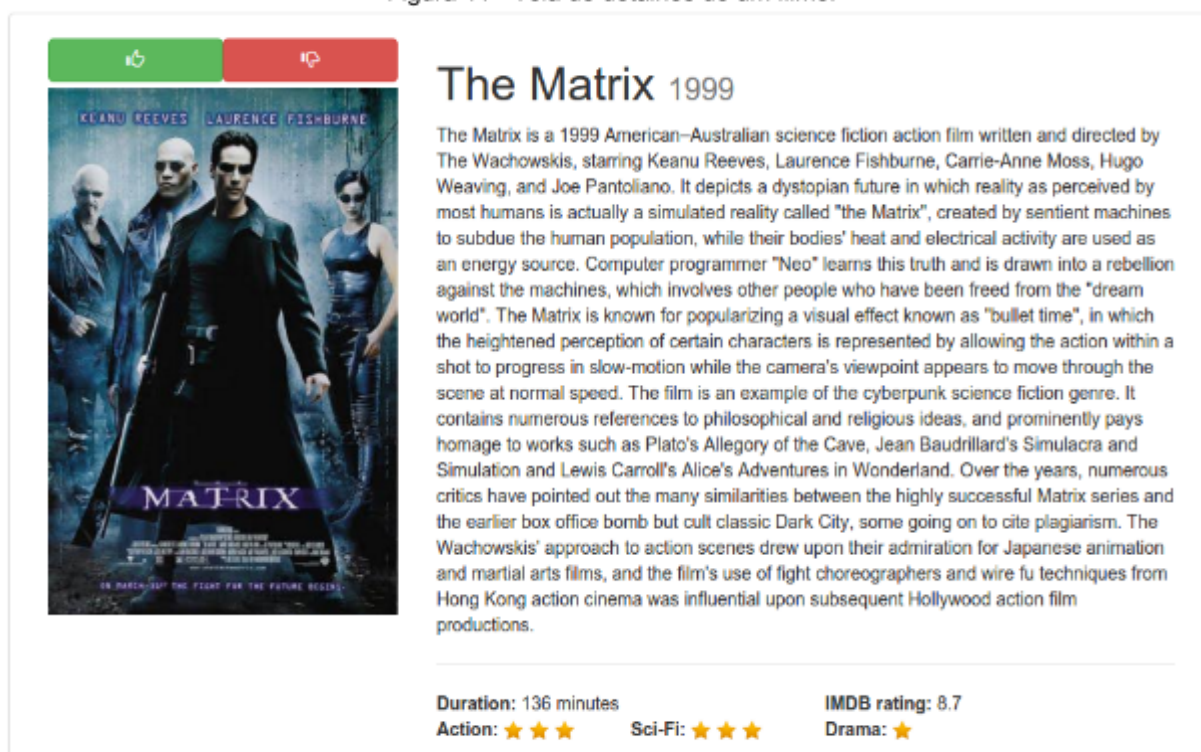
Fonte: Autor

### 4.3 Tela de detalhes de um filme

Com o cadastro de filmes funcionando, é necessário então exibir os

filmes cadastrados para os usuários que não são administradores, de forma que sejam exibidos dados relevantes do filme, utilizados em diversos outros sites, como o ano de lançamento, uma breve descrição e uma imagem. As *tags*, a princípio, só são mostradas quando o usuário está logado e é um administrador (Figura 11):

Figura 11 - Tela de detalhes de um filme.



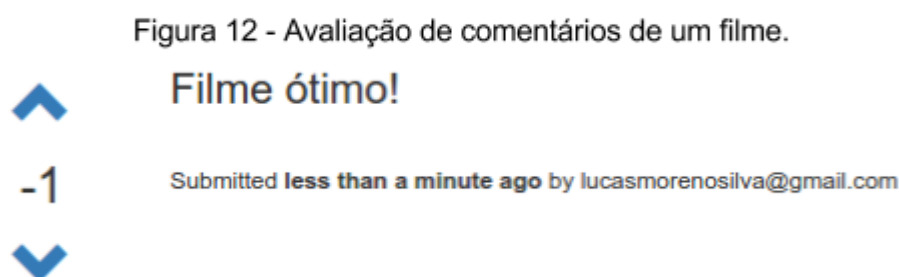
Fonte: Autor

Nessa tela, além das informações específicas sobre o filme, são exibidos os comentários de outros usuários ordenados por ordem de avaliação e também os botões de “gostei” e “não gostei”, que cria uma associação entre o filme e o usuário logado.

#### 4.4 Comentários

Mesmo não sendo um elemento que afete diretamente o sistema de recomendação de filmes, os comentários são um elemento muito importante nesse sistema visto que o usuário, ao chegar na tela do filme, consegue observar o que outros usuários acharam do filme de forma muito natural. Apesar de não entrar na lógica do algoritmo de sugestão, os comentários mais bem avaliados podem ser decisivos na hora de um usuário querer ou não assistir um filme.

Devido a isso, a implementação foi feita de forma que um usuário pode comentar mais de uma vez por filme, porém, só pode avaliar uma vez cada comentário, positivamente (seta para cima) ou negativamente (seta para baixo). Ao clicar na seta para cima, o “saldo” do comentário recebe +1. Ao clicar na seta para baixo, o saldo recebe -1. O comentário com maior saldo no filme aparece no topo (Figura 12).



Fonte: Autor

Para comentar sobre um filme, basta estar logado. O comentário é formado somente por um campo texto, sua data de criação e a associação com o usuário que o criou.

#### 4.5 Avaliação de filmes

Tendo as informações necessárias do filme, bem como os comentários de outros usuários, foi implementada a avaliação de filmes. O formato mais simples de avaliação foi utilizado: gostar ou não gostar. Os botões são exibidos acima da imagem do filme na tela de detalhes (Figura 11) e também sobre as miniaturas dos filmes exibidas na *home* (Figura 14), como será detalhado no tópico seguinte.

De acordo com a avaliação do usuário, uma relação será criada tanto entre o usuário logado e o filme, como o usuário logado e as *tags* daquele filme. A associação entre usuário e filme é muito simples. Considerando a tabela de usuários (*users*) e a tabela de filmes (*movies*), quando um usuário avalia positivamente ou negativamente um filme, uma entrada é adicionada à tabela *ratings*, que guarda o identificador do usuário logado, o identificador do filme e se a avaliação foi positiva ou negativa.

É necessária a criação de uma tabela entre usuários e filmes por se

tratar de uma relação  $M \times N$ , onde um usuário pode avaliar vários filmes e um filme pode ser avaliado por vários usuários. Tal relação é utilizada para identificar os filmes que cada usuário já avaliou, evitando que eles fiquem se repetindo desnecessariamente em sua tela de sugestão (*home*).

Além dessa associação entre usuários e filmes, outra associação que foi criada a partir da avaliação de um filme, diz respeito à relação entre usuários e *tags*. Na verdade, tais entidades são associadas de duas formas diferentes para que seja possível sugerir filmes de mais de uma forma.

A primeira relação entre essas entidades fica armazenada na tabela chamada *user\_tags*, onde são armazenadas a soma das intensidades das *tags* de cada filme que o usuário gostou agrupadas pelo seu gênero, bem como a subtração das *tags* de cada filme que o usuário não gostou, gerando um saldo que representa a intensidade que cada *tag* representa no gosto do usuário para aquele gênero. Por exemplo: caso o usuário goste de um filme do gênero ação, com intensidade 3 de ação e intensidade 2 de drama, sua relação com o gênero ação ficará representado exatamente pelas *tags* desse filme (ação: 3, drama: 2). Caso o mesmo usuário avalie positivamente outro filme de ação, com intensidade 2 de ação e intensidade 1 de ficção científica, sua relação com o gênero ação fica representado pela soma das *tags* desses dois filmes (ação: 4, drama: 2, ficção científica: 1).

Em caso de uma avaliação negativa também de um filme do gênero ação, as *tags* são subtraídas dessa relação. Em caso de avaliar negativamente um filme de ação com intensidade 3 de comédia e intensidade 1 de ação, sua relação com o gênero ação ficará com o resultado da subtração dessas *tags* com as demais (ação: 3, drama: 2, ficção científica: 1, comédia: -3).

Vale ressaltar que o número da soma em si não importa. A ordem com que as *tags* aparecem é o dado mais relevante, pois representa a preferência do usuário em relação aos filmes de ação. Caso um filme de outro gênero seja avaliado, não influenciará na soma dessas intensidades, pois elas são sempre associadas com o gênero principal do filme.

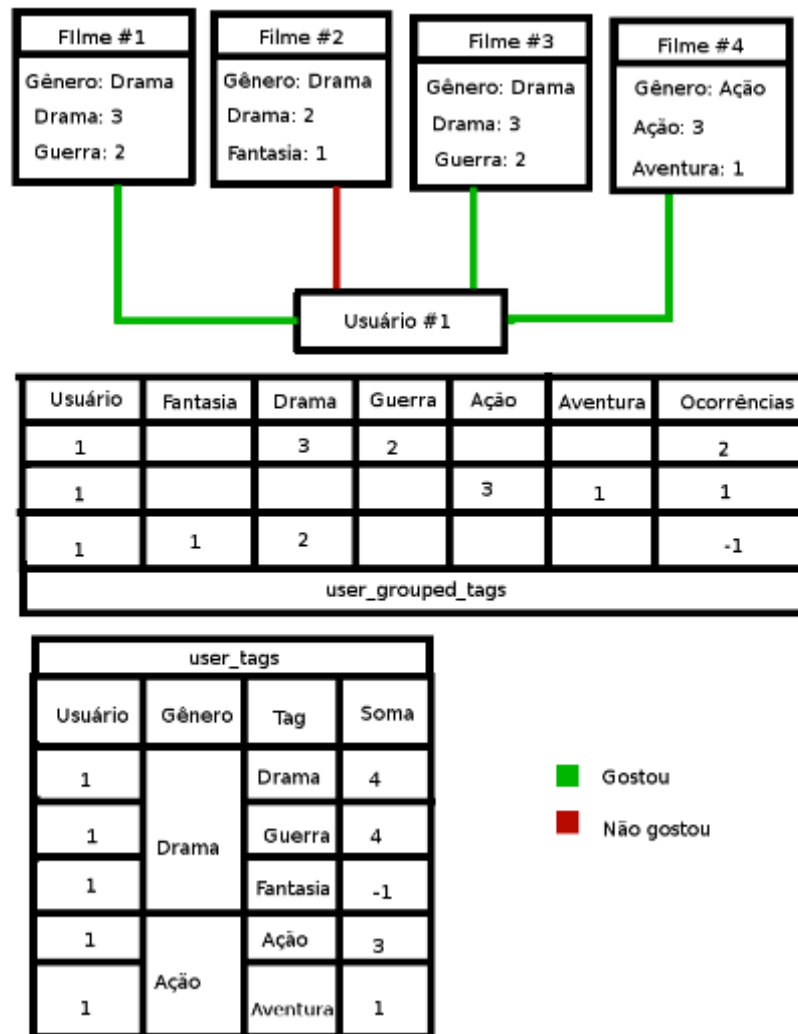
A outra associação criada entre usuários e *tags* fica na tabela chamada



*user\_grouped\_tags*, onde, ao invés de se armazenar a soma das intensidades das *tags* separadamente, armazena-se o número de vezes em que o usuário avaliou positivamente ou negativamente filmes com exatamente as mesmas *tags* do filme em questão. Supondo que um usuário avalie positivamente um filme com intensidade 1 de ficção científica e intensidade 2 de terror, uma entrada que representa essas *tags* **juntas** seria adicionada com o número de ocorrências igual a 1 nessa mesma tabela.

Caso o mesmo usuário volte a avaliar positivamente um filme com exatamente as mesmas *tags*, essa linha teria seu número de ocorrências incrementado em 1. No caso de uma avaliação negativa, teria seu número de ocorrências decrementado pela mesma quantidade. Vale ressaltar que essa segunda associação não é agrupada por gêneros principais. A Figura 13 apresenta como as relações se comportariam no exemplo retratado.

Figura 13 - Exemplo da associação entre usuários e tags



Fonte: Autor

#### 4.6 Sugestão de filmes

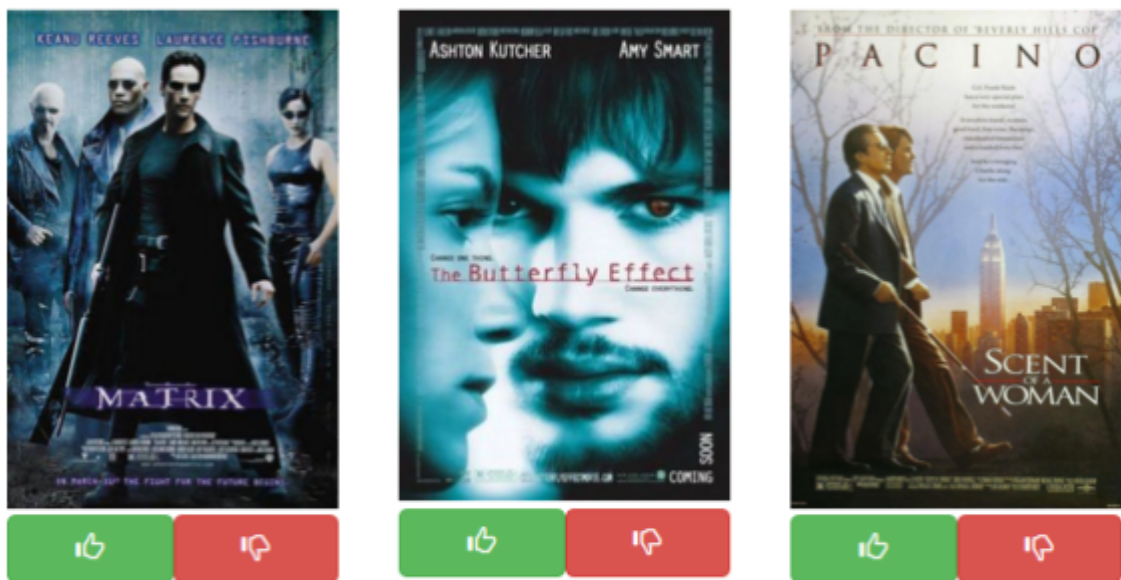
Tendo o banco modelado e a aplicação funcional, a implementação da sugestão de filmes teve início. Para tal, foram desenvolvidas quatro formas de sugerir filmes; sendo duas analisando perfis de usuários parecidos e outras duas procurando filmes com características parecidas com o gosto do usuário. Todas as formas de sugestão se baseiam nas duas tabelas que relacionam usuários e tags (*user\_tags* e *user\_grouped\_tags*).

Mesmo utilizando algoritmos de sugestão colaborativos e baseados em conteúdo, não pode-se considerar esse algoritmo híbrido, pois o resultado de um

algoritmo não influencia no resultado do outro.

Pelo menos a princípio, todos esses processos sugerem filmes aos usuários para que, com o tempo de uso, numa base com uma boa quantidade de filmes e usuários, seja possível determinar qual deles é o mais preciso. Todas as sugestões de filmes são mostradas na *home*, onde uma lista de filmes é mostrada e, quando o usuário avalia o filme, é salvo no banco qual a origem dessa sugestão (Figura 14).

Figura 14 - Lista de filmes na *home*.



Fonte: Autor

#### 4.6.1 USUÁRIOS COM USER\_GROUPED\_TAGS PARECIDAS

A primeira forma de sugestão se baseia na tabela *user\_grouped\_tags*, que armazena a ocorrência de avaliações positivas que um usuário atribuiu para filmes com *tags* exatamente iguais.

Por haver essa necessidade de serem *tags* exatamente iguais, achar usuários que tenham os cinco grupos de *tags* com mais ocorrências iguais é algo não muito comum, pois a probabilidade disso ocorrer é baixa. Porém, caso encontre, é bem provável que os usuários tenham gostos parecidos. Dessa forma, basta sugerir filmes que um gostou ao outro.

#### 4.6.2 *USUÁRIOS COM USER\_TAGS PARECIDAS*

A segunda forma de sugestão se baseia também na associação entre perfis de usuários, porém, essa associação usa como métrica os valores encontrados na tabela *user\_tags*. Da mesma forma, procura-se no banco um usuário com as cinco *tags* de maior intensidade parecidas com outro usuário para um determinado gênero. A grande diferença entre essa forma e a anterior consiste na dependência de se especificar um gênero para receber a sugestão.

#### 4.6.3 *FILMES COM TAGS DA TABELA USER\_GROUPED\_TAGS*

A terceira forma de sugestão não se baseia em usuários parecidos, mas em filmes com as características das cinco *user\_grouped\_tags* com maior número de ocorrências.

#### 4.6.4 *FILMES COM TAGS DA TABELA USER\_TAGS*

A quarta e última forma também busca diretamente por filmes, porém usa a tabela de *user\_tags* em suas buscas, buscando filmes com *tags* que apareçam nas cinco *tags* com maior intensidade do usuário em questão.

## 5 CONSIDERAÇÕES FINAIS

Com a implementação do sistema de sugestão de filmes apresentado nesse documento, acredita-se que o desenvolvimento de sistemas desse tipo é algo que tende a crescer junto com a quantidade de dados nos dias de hoje. A população tem acesso a tantas informações atualmente que realmente fica difícil selecionar algo relevante para ler, ouvir, assistir.

Também por causa da quantidade de informações, cruzar dados diretamente no banco de dados está se tornando um problema por se tratar de algo muito custoso. Tendo isso em vista, a associação por perfis de usuários se mostra uma técnica que, além de mais performática, muitas vezes acaba sendo mais precisa também. Mesmo quantificando os filmes das mais diferentes maneiras, nunca poderemos especificar exatamente todos elementos de um filme num banco de dados.

A ideia das *tags* quantitativas foi sendo adaptada de acordo com as necessidades encontradas no sistema, e realmente serviu muito bem para definir as intensidades das *tags* tanto para filmes como para usuários. Quanto aos resultados, o sistema visa avaliar cada uma das quatro formas de sugestão descritas de acordo com porcentagem de precisão de cada uma (baseado em avaliações positivas). Porém, para obter resultados relevantes, é necessária uma considerável quantidade de usuários utilizando o sistema por um certo tempo.

Apesar dos métodos de sugestão de filmes mostradas nesse documento não se basearem em um algoritmo de inteligência artificial, as informações levantadas com o uso constante do sistema pelos mais diferentes tipos de usuários irá gerar um banco muito rico que pode ser utilizado como entrada em um algoritmo de aprendizagem no futuro.

## REFERÊNCIAS BIBLIOGRÁFICAS

AKITA, Fabio. **Como não fazer pesquisas usando LIKE**. Disponível em: <<http://www.eventials.com/akitaonrails/como-nao-fazer-pesquisas-usando-like/>>.

Acesso em 26 mai. 2015.

ASSIS, Pablo de (2009). **O que é tag?** Disponível em: <<http://www.tecmundo.com.br/navegador/2051-o-que-e-tag-.htm>>. Acesso em 23 mai. 2015

CCM. **O protocolo HTTP**. 2016. <<http://br.ccm.net/contents/266-o-protocolo-http>>. Acesso em 10 jan. 2016.

CANALTECH. **O que é open source?** Disponível em <<http://canaltech.com.br/o-que-e/o-que-e/O-que-e-open-source/>>. Acesso em 22 dez. 2015.

CORUMBA, Daniela M. **Sistema de Recomendação baseado na similaridade de perfis do Twitter**. Disponível em <<http://www.machinelearning.org/proceedings/icml2007/papers/407.pdf>>. Acesso em 28 mai. 2015

DAVID, Marcio F. **Programação Orientada a Objetos: uma introdução**. 2007. Disponível em: <<http://www.hardware.com.br/artigos/programacao-orientada-objetos/>>. Acesso em 27 mai. 2015

EIS, Diego. **O básico: O que é HTML?**. 2011. Disponível em <<http://tableless.com.br/o-que-html-basico/>>. Acesso em 10 jan. 2016

GASPAROTTO, Henrique M. **Os 4 pilares da Programação Orientada a Objetos**.

2015. Disponível em:  
<<http://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>>.  
Acesso em 27 mai. 2015

HONG, K. **Model View Controller (MVC)**. 2015. Disponível em  
<[http://www.bogotobogo.com/RubyOnRails/RubyOnRails\\_Model\\_View\\_Controller\\_MVC.php](http://www.bogotobogo.com/RubyOnRails/RubyOnRails_Model_View_Controller_MVC.php)>. Acesso em 19 ago. 2015.

RUBY, Sam. **Agile Web Development with Rails 4**. The Pragmatic Bookshelf. 2013.

TURINO, C. **Uma gestão cultural transformadora**: Proposta para uma Política Pública de Cultura. 2006.

WIKIPEDIA. **Framework**. 2015a. Disponível em  
<<https://pt.wikipedia.org/wiki/Framework>>. Acesso em 22 dez. 2015

WIKIPEDIA. **GitHub**. 2015b. Disponível em <<https://pt.wikipedia.org/wiki/GitHub>>.  
Acesso em 22 dez. 2015.

WIKIPEDIA. **Ruby (programming language)**. 2015c. Disponível em  
<[https://en.wikipedia.org/wiki/Ruby\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Ruby_(programming_language))>. Acesso em 21 dez. 2015.

ZAMBARD, P. **‘Internet das Coisas’: entenda o conceito e o que muda com a tecnologia**. 2014. Disponível em  
<<http://www.techtudo.com.br/noticias/noticia/2014/08/internet-das-coisas-entenda-o-conceito-e-o-que-muda-com-tecnologia.html>>. Acesso em 09 jan. 2016.

## APÊNDICES

### Apêndice A - Entradas mais utilizadas no cabeçalho de uma requisição HTTP

Entrada	Descrição
Accept	Tipo de conteúdo aceitados pelo motor de pesquisa
Accept-Charset	Jogo de caracteres esperado pelo motor de pesquisa
Accept-Encoding	Codificação de dados aceite pelo motor de pesquisa
Accept-Language	Linguagem esperada pelo motor de pesquisa
Authorization	Identificação do motor de pesquisa junto do servidor
Content-Encoding	Tipo de codificação do corpo do pedido
Content-Language	Tipo de linguagem do corpo do pedido
Content-Length	Comprimento do corpo do pedido
Content-Type	Tipo de conteúdo do corpo do pedido
Date	Data de início de transferência dos dados
Forwarded	Utilizado pelas máquinas intermédias entre o motor de pesquisa e o servidor
From	Permite especificar o e-mail do cliente
Orig-URL	URL de origem do pedido
Referer	URL da ligação a partir da qual o pedido foi efetuado
User-Agent	Informações sobre cliente - nome e versão do <i>browser</i>



## Apêndice B - Modelagem final do banco de dados

