MAC0416/5855
Tópicos Especiais em Desenvolvimento para Web

Relatório Final

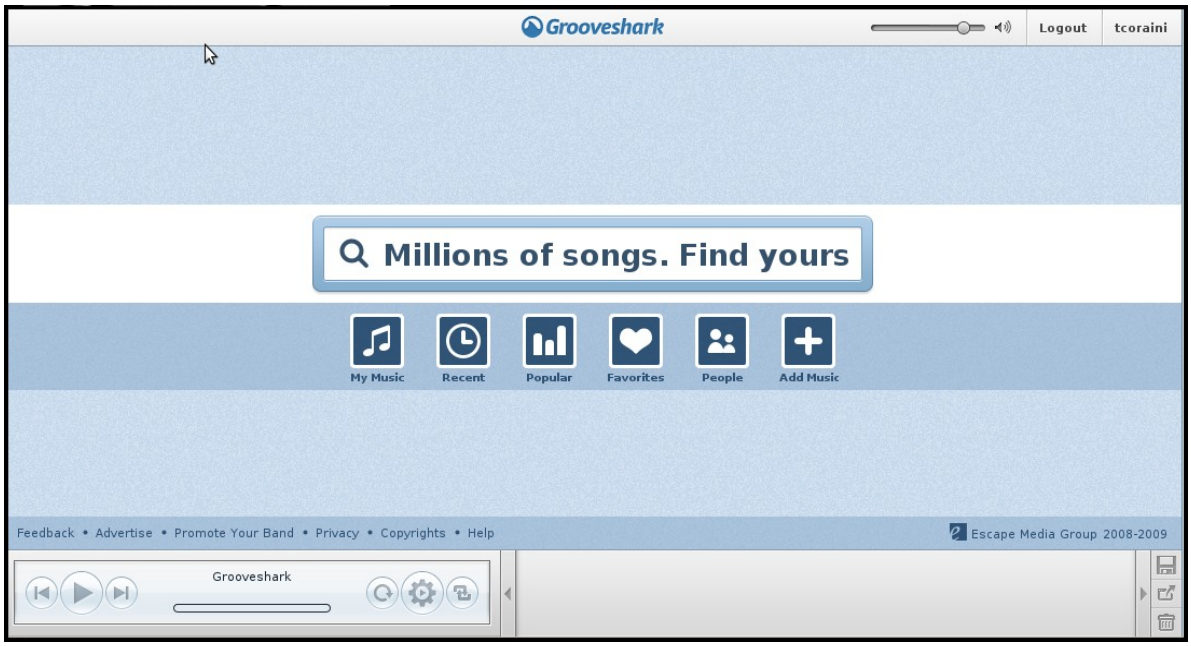
Repositório de músicas *online*

Roberto Piassi Passos Bodo
Thiago Henrique Coraini

1. Análise de um website

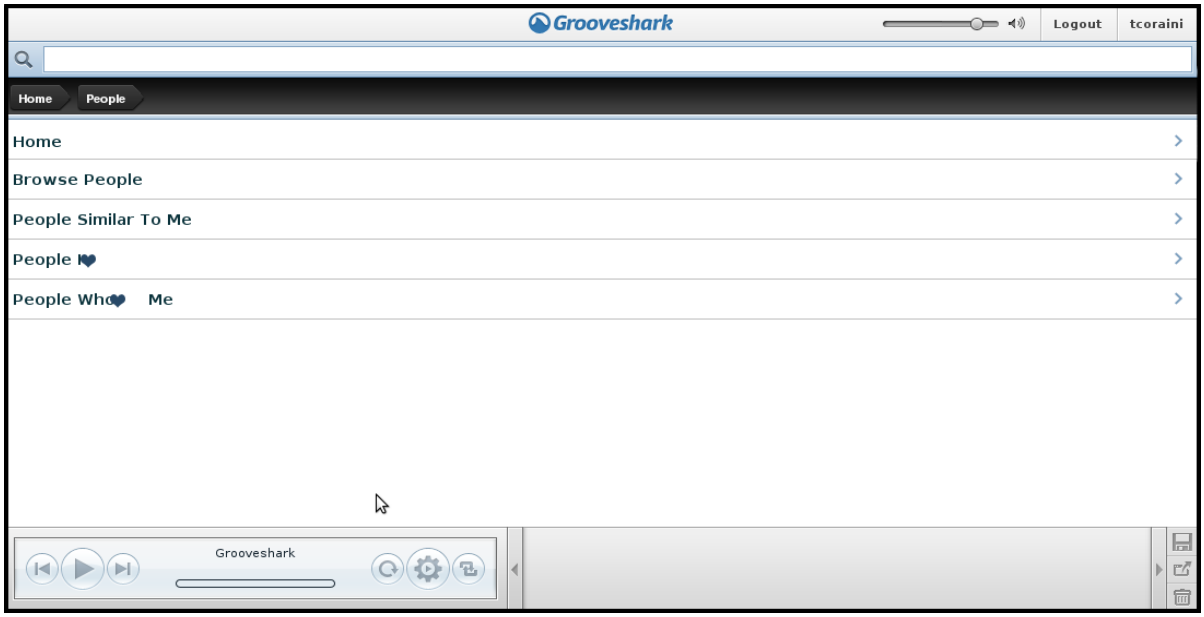
Para este relatório, foi pedido que analisássessmos um website que utilizasse tecnologias características da Web 2.0. Ao invés de escolher algum site da lista oferecida, resolvemos escolher um não inicialmente listado: *Grooveshark* — http://listen.grooveshark.com. O principal motivo por termos escolhido esse site é que a idéia dele é basicamente a mesma da aplicação desenvolvida por nosso grupo durante o semestre: é um repositório de músicas, onde os próprios usuários enviam as músicas e estas ficam disponíveis para todos os outros. Um dos fortes do site é, como poderia se esperar, o sistema de recomendação. Basicamente, o *Grooveshark* é tudo o que pensamos para nossa aplicação, porém é claro que implementado de maneira completa e estável, diferente de nossa aplicação que tinha interesse apenas de estudos.

O *Grooveshark* é feito totalmente em Flash, usando o *framework* Adobe Flex. Isso dá ao site um visual realmente impressionante, com menus deslizantes e uma interface muito agradável. Essa é a sua tela inicial:

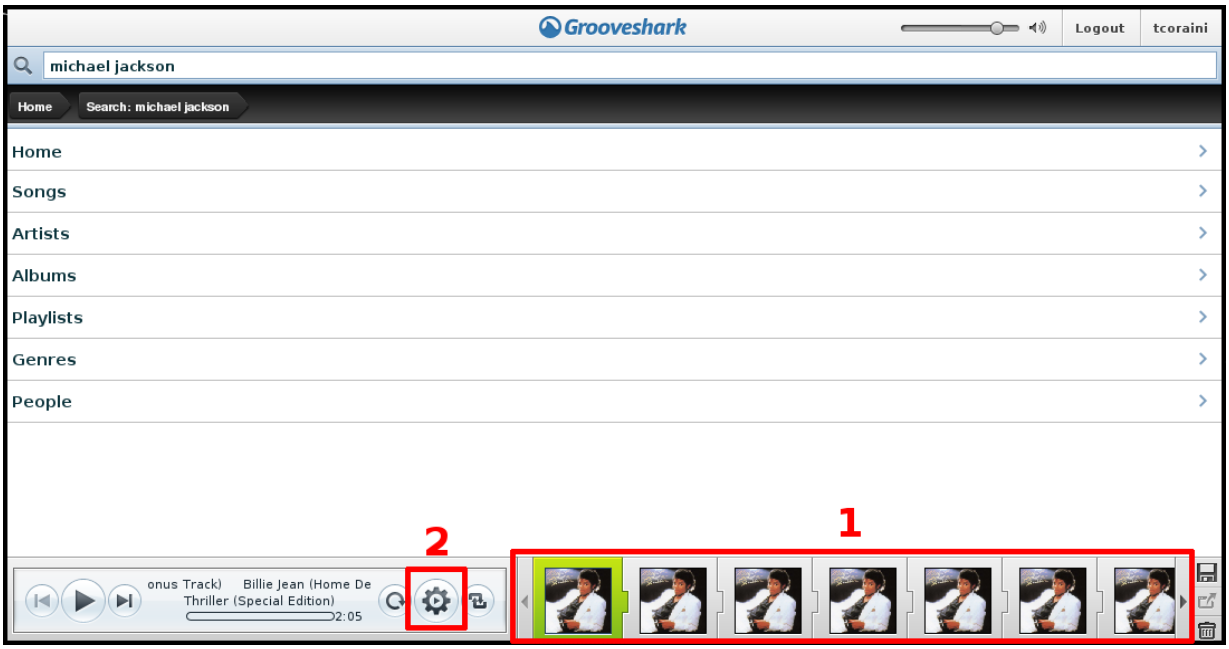


Como se pode observar, ele apresenta algumas opções bem interessantes. No menu *My music*, podemos ver por exemplo as *Playlists* criadas ou as músicas enviadas por nós ao sistema. Em *Recent*, podemos ver as últimas músicas que vimos ou reproduzimos. Em *Popular*, serão mostradas as músicas mais tocadas do site. Em *Add Music*, um *applet* será carregado permitindo baixar facilmente várias músicas do computador. O *Grooveshark* possui uma opção muito interessante, de favoritos. Podemos favoritar músicas e usuários, clicando num ícone de coração. No menu *Favorites*, podemos ver tudo que foi definido como favorito. Já no menu *People*, podemos procurar por usuários, ver aqueles que colocamos como favoritos e aqueles que nos colocaram em seus favoritos.

A figura abaixo mostra como é a tela de visualização de outros usuários do *Grooveshark*. Como podemos ver, com as opções de pessoas que nos colocaram nos favoritos e pessoas colocados por nós nos favoritos, temos um sistema que lembra o do *Twitter*: podemos "seguir" pessoas e podemos ter pessoas "nos seguindo", facilitando muito a interação entre usuários e o acompanhamento das novidades de cada um. Além disso, não podemos deixar de notar a opção "People Similar To Me", ou seja, além de músicas, o site recomenda também outros usuários que ele crê terem gosto parecidos com o seu.

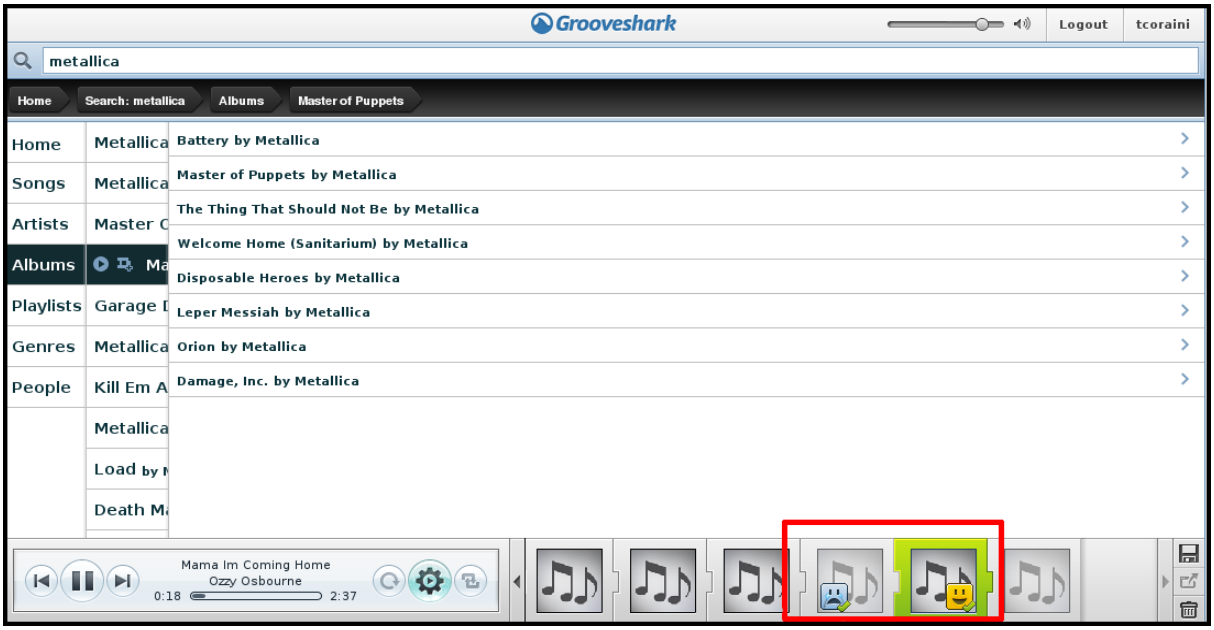


Após encontrar uma música que desejamos ouvir, devemos enviá-la para a fila (*queue*) do *Grooveshark*, para que seja tocada. Usando a estratégia de fila, fica muito fácil ir adicionando novas músicas que desejamos que toquem na continuação. Além disso, a qualquer momento podemos salvar o estado atual de uma fila para uma *playlist*, que poderá ser ouvida novamente no futuro. A imagem abaixo mostra, identificado por 1, uma fila do *Grooveshark* com diversas músicas que estão sendo tocadas:



Cada música que está na fila pode ser adicionada aos favoritos (clicando-se num símbolo de coração que aparece ao passar-se o *mouse* sobre ela). Além disso, para cada música podemos abrir uma janela de informações sobre ela, que irá conter músicas consideradas semelhantes àquela.

Já identificado com o número 2 na figura acima temos o *Autoplay*, ferramenta do *Grooveshark* que realiza as recomendações de músicas. Ao clicar naquele botão, o modo de *Autoplay* é ativado e então, uma música é adicionada no fim da fila. Após ser tocada, outra música será adicionada e assim sucessivamente, sempre com uma sugestão nova. No modo de *Autoplay*, temos mais duas novas opções: dizer se gostamos ou não de uma música, como mostra a figura abaixo:



Como podemos ver, nas duas músicas em destaque, uma delas possui um rostinho triste, indicando que não gostamos daquela sugestão. Já a outra foi sinalizada de forma a mostrar que aprovamos a recomendação. Isso fará com que o algoritmo de recomendação do *Grooveshark* possa se especializar cada vez mais para nosso gosto.

Certamente haveria mais para ser falado sobre esse site, mas convidamos todos a visitá-lo e conferir por si mesmo tudo que ele oferece. Aqui, cremos ter mostrado alguns de seus principais diferencias, e certamente vimos que muito do que foi aprendido na disciplina é utilizado na prática pelos criadores do site.

2. O sistema desenvolvido

2.1 Especificação

Nosso grupo escolheu como tema para o projeto a ser desenvolvido um repositório de músicas online. O que decidimos fazer era um site análogo ao *Youtube*, porém que armazenasse músicas ao invés de vídeos. Os usuários deveriam ser capazes de, após registrarem-se e estarem devidamente logados, enviarem músicas para o sistema, que as armazenaria com suas informações (como título, artista, etc). Após isso, as músicas estariam disponíveis para serem buscadas e ouvidas por qualquer usuário do sistema, e não apenas àquele que a submeteu. Ao ouvir uma música, a idéia é que o usuário fosse exposto a sugestões de outras músicas similares também disponíveis no sistema. Não sabíamos a princípio como fazer isso, mas sabíamos que seria um dos objetos de estudo da disciplina. Com o andar do curso, novas funcionalidades, não previstas a princípio, também foram adicionadas, como a exibição para o usuário da letra da música sendo tocada, e a possibilidade de usuários enviarem biografias de artistas para o site, que, se aprovadas, seriam exibidas junto ao perfil respectivo.

Para ilustrar melhor as funcionalidades do sistema desenvolvido, apresentamos a seguir os casos de uso tanto para um usuário comum como para um usuário administrador do sistema.

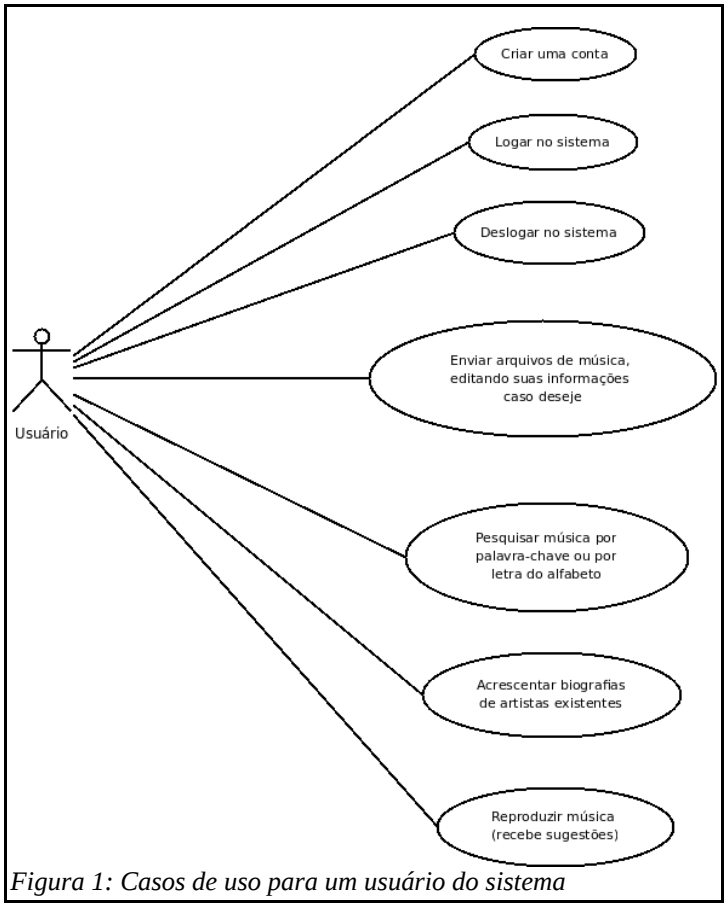
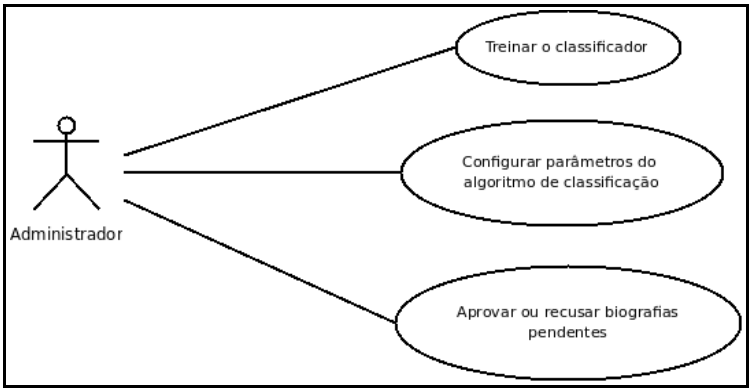


Figura 1: Casos de uso para um usuário do sistema



2.2 Arquitetura

Nosso sistema foi desenvolvido seguindo o mais de perto possível o padrão MVC - *Model View Controller*. No caso da *View*, ou seja, da interface com o usuário, estão incluídas todas as páginas JSP (que, em alguns casos, são páginas JSF). Inclui também a formatação dessas páginas, realizada com a ajuda de CSS. Não existe nenhuma grande novidade quanto a essa camada: tentamos diminuir o máximo possível a quantidade de código embutido nas páginas, e o máximo que usamos

foram *taglibs*, mesmo assim na menor quantidade possível. A idéia é que essa camada cuide apenas da apresentação, usando código apenas para obter os dados das outras camadas.

Na camada *Model*, ou seja, do modelo, foram implementadas as entidades que representam tabelas no banco de dados. No caso do nosso sistema, as seguintes entidades foram implementadas:

- **User:** representa um usuário do sistema.
- **Music:** representa uma música que existe no repositório.
- **Artist:** representa um artista que possui alguma música no repositório.
- **Tag:** representa uma tag que já foi aplicada, por um usuário, para alguma música no sistema.
- **PendingBio:** representa uma biografia que ainda está no estado pendente, ou seja, foi enviada mas precisa ser aprovada ou não por um administrador.
- **Feature:** utilizada pelo algoritmo de filtragem de documentos, representa uma característica (em geral palavras) extraída de um documento.
- **Category:** também utilizada pelo algoritmo de filtragem, representa uma das possíveis categorias nas quais um documento pode ser classificado (no caso do sistema, apenas biografias ou não-biografias).

Cada uma dessas entidades possuía um DAO (*Data Access Object*) respectivo, contendo os métodos comumente utilizados na manipulação de bancos de dados, e portanto que não serão descritos aqui. Todos os DAO's utilizavam Hibernate para manipular o banco de dados.

Já na camada *Controller*, que é a camada responsável por responder as manipulações do usuário na camada *View* e fazer as ações necessárias sobre a camada *Model*, foi implementada toda a lógica de negócios do sistema. Numa primeira fase, essa lógica era executada basicamente por *Servlets*, implementadas uma para cada ação possível. Posteriormente, com a inserção de JSF ao sistema, algumas ações da camada de controle apareciam em *Backing Beans*, que eram ligados a páginas JSF. Como nem todo o sistema foi portado para JSF, algumas *Servlets* continuaram existindo. As ações principais incluem: *login*, registro, *upload* de músicas, reprodução de músicas, busca de músicas, entre muitas outras.

Além dessas três camadas, aparece ainda no código algumas classes responsáveis pelos testes unitários do sistema, utilizando o *framework Junit*.

2.3 Tecnologias utilizadas

Diversas tecnologias foram estudadas no decorrer do semestre. Algumas foram mantidas no sistema, outras acabaram sendo excluídas em fases posteriores. Abaixo, explicamos a utilização de cada uma e justificamos se ela foi mantida na aplicação ou não.

Ajax, Comet e JSON

Ajax não é bem uma tecnologia, mas uma maneira de se combinar tecnologias já existentes para conseguir resultados realmente impressionantes em páginas *web*. Em nossa aplicação, foi utilizado em alguns lugares, como, por exemplo, na página de busca. Quando o usuário escolhe uma letra no alfabeto, o conteúdo da página é modificado dinamicamente e de modo assíncrono, mostrando a lista de artistas com essa letra. Quando um artista é escolhido, é mostrada uma lista com os álbuns desse artista. E assim por diante até ele encontrar a música desejada.

JQuery

JQuery facilita muito o uso de Ajax e é uma ótima ferramenta para modificar aspectos estruturais e visuais das páginas, evitando que tenhamos que mexer diretamente com Javascript e em particular tratar as incompatibilidades entre diferentes navegadores. Procuramos usar sempre que possível essa biblioteca para facilitar nosso trabalho, e ela de fato ajudou muito.

JSF

A tecnologia *JavaServer Faces* facilita bastante o desenvolvimento de aplicações web, oferecendo facilidades para a criação de interfaces com o usuário através de componentes (que podem ser reutilizáveis), e a definição de classes Java que irão interagir de forma transparente com esses componentes. Possui ainda meios de realizar, sem grandes dificuldades, tarefas muito importantes como a validação de formulários, a conversão dos valores de diferentes campos, entre outros. Porém, devido a ter sido estudada somente no decorrer da implementação de nossa aplicação, e pela falta de tempo principalmente, nem todo o sistema foi adaptado para funcionar utilizando JSF. Apenas algumas páginas, na versão final, utilizam a tecnologia. Isso, porém, não nos impediu de aprender o básico sobre a tecnologia e ver de toda sua potencialidade. Com mais tempo, sem dúvida uma das primeiras tarefas seria portar toda aplicação para que passe a usar JSF.

Struts

A tecnologia *Struts* foi implementada, como solicitado, após a apresentação do grupo 4 da fase 2. Porém, após essa implementação, acabamos resolvendo por não mantê-la na aplicação. Isso se deve ao fato de, em nossa opinião, uma aplicação que utilize *JSF* e *Struts* não faz muito sentido, já que as duas tecnologias, apesar de não terem abordagens totalmente iguais, propõe-se a resolver o mesmo problema geral, de controlar o fluxo de uma aplicação web seguindo o padrão MVC. Em particular, preferimos manter o uso de *JSF* em detrimento de *Struts* principalmente pela primeira ser uma tecnologia mais nova, e que já teve sua especificação incorporada ao padrão Java EE. Dessa forma, cremos que, no estado atual das tecnologias para web, é mais interessante o estudo e a utilização de *JSF* (ainda que muitas empresas continuem usando *Struts*, é verdade).

Hibernate e JPA

Toda aplicação acessa o banco de dados utilizando o Hibernate. As tabelas e as colunas foram criadas utilizando *annotations* de JPA. A migração de JDBC, da fase inicial, para o Hibernate foi feita com certa tranquilidade. Como nossa aplicação seguia o padrão MVC, só tivemos que modificar as classes DAO's, basicamente. Uma dificuldade que enfretamos foi na hora de utilizar a annotation *@ManyToMany*. Quando adicionamos as tags em nosso sistema, definimos que toda música poderia "possuir" qualquer tag. E que toda tag poderia "possuir" qualquer música. Quando temos uma relação desse tipo, o *Hibernate* cria automaticamente uma tabela intermediária e insere nela (automaticamente também). Para isso acontecer precisávamos ter uma das classes "comandando" a relação e a inserção deveria ser feita em uma única *Session*, o que demandou algum tempo até que todos os ajustes necessários fossem descobertos e realizados. Porém sem dúvida a ajuda oferecida por esse *framework* é imensa, facilitando muito o trabalho com bancos de dados.

Spring

O framework *Spring* mostrou-se bastante interesse e muito abrangente. Seria, possivelmente, de grande valia para nosso sistema. Porém, tivemos uma certa dificuldade no momento de seu estudo, chegando a conclusão que sua curva de aprendizado não seja das mais suaves. Isso pode também ser consequência da falta de tempo durante o momento em que estavámos nos dedicando a sua incorporação ao projeto. No final, não tivemos sucesso implementando nada de programação orientada a aspectos, um tema que parece ser muito interessante e o qual desejavámos muito que

fizesse parte de nossa aplicação. No fim, acabamos ficando no "feijão com arroz" de *Spring* e apenas usando-no para realizar algumas injeções de dependência em pontos específicos da aplicação, dizendo respeito em particular a objetos do *Hibernate*. Dessa forma, resolvemos para a versão final não manter a utilização de *Spring*, já que apenas a injeção de dependências era usada e, ainda que essa facilidade seja muito bem-vinda em projetos em geral, no caso de nossa aplicação, pouco extensa, ela não apresentava um grande ganho, seja na facilidade de entendimento do código, seja na eficiência da aplicação. Para uma melhoria posterior, tentaríamos novamente a utilização de algo referente a aspectos para enriquecer de fato nosso projeto.

JUnit e Selenium

Essas tecnologias foram mantidas no projeto, ainda que representem um acréscimo "invísivel" aos usuários. Porém, a escrita de testes unitários já foi comprovada como uma prática essencial para o desenvolvimento de qualquer sistema que se preze. No nosso sistema, os testes realizados não abrangem uma porcentagem tão grande assim de todo o código, é bem verdade, mas já são um começo para ter um sistema quase todo coberto por testes, e portanto muito mais estável e seguro no que diz respeito a modificações e acréscimos de novas funcionalidades.

Webservices e SOA

O serviço implementado em nosso sistema faz requisições SOAP para um site de letras de músicas. Ele mostra, quando requisitado pelo usuário, a letra da música tocada no momento. No caso, resolvemos não utilizar o comando *'wsimport'*, apresentado em sala de aula. Após estudar a estrutura básica de uma mensagem SOAP e procurar alguns exemplos em tutoriais, nós mesmos criamos as requisições SOAP, a partir do arquivo .wsdl fornecido pelo site. Reconhecemos que essa maneira, não automatizada, de gerar requisições pode apresentar problemas no futuro, mas podemos afirmar que aprendemos muita coisa, criando as mensagens SOAP "na mão".

Estilo arquitetural REST e Mashups

Esse tema foi o apresentado pela nossa dupla, e por isso não implementamos o mesmo em nosso projeto. Todavia, vale lembrar que, para a demonstração de utilização de *Web Services*, o que foi de fato implementado foi um *mashup*, utilizando um site que disponibilizava letras de músicas.

Ruby on Rails

O sistema criado em Rails não está inserido na versão final, por motivos óbvios. Para não deixarmos a aplicação em Rails completamente afastada da aplicação em Java, fizemos com que as duas "olhassem" para o mesmo banco de dados. Seguindo um tutorial apresentado no site oficial do Ruby on Rails, criamos toda a aplicação em Rails quase que de forma automática e, utilizando o comando *"scaffold"* conseguimos modelar as entidades de forma idêntica. Adaptando alguns arquivos de configuração do Rails e o nome das tabelas na aplicação Java, conseguimos com que toda modificação feita no bando de dados fosse vista por ambas as aplicações.

2.4 Algoritmos de Inteligência Coletiva

Na terceira fase da disciplina foram apresentados diversos algoritmos sobre inteligência coletiva. Foi solicitado que escolhêssemos três deles para serem incorporados ao nosso sistema. Devido a falta de tempo, característica de finais de semestre, conseguimos implementar satisfatoriamente apenas dois algoritmos. Descrevemos abaixo seu uso.

Classificação das Músicas com *Tags*

Para versão final, adicionamos ao nosso projeto um sistema de classificação das músicas com *tags*. No momento do upload de um arquivo, o usuário pode colocar até 10 *tags*, na música enviada. A partir dessas *tags*, podemos recomendar novas músicas aos usuários. Na página do *player*, temos uma lista com cinco músicas, por *tag*, relacionadas à que está sendo tocada no momento. Essa recomendação é feita baseada nas músicas que apresentam as mesmas *tags*, simplesmente.

No caso, estamos seguindo o modelo do Youtube, em que somente o "dono" do arquivo pode colocar *tags* nas músicas. Uma melhoria para o sistema seria permitir que todos os usuários classificassem as músicas e, com isso, criar um sistema de recomendação que calcule a correlação entre as músicas. Isso não foi implementado na versão final por enfrentarmos problemas de tempo e problemas com o Hibernate. Conforme foi explicado no item de Hibernate acima, a tabela que lida com a relação das músicas com as *tags* é gerada pelo Hibernate. Além de perdermos um tempo razoável para conseguir que tudo fosse inserido de forma correta, não encontramos um jeito de adicionar uma coluna extra nesta tabela. Com essa coluna poderíamos guardar quantas vezes uma mesma tag foi associada à uma música e, com esse número, poderíamos calcular a correlação entre músicas, conforme visto na apresentação em aula.

Classificação de Documentos

Os algoritmos de filtragem de documentos foram estudados no capítulo 6 do livro do Seragan, e foram um dos implementados em nosso projeto, no que diz respeito à inteligência coletiva. Basicamente, algoritmos desse tipo recebem um documento e são capazes de filtrá-lo automaticamente e definir a qual categoria, dentre um conjunto pré-determinado, ele pertence. Esse processo deve vir depois de uma fase de treinamento, no qual documentos e suas respectivas categorias são fornecidas ao algoritmo para que ele "aprenda" o que caracteriza uma determinada classe de documentos e como diferenciá-los.

Em nossa aplicação, o que implementamos foi a possibilidade de usuários postarem biografias de seus artistas favoritos. Para isso, bastava abrir a página de um artista e, caso ainda não houvesse uma biografia para ele, haveria um link permitindo o envio de um texto. Porém, é claro que não poderíamos simplesmente exibir qualquer texto enviado por usuários. O que desejaríamos era justamente filtrar o que eram de fato biografias do que não eram dentre os textos enviados. O algoritmo acima foi então utilizado, especificamente o classificador de *Fisher*. Foi criada uma página para que um administrador do sistema efetuasse o treinamento do algoritmo, o que deveria ocorrer numa fase anterior a sua colocação em uso, com uma carga minimamente razoável de textos (tanto biografias válidas quanto textos indesejáveis). Após o treinamento, o algoritmo iria decidir automaticamente se textos enviados eram ou não biografias. Ele então decidiria se iria exibir tal texto na página do artista ou não. Porém, todos os textos enviados ficavam ainda numa lista de "biografias pendentes", à espera de um administrador verificar manualmente a validade de tal texto e decidir por aceitá-lo em definitivo ou não.

O resultado final é difícil de ser analisado: tudo funciona conforme o esperado e a idéia nos agrada, porém saber se o algoritmo de fato atingiria um bom nível de acerto dependeria de treiná-lo com uma carga muito grande de textos. Isso implicaria tanto um treinamento inicial extenso quanto uma posterior grande quantidade de biografias sendo enviadas e avaliadas por um administrador (nessa fase o *feedback* do administrador realimenta o algoritmo e também serve como treinamento). Uma

forma de melhorar ainda mais a interação do site com os usuários seria permitir que eles mesmos avaliassem uma biografia como válida ou não. Porém, isso realmente não foi implementado e ficaria para uma próxima etapa.

3. Conclusão

A idéia da disciplina nos agrada bastante, em particular o estudo de alguns aspectos bastante inovadores que vêm sendo observados na web, como a aparição de diversos sites que utilizam idéias e algoritmos da chamada "inteligência coletiva". Achamos importante o estudo dessas inovações, e em particular um estudo mais teórico do que está acontecendo por trás de seu uso e porque muitas vezes aplicações que funcionam apenas com a contribuição de usuários da Internet parecem funcionar tão bem.

Todavia, cremos que a execução da disciplina deixou um pouco a desejar, em especial pelo tempo de trabalho que ela acabou exigindo. Achamos, sinceramente, que esse tempo fugiu do que seria o esperado, e acabou atrapalhando o aproveitamento ideal de alguns temas tratados. Bom, sabemos que muitos alunos que fizeram a disciplina já tinham bons conhecimentos acerca da maioria das tecnologias estudadas, em particular as básicas da primeira fase, e o processo pode ter sido mais fácil para eles. Não era o caso de nenhum de nós dois, que entramos na disciplina sem ter muita idéia da maior parte das coisas que veria, exceção feita a HTML e Java. É claro que não teria como fugir do estudo de *Servlets*, JSP, Hibernate e AJAX, e concordo que nosso conhecimento foi bastante engrandecido na área de tecnologias para *web* com essa disciplina. Porém, pensamos que, na segunda fase do projeto, exigir de todos os grupos a aprendizagem e implementação das 10 tecnologias diferentes foi um certo exagero, ainda mais com a apresentação de 2 tecnologias por semana. O que acabou acontecendo é que muitas vezes não tínhamos o tempo necessário para aprender **direito** uma determinada tecnologia, mas tínhamos que implementar o que conseguíssemos para já partir para a próxima. Em nossa opinião, o ideal seria restringir o número dessas tecnologias para algumas consideradas prioritárias, ou falar sucintamente de todas mas deixar em aberto para cada dupla a escolha de apenas algumas para um estudo mais detalhado e uma posterior implementação.

E, de alguma forma, achamos que o curso deveria focar mais na fase 3, que parece ser o grande diferencial. Afinal, aprender sobre tecnologias já bastante aceitas como Hibernate ou JSF não é uma grande dificuldade, com a grande quantidade de materiais disponíveis na rede (claro, dispondo de um tempo adequado). Porém, os algoritmos de inteligência coletiva mostram-se como algo realmente fascinante, que cada vez mais vemos na internet mas raramente lemos sobre seu real funcionamento. Talvez um espaço de apenas 4 aulas para a apresentação de todos os algoritmos selecionados foi pouco, e acabou dificultando um melhor aproveitamento das particularidades de cada um. Novamente, talvez pudesse ser escolhido um número menor de algoritmos para se focar, ou então pudesse ser reservado um tempo maior no curso para o estudo de todos eles. Por mais que o estudo desses algoritmos muitas vezes exija algum conhecimento das tecnologias apresentadas anteriormente, pensamos que deveria ser feito um esforço para focar o máximo possível do curso neles.

Para finalizar, devemos dizer que o curso foi bastante cansativo, o que acabou criando momentos de desânimo durante seu decorrer, e portanto em nossa opinião sua carga de trabalhos deveria ser revista. Porém, foi também bastante gratificante, em particular vendo o resultado final, com uma aplicação que, ainda que não cumprisse todos nossos objetivos iniciais, já mostrava-se muito bacana, e totalmente utilizável.