Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Disciplina Programação Modular (DCC0052) – 2016/01

Alexandre Alphonsos Rodrigues Pereira
Jerônimo Nunes Rocha
Felipe Marcelino

Mimimi

Sistema Web (Trabalho Prático 3)

Trabalho apresentado à disciplina de Programação Modular do curso de Ciência da Computação da UFMG.

Belo Horizonte
17 de Junho de 2016

1. Introdução:

O trabalho proposto foi a implementação de um sistema web semelhante ao Twitter que se chama Mimimi. O sistema é uma rede social online que permite que os usuários enviem, leiam e curtam mensagens curtas chamadas mimimi.

A ideia é que os mimimis sejam reclamações ou demonstrem insatisfação sobre os mais diversos temas. Como no Twitter, usuários podem ser marcados nas mensagens através do “@nomedousuario” ou utilizar o “#assunto” para classificar a respeito do que a mensagem trata, facilitando assim a busca por tópicos. É possível visualizar uma lista que contém todas as mensagens com o “#assunto” em ordem cronológica por exemplo.

Além disso cada usuário possui uma foto de perfil, uma foto de capa e uma pequena lista de informações básicas como: Nome, data de nascimento, cidade em que habita, que servem como apresentação da sua página principal com seus mimimis.

2. Implementação:

A implementação da Aplicação se dá através da linguagem de programação Java com o pré-requisito da máquina virtual 1.7. O pacote da aplicação pode ser gerado através da ferramenta de compilação Maven para ser carregado em um servidor existente ou executado através do linux shell com os comandos make && make run. Caso seja executada através do make run, a aplicação estará disponível na porta 8080 da máquina na qual está rodando.

2.1. Tecnologias e bibliotecas usadas:

2.1.1. Application Web Server: Apache Tomcat 8.0.28 embedded

O Tomcat é um servidor web Java, mais especificamente, um container de *servlets*. O Tomcat implementa, dentre outras de menor relevância, as tecnologias Java Servlet e JavaServer Pages (JSP). Foi escolhida a versão Embeded do servidor para facilitar as correções do trabalho prático, mas também é possível gerar um WAR (Web Application Resource) da aplicação e disponibilizá-la em um servidor Tomcat pré-instalado**.**

2.1.2. JPA - ORM

Mapeamento objeto-relacional (ou ORM, do inglês: Object-relational mapping) é uma técnica de desenvolvimento utilizada para reduzir a impedância da programação orientada aos objetos utilizando bancos de dados relacionais. As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes. (Trecho retirado da Wikipédia: https://pt.wikipedia.org/wiki/Mapeamento_objeto-relacional).

A aplicação Mimimi fez o uso da tecnologia ORM através da Java Persistence API (JPA) implementada pelo Hibernate. Foi feita essa escolha para agilizar o desenvolvimento da persistência dos dados da aplicação. O uso das ORMs também trazem a vantagem de que a aplicação não depende de uma implementação específica de banco de dados, apenas que a implementação utilizada implemente o Dialeto da API.

2.1.3. Database: Apache Derby

Apache Derby é um sistema de gerenciamento de banco de dados relacional Java que pode ser embutido em programas Java e usado para processamento de transações

online. Ele foi escolhido para uso no Mimimi pelo fato de ser 100% Java e por isso poder ser embutido na aplicação. Como foi utilizada a ORM do Hibernate a aplicação não é dependente do Apache Derby e pode-se optar por outro banco de dados mudando as configurações correspondentes no arquivo META-INF/persistence.xml.

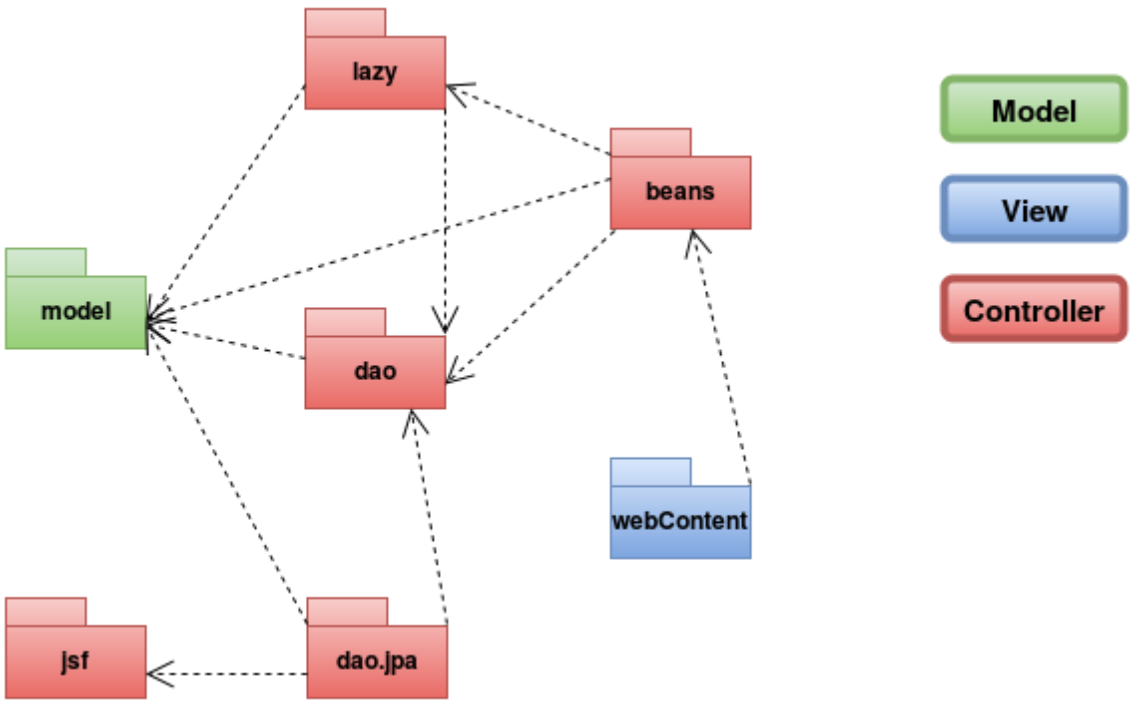
2.1.4. Frontend: HTML, CSS, javascript, JSF com PrimeFaces

JavaServer Faces (JSF) é uma especificação Java para a construção de interfaces de usuário baseadas em componentes para aplicações web. Possui um modelo de programação dirigido a eventos, abstraindo os detalhes da manipulação dos eventos e organização dos componentes, permitindo que o programador se concentre na lógica da aplicação. A implementação do JSF escolhida para o projeto foi a implementação da Apache, MyFaces.

PrimeFaces é uma biblioteca de componentes UI (User Interface) para JSF.

2.2. Estruturação dos pacotes:

A implementação do trabalho foi dividida em 7 pacotes conforme diagrama a seguir:



Cada pacote possui sua descrição (package-info.java), seu diagrama de classes (package-diagram.png) e suas respectivas classes, exceto o pacote WebContent, que não é um pacote Java, mas sim a raiz da aplicação Web.

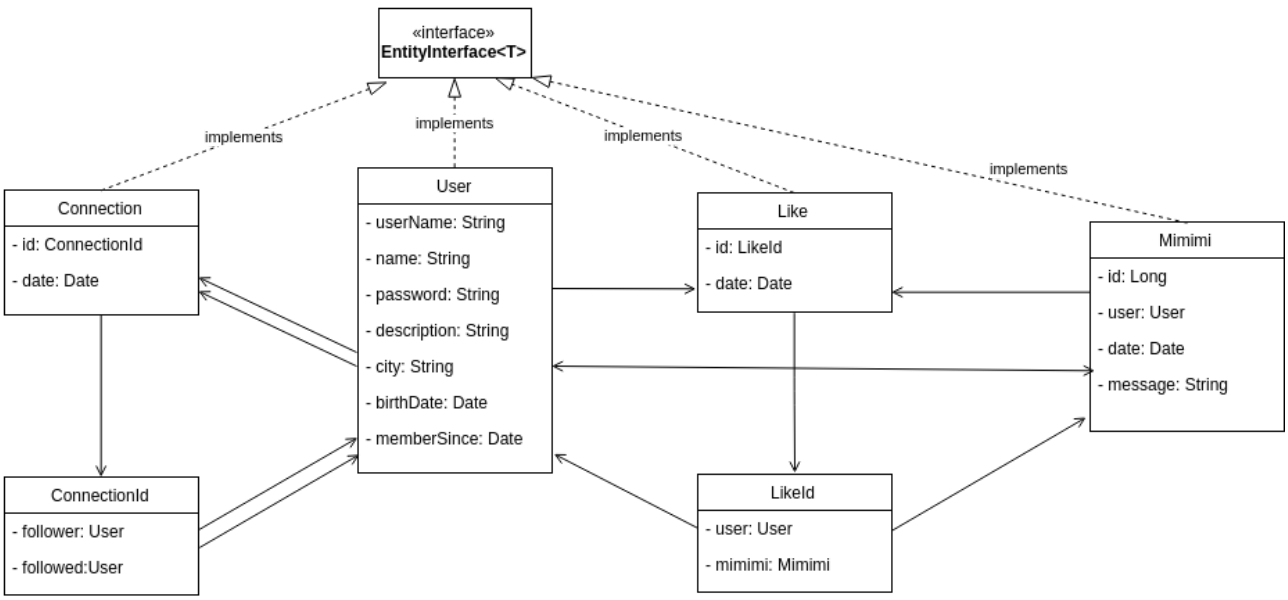
A aplicação foi desenvolvida pensando no padrão de projetos MVC onde os modelos, visualizações e as controladoras são separados para melhor entendimento do código. Nesse projeto os Modelos estão contidos em Model, as Views em WebContent e o controle é dado pelos outros pacotes.

2.2.1. WebContent

A pasta WebContent contém as pastas faces, META-INF, resources e WEB-INF. A pasta faces contém os arquivos que representam a interface do usuário, são processados no servidor pela Servlet do Java Server Faces e retornados como HTML, CSS e JavaScript. Nas pastas META-INF e WEB-INF estão contidos arquivos de configuração da aplicação web e na pasta resources são mantidos os JavaScript, CSS e imagens para ser utilizados junto ao HTML gerado pelas faces.

2.2.2 Model

O pacote Model contém os modelos da aplicação, as classes que armazenam os dados relevantes e por termos utilizados a tecnologia ORM, representam também as tabelas no banco de dados. Ele é o **pacote mais estável** da aplicação, não tendo nenhuma dependência de outros pacotes. Segue UML:



Como pode ser observado, todas as entidades implementam a interface EntityInterfaces, que possui o metódo #getId:T. Essa interface indica que a classe

representa uma tabela no banco de dados e por isso deve ser possuir uma chave primária. As classes LikeId e ConnectionId existem porque a chave primária (identificador) das classes Like e Connection respectivamente, possuem chave primária composta.

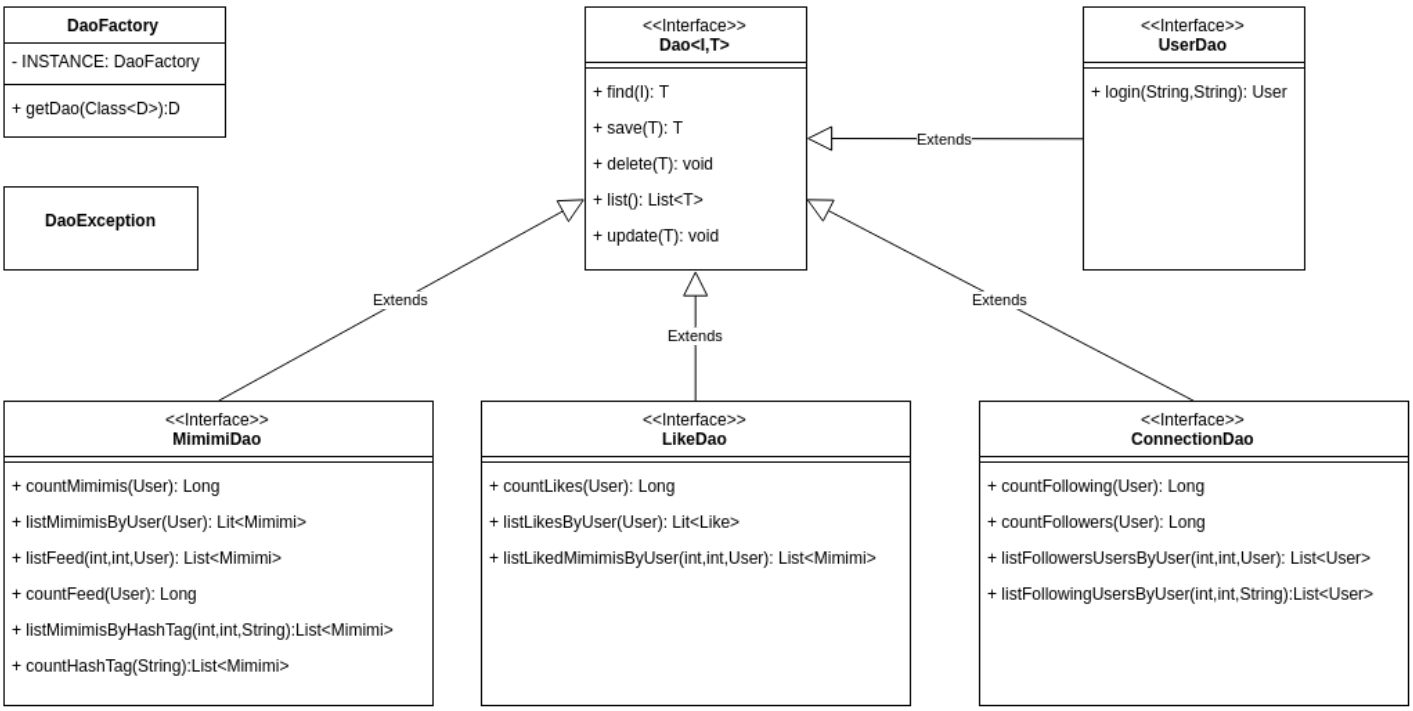
A modelagem da Conexão (seguinte e seguidor) foi feita utilizando uma classe para a mesma ao invés de simplesmente uma lista de seguintes e seguidores, não só para poder manter o modelo de uma classe para cada tabela no banco, mas também para poder manter um atributo nesse relacionamento n pra n que é a data da criação. O Like foi modelado de forma análoga pelo mesmo motivo.

O User e o Mimimi são os modelos principais, um para representar o usuário do sistema e o outro para representar a mensagem proposta. O Mimimi foi modelado como uma classe ao invés de uma lista de mensagens no usuário para que se possa trabalhar melhor com ela, por exemplo salvar mais detalhes como a data da postagem.

2.2.3. DAO

O DAO abstrai a origem e o modo de obtenção / gravação dos dados, de modo que o restante do sistema manipula os dados de forma transparente, sem se preocupar com o que acontece por trás dos panos. Isso ajuda muito em processos de migrações de fonte de dados e testes unitários. (Trecho retirado de http://javafree.uol.com.br/artigo/871452/Introducao-ao-pattern-DAO.html).

O Mimimi possui o pacote DAO com o contrato referente às possibilidades no que diz respeito ao acesso dos dados da aplicação conforme seguinte UML:



Optou-se pelo padrão de projeto **Fábrica** como forma de criação dos DAO para que não houvesse necessidade do programador conhecer a implementação do mesmo e para que ficasse facilitada a troca caso necessário. A fábrica presente neste pacote é **abstrata**, lendo qual classe concreta deve instanciar e guardar em INSTANCE (que é **estático**) de um arquivo de configuração, em tempo de execução e a instanciando através de **reflections**. Dessa forma o pacote DAO não possui dependências de nenhum outro pacote exceto modelo, sendo um pacote bem **estável**. A fábrica faz uso do padrão de projetos **Singleton**, para que não seja necessário ler o arquivo de configuração a cada acesso, o que a tornaria menos eficiente e também para dar maior controle a quem a implementa.

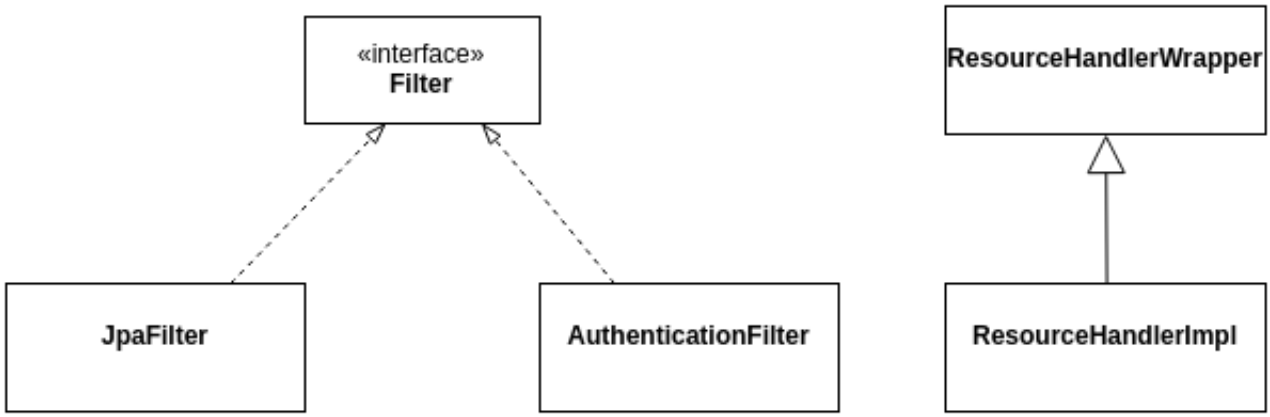
O pacote conta com uma interface Dao **genérica** que define um contrato de métodos básicos que devem ser implementados em qualquer Dao, um Dao específico para cada entidade (implementação de EntityInterface) e também uma **exceção verificada** DaoException que deve ser lançada caso alguma falha no acesso a dados ocorra.

2.2.4. JPA

Uma das implementações possíveis do pacote DAO foi feita para uso no trabalho com a JPA (Java Persistence API), mais especificamente com a implementação do Hibernate. Isso permitiu que os métodos ficassem extremamente simples pois as operações CRUD podem existir em uma classe **abstrata** e as consultas puderam ser realizadas através de JPQL (Java Persistence Query Language). Todos os DAO implementados no pacote JPA possuem construtores protected, visto que sua instanciação deve ser controlada pela Fábrica. Como os DAO JPA são **Thread safe**, eles também são **singleton** com sua instâncias sendo mantidas pela fábrica. O diagrama UML pode ser conferido junto ao pacote no projeto.

2.2.5. JSF

O pacote JSF contém classes que implementam ou estendem interfaces ou classes do Java Server Faces. Segue UML:



Ambos JpaFilter e Authentication Filter implementam a interface Filter definida em javax.servlet. O primeiro cria uma transação no banco de dados a cada requisição do usuário e a completa ao final. O segundo impede que um usuário não autenticado acesse páginas do restritas do sistema. Já a classe ResourceHandlerImpl, que estende ResourceHandlerWrapper, definida em javax.faces.application, intercepta requisições de javax.faces.application.Resource. Foi necessário realizar essa interceptação para que uma imagem padrão fosse transmitida caso o usuário ainda não tenha carregado sua foto de perfil ou de capa.

2.2.6. Lazy

O pacote Lazy é responsável por fazer acessos aos DAO e trazer informações gradativas à interface do usuário. Esse recurso é necessário para que não se carregue a tabela do banco de dados completa quando consultando usuários ou mimimis. O componente gráfico utilizado para tal finalidade foi um DataScroller da biblioteca Prime Faces, que nos exigiu a extensão da classe LazyDataModel e implementação dos métodos para carregar os dados.

Uma classe **abstrata** AbstractLazyList foi implementada para que os detalhes comuns às entidades do projeto pudessem ser mantidos e portanto prover reuso de código e um método abstrato foi definido para fazer a leitura específica dos dados. O uso de **generics** foi necessário para que se pudesse prover um melhor reaproveitamento de código. O pacote acompanha várias sub-classes de AbstractLazyList que podem ser conferidas no diagrama UML no pacote do projeto.

2.2.7. Beans

O pacote Beans contém todos os ManagedBeans da aplicação. ManagedBeans são, segundo Tutorials Point (http://www.tutorialspoint.com/jsf/jsf_managed_beans.htm),

um Java Bean regular gerenciado pelo JSF. Ele contém getters and setters, regras de negócio ou mesmo os valores dos formulários html e serve de base (modelo) para a construção dos componentes de visualização. A UML pode ser conferida junto ao pacote no projeto.

Os Beans do projeto Mimimi correspondem às páginas acessíveis ao usuário, sendo o LoginBean e o HeaderBean compartilhados entre as páginas. O primeiro armazena o usuário que entrou no sistema e o segundo o usuário selecionado para visualização. Os outros Beans mantém um objeto do tipo AbstractLazyList filtrado com alguma característica específica para acesso da face correspondente.

Toda modificação nos Beans e por conseguinte ao banco de dados é dada através do **padrão de projeto Observer**, no qual métodos nos ManagedBeans são registrados como listeners de eventos que ocorre nas faces (interface do usuário).

3. Testes unitários

Testes unitários para as faces do WebContent, os Beans, os elementos de JSF ou as LazyList não foram construídos devido ao fato de estes não conterem muita lógica, mas sim serem detalhes específicos de frameworks específicos. Ficaria como melhoria da aplicação mais testes para atenderem esses casos.

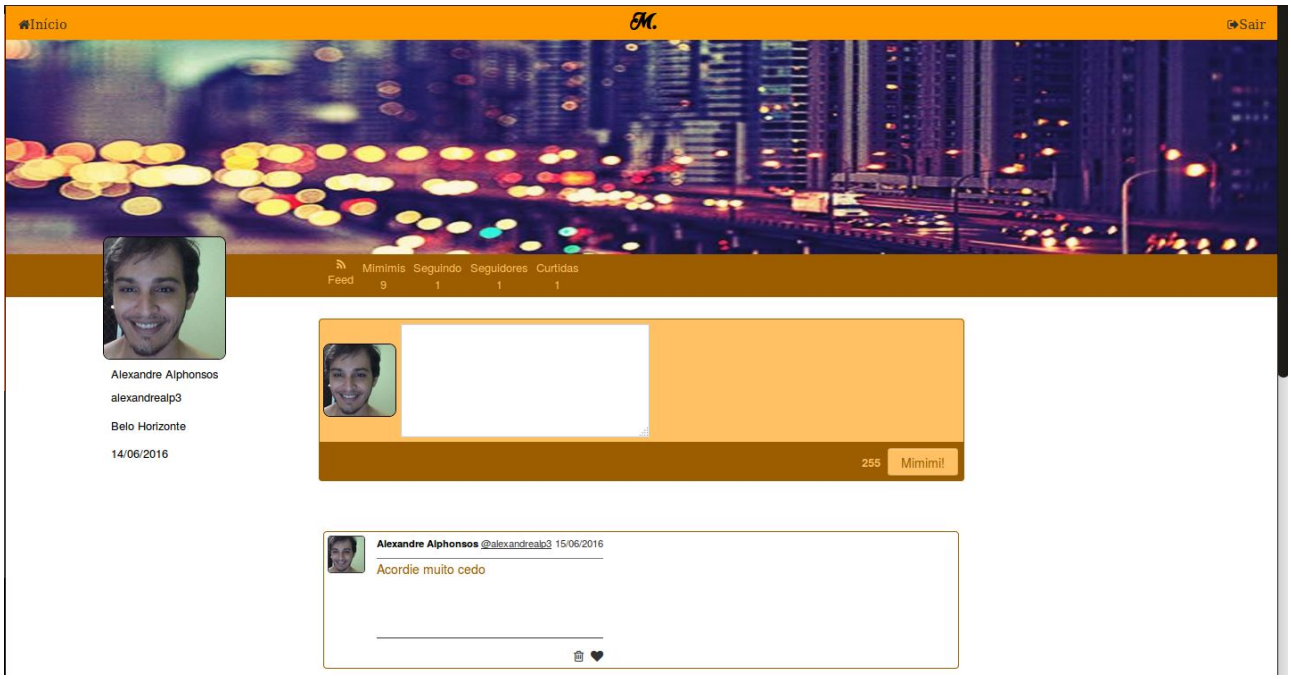
Foram criados testes para verificar o funcionamento dos DAO, mais especificamente para a implementação com a JPA e de questões específicas dos modelos User e Mimimi. Uma classe abstrata de testes foi criada para manter os detalhes comuns a todas as entidades e também um teste de inserção, listagem e remoção. Um teste para o usuário foi criado fazendo teste de login e um foi criado para o Mimimi para verificar a geração de URLs nas mensagens marcadas.

4. Interface:

A cor base para a aplicação foi escolhida arbitrariamente e as outras cores da paleta foram escolhidas através da ferramenta paletton (paletton.com) para que cores agradáveis aos usuários fossem utilizadas.

4.1. Tela de login:

4.2. Tela princinpal:



5. Melhorias

Alguns recursos importantes foram deixados de lado como a busca. Atualmente só é possível encontrar usuários ou visualizar assuntos clicando na marcação. Também seria interessante ter páginas diferentes para a visualização do usuário atual e de outros usuários. Páginas de erro também são uma melhoria interessante, atualmente se erros alcançarem a JSF os eventos da página serão ignorados, necessitando uma atualização ou às vezes outro login para que o sistema volte a operar.

6. Conclusão:

Concluiu-se com esse trabalho que os conceitos ensinados na disciplina de Programação Modular são extremamente importantes na criação de aplicações web como o Twitter e que uma aplicação como essa fica muito simples de manter caso seja desenvolvida aplicando bem os conceitos de programação modular.

7. Referências:

https://pt.wikipedia.org/wiki/Mapeamento_objeto-relacional

http://javafree.uol.com.br/artigo/871452/Introducao-ao-pattern-DAO.html

http://www.tutorialspoint.com/jsf/jsf_managed_beans.htm

http://paletton.com/