Relatório Milestone
Projeto Jackut

Descrição

O projeto Jackut consiste na implementação de uma pequena e simples rede social. Desenvolvida em Java e com comandos controlados pelo Terminal. Durante o uso do Jackut, é possível realizar ações como: Criar perfil, editar atributos do perfil, logar no perfil, fazer amigos, além de enviar e ler recados para os mesmos.

A implementação de todas estas funções foi devidamente testada por arquivos previamente definidos, localizados em *tests/*, com as verificações feitas pela biblioteca EasyAccept.

User Stories

A implementação das funções do projeto foram divididas em 4 etapas (ou *user stories*), cada uma com foco em um tipo específico de funcionalidade, confira cada uma a seguir:

User Story 1

Focada em possibilitar a criação e interação individual de perfis, na primeira etapa do desenvolvimento do site foram solicitadas 3 funções:

- **Criar usuário**: Utilizando login, senha e nome. Essa função consiste no salvamento destes dados no arquivo de armazenamento do projeto (*data.dat*), com verificação básica de dados (Login repetido, dados completos, etc.);
- **Ler atributo**: Esta função, a partir da persistência de dados, percorre pelo registro dos usuários para procurar uma conta específica pelo login. A partir do momento que é encontrado (se encontrado), a função retorna o atributo do usuário especificado na linha de comando;
- **Abrir sessão**: Semelhante a um processo de *sign in*, esta função retorna um id que permitirá ao usuário editar seu próprio perfil ou enviar e ler mensagens;

User Story 2

A próxima etapa nos apresenta uma nova função e um melhoramento de uma funcionalidade anterior:

- **Editar usuário**: A partir das sessões criadas na user story anterior, é possível editar dados do perfil logado, com uma novidade: Qualquer dado pode ser

registrado, até mesmo os que não estavam previamente definidos na função *Criar Usuário* (Ex.: Data de nascimento, gênero, descrição, etc.);
- **Ler atributo**: Foi necessária uma adaptação desta função por conta da nova funcionalidade do *Editar Usuário*, onde agora é possível passar qualquer atributo para esta função, e não apenas *nome* e *senha*, como implementado anteriormente;

User Story 3

Etapa focada na implementação de amizades (Classe *friendship*), todas as funcionalidades implementadas se relacionam com isto, vejamos a seguir:

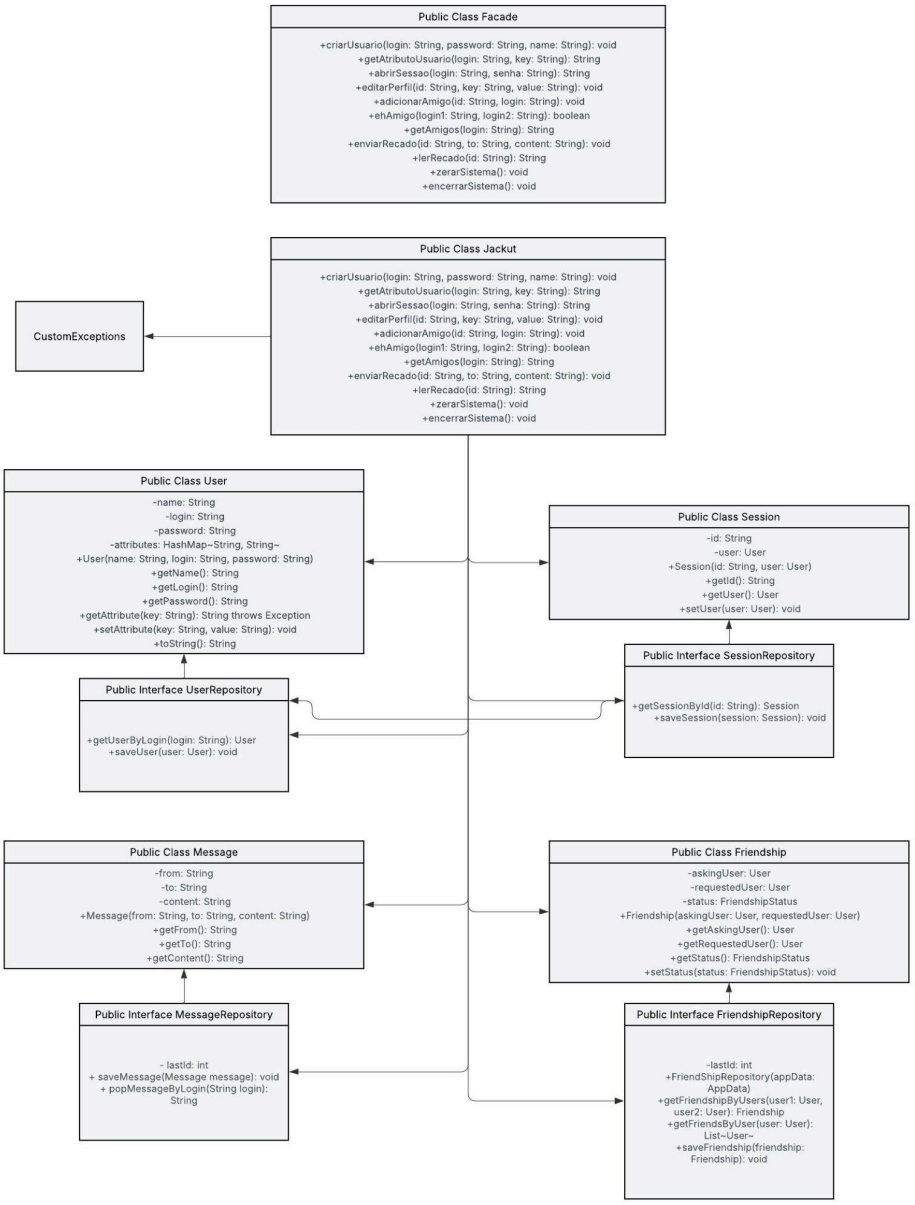
- **Adicionar amigo**: Consiste num pedido básico de amizade que precisa ser enviada pelos dois usuários para ser confirmada, caso seja enviada por apenas um, terá como status *pendente*. A verificação desta função acaba sendo um pouco mais rigorosa por conta de diferentes cenários que envolvem os dois usuários;
- **É amigo?**: Esta função retorna um *Boolean* que confirma se os dois usuários informados possuem uma amizade com status *confirmada*;
- **Ler amigos**: Esta função retorna uma *String*, que retorna todos os amigos do perfil informado pelo Terminal, separados por vírgula;

User Story 4

O último passo da implementação das funções teve o objetivo de possibilitar o envio e leitura de recados:

- **Enviar recado**: Caso o destinatário tenha mais de uma mensagem não lida, elas ficam acumuladas para que possam ser lidas individualmente;
- **Ler recado**: Lê a primeira mensagem da pilha de mensagens do usuário informado, caso não tenha nenhuma, é retornada uma exceção;

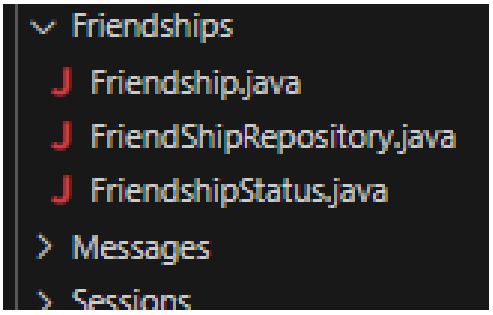
Diagrama de classes



[Link para SVG]

Modelos

Cada modelo (ou entidade) presente na aplicação segue a seguinte estrutura, usamos de exemplo a classe *Friendship*, temos a seguinte pasta:



Cada entidade possui suas classes relacionadas numa pasta própria

Na classe da entidade, está presente seus dados, construtor e métodos utilizados para alterar seu registro individual.

Repositórios

Os repositórios, presentes em cada entidade, são responsáveis pelos métodos que se conectam diretamente com o arquivo de armazenamento, seja para salvar usuários, criar amizades, deletar mensagens, etc.

Toda função que atue diretamente com a manipulação dos dados da aplicação será chamada a partir de uma classe de repositório.

Facade

Separadas pelas funções de cada User Story, esta classe é utilizada para apontar para os métodos que serão testados pelo EasyAccept, geralmente direcionado para os métodos da classe *Jackut*.

Jackut

Classe principal do projeto, onde ficam localizadas todas as funções chamadas pela *Facade*, nela se carregam, alteram e lêem os dados salvos no arquivo de armazenamento (*data.dat*).

As demais funções, também divididas pela sua respectiva User Story contém toda a lógica necessária para sua execução, seja a verificação de parâmetros, tratamento dos dados ou até a invocação dos métodos.

AppData

Esta classe é o *core* do armazenamento dos dados dos modelos presentes no Jackut (usuários, amizades, recados e sessões), os dados obtidos a partir desta função são passados para os construtores dos repositórios dos modelos, para que se faça uma conexão direta das novas alterações com o arquivo de armazenamento.

AppConfig

Responsável por armazenar dados importantes da aplicação, no contexto atual, armazena apenas o caminho para o arquivo de armazenamento (*/data.dat*).

Exceções customizadas

No caso de exceções que não necessariamente se encaixa nas exceções nativas do Java, foram criadas classes personalizadas para representá-las, localizadas na pasta *Exceptions/*, estas são as seguintes:

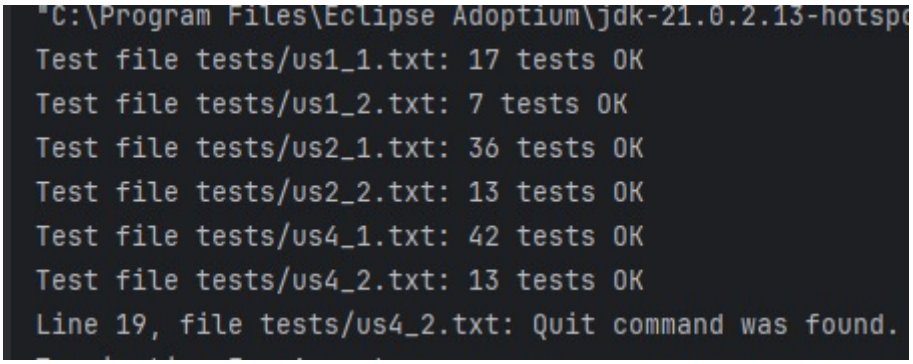
- **UserNotRegisteredException** (Usuário não registrado);
- **UserAlreadyExistsException** (Usuário já existe);
- **InvalidLoginOrPasswordException** (Login ou senha inválidos);
- **FriendshipExistsException** (Amizade já existente);
- **SamePersonFriendshipException** (Pedido de amizade para si mesmo);
- **SamePersonMessageException** (Mensagem para si mesmo);
- **NoMessagesException** (Usuário sem mensagens pendentes);

Cada exceção permite a exibição de uma mensagem customizada, estas foram padronizadas por conta dos testes do EasyAccept.

Testes

Os testes para verificar a confiabilidade das funções e persistência de dados do Jackut foi feito pela biblioteca *EasyAccept* e pelos arquivos localizados na pasta *tests/* (2 para cada User Story).

Em cada arquivo é possível ler uma descrição do objetivo dos testes, sempre baseados nas etapas citadas anteriormente.



Todos os testes dos arquivos presentes passaram corretamente

Obs.: Para facilitação da execução e leitura das mensagens e testes, eu comentei o comando 'quit' dos 5 primeiros arquivos (para que os testes sejam feitos de uma vez só), também troquei caracteres como 'á' para 'a' para evitar erros de codificação, sem mudança alguma no conteúdo dos testes ou mensagens.