UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA: PROGRAMAÇÃO 2
ALUNO: CAIO CÉSAR MEDEIROS LOPES

RELATÓRIO DO MILESTONE 2 (JACKUT)

Maceió, 5 de maio de 2025

1. INTRODUÇÃO

Este relatório apresenta uma visão geral do projeto Jackut, um sistema de rede social fictícia. A Milestone 2 apresenta testes de aceitação de us_5 a us_9 que visam implementar novas funcionalidades no sistema: criação e gerenciamento de comunidades, novos tipos de relações entre usuários e remoção de usuários do sistema. Os testes consistem em arquivos de texto (.txt) utilizando o EasyAccept para sua execução e utiliza a interface *Serializable* para permitir a persistência de dados do sistema.

2. ARQUITETURA E ORGANIZAÇÃO DO CÓDIGO

O projeto está divido em pacotes:

- code: classes que fazem parte do sistema e modelam suas funcionalidades, havendo separações para delegações menores.
- exceptions: separada de maneira similar ao pacote code porém lidando com exceções customizadas para lidar com o tratamento de erros de forma consistente.

A estrutura possui a classe *Main* que é responsável pela execução dos devidos testes com o EasyAccept dentro do sistema utilizando a classe *Facade*. O projeto organiza os dados de persistência no diretório data enquantos os testes do sistema são armazenados no diretório tests.

3. AVALIAÇÃO DAS NOVAS FUNCIONALIDADES

3.1 COMUNIDADES (us: 5, 6 e 7)

A inserção do sistema de comunidades é simples, tendo em vista que não trás grandes alterações em classes previamente implementadas na Milestone 1. Para ela foi necessária a implementação das classes:

- *Community*: gerencia informações da própria comunidade, assim como seus membros.
- *CommunityManager*: classe utilizada pelo sistema Jackut para que o mesmo possa fazer o correto gerenciamento de todas as comunidades.

O sistema de comunidades também trás para o sistema a funcionalidade de envio de mensagens de um usuário para a comunidade, para tal, foi necessária uma pequena refatoração, utilizando o padrão de projeto Strategy a classe *Message* (utilizada na Milestone 1 apenas para fazer o envio de mensagens privadas) foi refatorada para uma interface, para que fosse possível implementar de forma distinta e consistente o envio de mensagens de um usuário para outro (*PrivateMessage*) e de um usuário para uma comunidade (*CommunityMessage*).

3.2 NOVOS RELACIONAMENTOS (us: 8)

A implementação deste sistema requeriu um maior refatoramento no código, foi feito o desacoplamento dos metódos de relações do usuário (apenas relações de amizade na Milestone 1) da classe *User* para a nova classe *RelationsManager*, a mesma é utilizada pela classe *User* para fazer o gerenciamento de suas relações com outros usuários por meio de listas distintas.

3.3 REMOÇÃO DE UM USUÁRIO (us: 9)

Para a última user stories não houve implementação de novas classes nem refatoramentos, apenas adições dos métodos necessários para a devida remoção de um usuário, foram adicionados os métodos:

- **removeUser** na classe *Jackut* que chama os metódos das classes que precisam fazer as suas devidas remoções.
- **removeUserFromCommunitys** na classe *CommunityManager* que se responsabiliza de remover o usuário em questão das comunidades na qual ele participa.
- **removeSentMessagesFromUser** e **removeUserReferences** na classe *UserManager* que se responsabiliza por buscar e excluir as mensagens e relações que o usuário a ser removido possa ter com outros usuários assim como a exclusão dele na lista de usuários.
- **removeSession** na classe *SessionManager* que remove a sessão ativa do usuário do gerenciador de sessões após sua devida exclusão.

4. PADRÕES DE PROJETO UTILIZADOS

Foram utilizados os seguintes padrões de projeto:

- Facade: utilizado por todos os alunos para que fosse possível utilizar testes no sistema e validar seu funcionamento por meio do EasyAccept.
- Strategy: com a interface *Message*, as classes *PrivateMessage* e *CommunityMessage* a implementam de forma a permitir diferentes lógicas de envio.
- Template Method: utilizado na classe *SerializableData*, a mesma visa permitir que uma classe filha possa ser serializada permitindo persistência mas abstraindo quase toda a lógica necessária para tal, apenas o método **castObject** deve ser implementado pela classe filha para garantir a correta desserialização do objeto para a classe que a estendeu. As classes que estendem esse template são: *UserManager* e *CommunityManager*, ambas utilizadas na classe *Jackut*.

5. POSSIVEIS MELHORIAS

Visando uma melhor consistência em certos pontos do sistema, algumas melhorias que sugiro são:

- Uso do padrão de projeto Singleton para garantir apenas uma instância da classe *Jackut* visando evitar múltiplos acessos aos arquivos de persistência do sistema.

- Uso do padrão Strategy para definir as diferentes relações entre usuários de forma mais sólida ao invés de delegar ao sistema e a classe *RelationsManager* a maneira de como lidar a lógica de relações como descrito no tópico 3.2.
- Uso do padrão Observer para:
    - desacoplar a lógica de remoção de um usuário de outras classes e reduzir a complexidade de tal operação guardando quais usuários ele possui alguma relação (vale observar que algumas relações não são reciprocas e/ou “secretas”).
    - as ações que devem ser tomadas em certas relações entre usuários (a exemplo a relação de paquera onde uma mensagem é enviada para cada usuário caso haja uma paquera mútua).
- Qualidades menores que não vem diretamente ao objetivo da matéria, como: limitar o tempo de sessão de um usuário, verificação mais robusta de senhas, implementação de um banco de dados para as informações necessárias na persistência, criação de mais métodos na classe *User* para abstrair a necessidade do uso do método **getRelationManager** em partes do código no geral (melhoria não implementada para evitar uma sobrecarga de linhas na classe *User*, também é uma possível oportunidade de utilizar o padrão de projeto Bridge).

6. CONCLUSÃO

A implementação das novas funcionalidades para a Milestone 2 foi bem-sucedida, trazendo novas capacidades ao sistema. A aplicação de padrões de projeto junto das noções de POO ajudou na visualização de novos caminhos e diferentes formas de organizar o código e o projeto. As melhorias sugeridas visam aumentar ainda mais a consistência e eficiência do sistema, além de reduzir a complexidade em operações críticas como a remoção de usuários.