UNIVERSIDADE FEDERAL DE ALAGOAS

INSTITUTO DE COMPUTAÇÃO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DISCIPLINA: PROGRAMAÇÃO 2

ALUNA: STHEFANY BARBOZA DE LIMA

RELATÓRIO DO MILESTONE 2

MACEIÓ, 05 DE MAIO DE 2025

1. INTRODUÇÃO

Este relatório descreve as atividades realizadas no desenvolvimento e refatoração do projeto Jackut, uma rede social desenvolvida em Java, seguindo os princípios de Orientação a Objetos e aplicando padrões de projetos, focando nas funcionalidades implementadas e nos ajustes realizados para atender aos requisitos estabelecidos nas user stories. O relatório está estruturado da seguinte forma:

- **Avaliação do Projeto**: Discussão sobre a estrutura e o design do sistema, incluindo os pontos positivos e negativos.

- **Refatoramento e Desenvolvimento Realizado**: Descrição das mudanças realizadas no código para a implementação das novas funcionalidades.

- **Conclusão**: Resumo do processo, dificuldades enfrentadas e impactos das mudanças no código.

2. AVALIAÇÃO

Aqui, descrevemos a avaliação do código do sistema, com foco na análise de seu design, qualidade e possíveis áreas de melhoria.

2.1 Virtudes do Projeto

- O sistema apresenta uma boa organização geral, com uma divisão clara entre os pacotes.

- As classes principais estão bem estruturadas e documentadas com Javadoc.

- As funcionalidades do sistema estão distribuídas de forma coesa, e a utilização de padrões de design foi adequada para as funcionalidades já implementadas.

2.2 Fraquezas do Projeto

- **Exceções**: O sistema apresenta excesso de tipos de exceções, o que pode prejudicar a legibilidade e manutenção do código. A granularidade das exceções pode ser reduzida.

- **Acoplamento**: O código possui um alto grau de acoplamento entre as classes principais, o que dificulta a extensibilidade e reutilização.

- **Persistência**: O uso de arquivos de texto pode ser menos eficiente e menos seguro que um banco de dados para sistemas maiores.

- **Desacoplamento de Responsabilidades**: Algumas classes estão responsáveis por mais tarefas do que o necessário, violando o princípio de alta coesão.

3. REFATORAMENTO E DESENVOLVIMENTO REALIZADO

Mudanças implementadas no código, incluindo detalhes sobre como os requisitos das user stories foram atendidos.

3.1 User Story 1 - Criação de conta

Descrição da funcionalidade: Esta user story se concentra na criação de uma conta de usuário no sistema Jackut, onde é necessário fornecer um login, senha e nome.

Padrão utilizado: A implementação seguiu um modelo simples de verificação de dados de entrada e persistência, utilizando a classe `Facade` para gerenciar a criação do usuário e garantir que o login fosse único. O padrão **Factory Method** pode ser observado aqui, pois a criação de um novo usuário envolve uma validação rigorosa.

Refatoramento realizado: A classe `Facade` foi refatorada para garantir que a criação de novos usuários fosse realizada de forma consistente. O método `criarUsuario` foi implementado com checagem de erros para garantir que o login não estivesse em uso e que os dados fornecidos fossem válidos. A verificação de "Usuário não cadastrado" foi implementada para garantir a integridade dos testes.

3.2 User Story 2 - Criação/Edição de perfil

Descrição da funcionalidade: Esta user story permite que um usuário cadastrado edite ou crie atributos em seu perfil, como "estado civil", "idiomas", "cidade natal", entre outros.

Padrão utilizado: Utilizou-se o padrão **Builder** para a construção do perfil do usuário, permitindo adicionar ou editar atributos dinâmicos, sem a necessidade de criar múltiplas instâncias.

Desenvolvimento e mudanças: A classe `Perfil` foi criada para armazenar os atributos personalizados de cada usuário. O método `editarPerfil` foi introduzido para modificar os dados do perfil, e a classe `Usuario` agora possui um mapa de atributos para armazenar os valores preenchidos.

3.3 User Story 3 - Adição de amigos

Descrição da funcionalidade: A funcionalidade permite que um usuário envie um convite para adicionar outro usuário como amigo. O relacionamento é confirmado quando o outro usuário aceita o convite.

Padrão utilizado: O padrão **Observer** foi utilizado, onde as mudanças nas relações de amizade entre os usuários são observadas e atualizadas conforme as ações de adição de amigos.

Refatoramento realizado: Foram feitas melhorias na classe `Usuario` para lidar com convites de amizade, com os métodos `adicionarAmigo`, `aceitarConvite` e `temConvitePendenteDe`. Também houve alterações na classe `Facade` para garantir que o processo de envio e aceitação de convites fosse feito de forma eficiente e sem falhas.

3.4 User Story 4 - Envio e leitura de recados entre usuários

Descrição da funcionalidade: Permitindo que um usuário envie um recado a outro usuário dentro da plataforma Jackut.

Padrão utilizado: O padrão **Command** foi implementado para garantir que as ações de enviar e ler recados fossem tratadas de forma modular e desacoplada.

Desenvolvimento e mudanças: A classe `Usuario` foi expandida para incluir a funcionalidade de receber e ler recados através das filas de recados. O método `enviarRecado` foi implementado para garantir que a comunicação entre usuários fosse eficiente.

3.5 User Story 5 - Criação de comunidades

Descrição da funcionalidade: Esta user story permite a criação de comunidades, onde o criador se torna o dono e pode gerenciar os membros.

Padrão utilizado: O padrão **Singleton** foi usado na classe `Facade`, garantindo que as instâncias de comunidade sejam gerenciadas por uma única fonte. Além disso, foi utilizado um controle de unicidade para o nome das comunidades, o que impede duplicações.

Refatoramento realizado: A classe `Comunidade` foi criada e refatorada para permitir a gestão de membros, incluindo o método `adicionarMembro`. A classe `Facade` foi atualizada para suportar a criação e a verificação de comunidades existentes.

3.6 User Story 6 - Adição de comunidades

Descrição da funcionalidade: Permite que um usuário se adicione a uma comunidade existente.

Padrão utilizado: O padrão **Facade** foi utilizado para simplificar o gerenciamento das ações do usuário dentro do sistema, centralizando a interação com as comunidades na classe `Facade`.

Desenvolvimento e mudanças: A funcionalidade foi integrada na classe `Facade` com o método `adicionarComunidade`, que garante que o usuário possa se inscrever nas comunidades que desejar, desde que ainda não faça parte delas.

3.7 User Story 7 - Envio de mensagens a comunidades

Descrição da funcionalidade: Usuários devem ser capazes de enviar mensagens para uma comunidade, e todos os membros da comunidade devem recebê-las.

Padrão utilizado: O padrão **Observer** foi novamente utilizado aqui, para que as mensagens enviadas a uma comunidade sejam recebidas por todos os membros dessa comunidade.

Refatoramento realizado: A classe `Usuario` foi expandida para incluir a funcionalidade de receber mensagens de comunidade. A classe `Facade` também foi modificada para garantir que o envio de mensagens para comunidades fosse eficaz e sem erros.

3.8 User Story 8 - Criação de novos relacionamentos

Descrição da funcionalidade: Permite que um usuário crie diferentes tipos de relacionamentos, como fãs, ídolos, paqueras e inimigos, além de amigos.

Padrão utilizado: O padrão **Strategy** foi utilizado para permitir a flexibilidade na criação e gerenciamento desses novos tipos de relacionamento.

Desenvolvimento e mudanças: A classe `Usuario` foi expandida para incluir métodos que gerenciam fãs, ídolos, paqueras e inimigos. Cada tipo de relacionamento possui regras específicas, que foram implementadas para garantir que os relacionamentos fossem adicionados corretamente.

3.9 User Story 9 - Remoção de conta

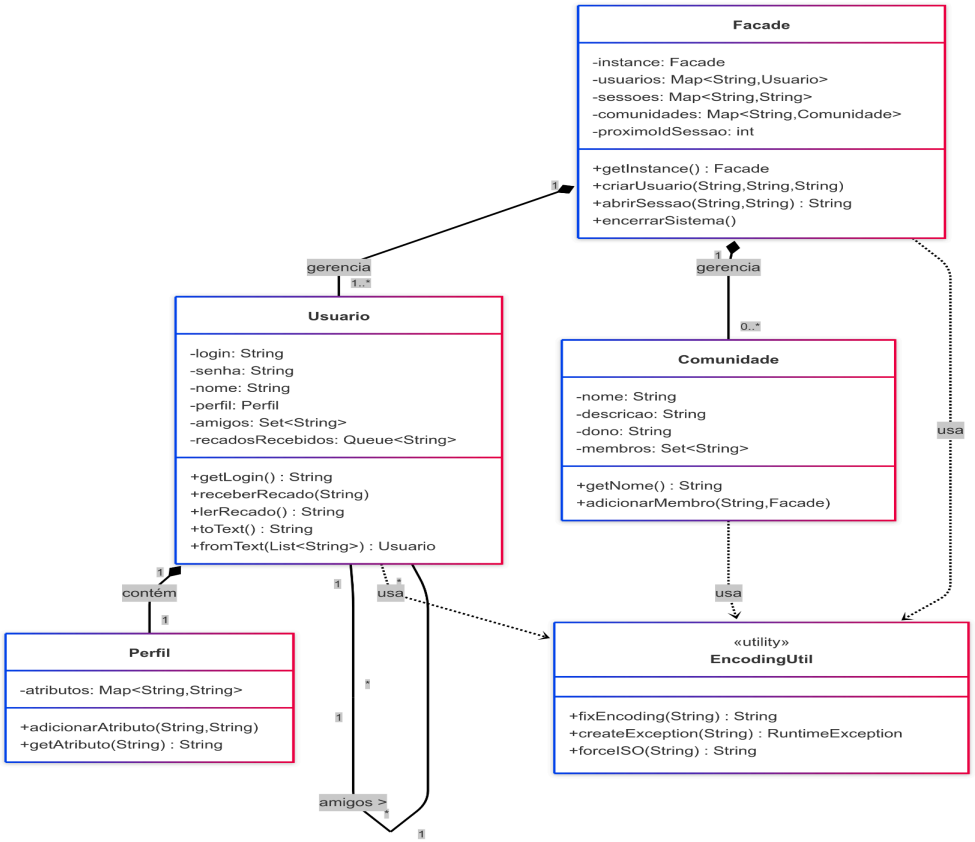
Descrição da funcionalidade: Permite a um usuário encerrar sua conta, apagando todas as suas informações do sistema, incluindo mensagens, relacionamentos e perfil.

Padrão utilizado: O padrão **Composite** foi aplicado aqui para garantir que a remoção do usuário fosse feita de forma integral, apagando todas as referências ao usuário em várias classes.

Refatoramento realizado: A classe `Facade` foi aprimorada para incluir o método `removerUsuario`, que garante a exclusão completa do usuário e de suas informações. As verificações de erros foram implementadas para garantir que a exclusão não falhasse em caso de inconsistências.

4. DIAGRAMA DE CLASSES

O diagrama de classes a seguir ilustra a estrutura do sistema após as modificações realizadas. Ele reflete as classes principais, suas interações e a organização geral do sistema.



4.1. Facade (Fachada)

- O que é: Responsável por centralizar a interação que controla todo o sistema Jackut.
- Armazena:
    - Lista de usuários (usuarios)
    - Sessões ativas (sessoes)
    - Comunidades (comunidades)
- O que faz:
    - Cria usuários (criarUsuario())
    - Gerencia logins (abrirSessao())
    - Controla o sistema (encerrarSistema())
- Relacionamentos:
    - Gerencia múltiplos Usuario e Comunidade.
    - Usa EncodingUtil para tratamento de texto.

4.2. Usuario

- O que é: Responsável pela gestão dos dados do usuário e interações com a comunidade.
- Armazena:

    - Dados básicos (login, senha, nome)
    - Perfil personalizado (perfil)
    - Lista de amigos (amigos)
    - Recados recebidos (recadosRecebidos)
- O que faz:
    - Gerencia recados (receberRecado(), lerRecado())
    - Converte dados para texto (toText(), fromText())
- Relacionamentos:
    - Tem um Perfil (composição).
    - Pode ter amizade com outros Usuario (auto-relacionamento).
    - Usa EncodingUtil para codificação.

4.3. Comunidade

- O que é: Representa a comunidade e gerencia os membros.
- Armazena:
    - nome, descricao
    - dono (login do criador)
    - membros (logins dos participantes)
- O que faz:
    - Adiciona membros (adicionarMembro())
- Relacionamentos:
    - Gerenciada pela Facade.
    - Usa EncodingUtil indiretamente.

4.4. Perfil

- O que é: Atributos personalizados do usuário (ex: idade, interesses).
- Armazena:
    - Pares atributo=valor (atributos)
- O que faz:
    - Adiciona/recupera atributos (adicionarAtributo(), getAtributo())
- Relacionamento:
    - Pertence a um Usuario (composição).

4.5. EncodingUtil

- O que é: Classe utilitária para tratamento de texto.
- O que faz:
    - Corrige encoding (fixEncoding())
    - Cria exceções (createException())
    - Força codificação ISO (forceISO())
- Relacionamento:

- Usada por Facade, Usuario e Comunidade.

Relacionamentos Chave:

1. Facade → Usuario/Comunidade:
   - A Facade gerencia todas as instâncias de Usuario e Comunidade (composição).
2. Usuario → Perfil:
   - Cada Usuario tem exatamente um Perfil (composição).
3. Usuario → Usuario:
   - Usuários podem ser amigos de outros usuários (auto-associação).
4. Dependências → EncodingUtil:
   - Várias classes usam EncodingUtil para manipulação de texto (dependência).

---

5. CONCLUSÃO

O processo de refatoração e desenvolvimento das funcionalidades foi bem-sucedido, proporcionando melhorias significativas na estrutura do código e permitindo a implementação das novas funcionalidades de maneira eficiente. No geral, o projeto segue boas práticas de design, princípios básicos de Orientação a Objetos e padrões de projetos, embora existam áreas que ainda necessitam de melhorias, como o desacoplamento de algumas classes.

5.1. Avaliação Objetiva

- **Qualidade da Documentação**: 9.0 – A documentação foi mantida atualizada, com a maior parte do código comentado utilizando Javadoc.

- **Qualidade do Design**: 8.0 – O design segue boas práticas, mas alguns pontos de acoplamento podem ser melhorados e realizar uma distribuição mais equilibrada de responsabilidades.

- **Qualidade do Código**: 8.5 – O código está bem estruturado, mas poderia ter um tratamento mais refinado de exceções e menor acoplamento.

5.3. Recomendações para Melhorias

1. Implementar um sistema de hash para armazenamento seguro de senhas.
2. Dividir a classe Facade em classes menores com responsabilidades mais específicas.
3. Criar classes de exceção específicas para diferentes tipos de erros.
4. Considerar o uso de um banco de dados em vez de arquivos texto para persistência.
5. Implementar testes unitários para garantir a qualidade do código.