UNIVERSIDADE FEDERAL DE ALAGOAS
DISCIPLINA: PROGRAMAÇÃO 2
ALUNA: CARLA EVELLYN FERREIRA DOS SANTOS

RELATÓRIO DO MILESTONE 2

MACEIÓ - AL, 05/05/2025

1. INTRODUÇÃO

Este relatório apresenta a avaliação e implementação do sistema Jackut, uma rede social desenvolvida em Java como parte do Milestone 2 da disciplina de Programação 2. O projeto foi estruturado seguindo princípios de orientação a objetos, design patterns e boas práticas de programação. O relatório está organizado da seguinte forma:

- Seção 2 (Avaliação): Análise crítica do design e implementação do sistema, destacando pontos fortes e áreas para melhoria.
- Seção 3 (Refatoramento e Desenvolvimento)**: Descrição das funcionalidades implementadas e dos padrões utilizados.
- Seção 4 (Conclusão): Avaliação geral do trabalho realizado e sugestões para futuras melhorias.

2. AVALIAÇÃO

O sistema Jackut apresenta um design modular e bem organizado, com separação clara de responsabilidades entre as classes. A seguir, são destacados os principais aspectos do projeto.

2.1 VIRTUDES DO PROJETO

- Arquitetura Modular: O sistema foi dividido em pacotes (`models`, `managers`, etc.), cada um com responsabilidades bem definidas.
- Facade Pattern: A classe `Facade` centraliza as operações do sistema, fornecendo uma interface simplificada para interação com os módulos internos.
- Persistência em XML: Utilização da biblioteca XStream para serialização em XML, facilitando a leitura e manutenção dos dados.
- Baixo Acoplamento: As classes `UserManager`, `CommunityManager`, e `RelationshipManager` são independentes, comunicando-se apenas através de interfaces bem definidas.

- Tratamento de Exceções: Mensagens de erro claras e específicas são lançadas em situações inválidas, como tentativas de login incorreto ou operações não permitidas.

2.2 FRAQUEZAS DO PROJETO

- Exceções Genéricas: Algumas exceções poderiam ser mais específicas para facilitar a depuração (ex: `RuntimeException` em vez de exceções customizadas).
- Documentação de Métodos: Embora a maioria dos métodos esteja documentada, alguns carecem de descrições mais detalhadas sobre seu comportamento e retorno.
- Testes de Aceitação: O sistema utiliza o framework EasyAccept, mas os scripts de teste poderiam ser mais abrangentes para cobrir casos extremos.
- Redundância em Métodos: Alguns métodos, como `getAmigos` e `getFas`, possuem lógica de ordenação repetida, o que poderia ser abstraído em uma classe utilitária.

3. REFATORAMENTO E DESENVOLVIMENTO REALIZADO

3.1 IMPLEMENTAÇÕES PRINCIPAIS

Persistência de Dados:

- Utilização do padrão Singleton no `PersistenceManager` para garantir uma única instância durante a execução.
- Serialização em XML para armazenar usuários e comunidades, facilitando a recuperação de dados.

Gerenciamento de Relacionamentos:

- Implementação de métodos para adicionar amigos, ídolos, paqueras e inimigos, com validações para evitar conflitos (ex: autoamizade).
- Uso de estruturas de dados como `LinkedHashSet` para manter a ordem de inserção e evitar duplicatas.

Comunidades:

- Criação e gerenciamento de comunidades, com capacidade de enviar mensagens para todos os membros.

Sessões de Usuário:

- Controle de sessões ativas através do `SessionManager`, garantindo segurança e isolamento entre usuários.

3.2 PADRÕES UTILIZADOS

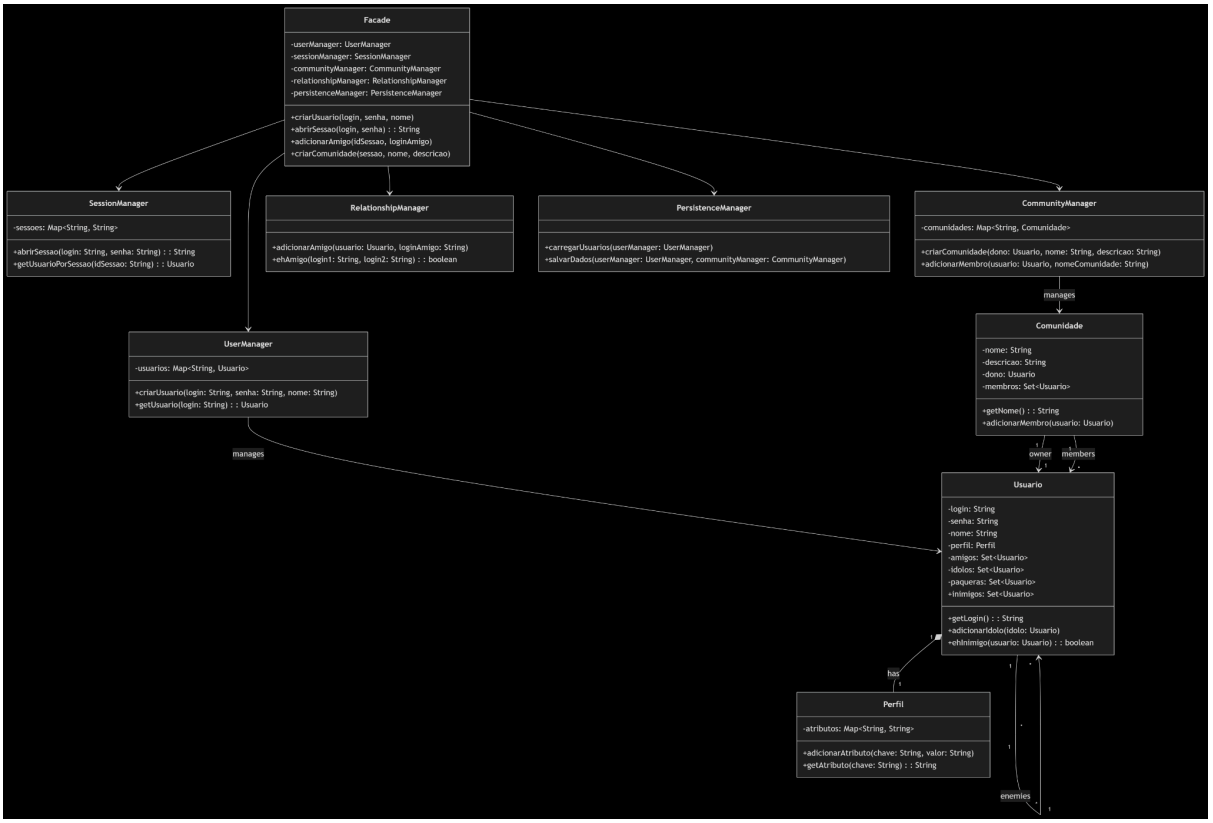
- Facade: A classe `Facade` simplifica a interação com o sistema, escondendo a complexidade dos módulos internos.
- Singleton: `PersistenceManager` garante que apenas uma instância gerencie a persistência de dados.
- Observer: Embora não explicitamente implementado, o envio de mensagens para comunidades segue o princípio de notificação múltipla.

3.3 DIAGRAMA DE CLASSE (SIMPLIFICADOS)

```
classDiagram
%% Main Classes
class Facade {
  -userManager: UserManager
  -sessionManager: SessionManager
  -communityManager: CommunityManager
  -relationshipManager: RelationshipManager
  -persistenceManager: PersistenceManager
  +criarUsuario(login, senha, nome)
  +abrirSessao(login, senha): String
  +adicionarAmigo(idSessao, loginAmigo)
  +criarComunidade(sessao, nome, descricao)
}

class Usuario {
  -login: String
  -senha: String
  -nome: String
  -perfil: Perfil
  -amigos: Set~Usuario~
```

```
    -idolos: Set~Usuario~
    -paqueras: Set~Usuario~
    +inimigos: Set~Usuario~
    +getLogin(): String
    +adicionarIdolo(idolo: Usuario)
    +ehInimigo(usuario: Usuario): boolean
}

class Perfil {
    -atributos: Map~String, String~
    +adicionarAtributo(chave: String, valor: String)
    +getAtributo(chave: String): String
}

class Comunidade {
    -nome: String
    -descricao: String
    -dono: Usuario
    -membros: Set~Usuario~
    +getNome(): String
    +adicionarMembro(usuario: Usuario)
}

%% Managers
class UserManager {
    -usuarios: Map~String, Usuario~
    +criarUsuario(login: String, senha: String, nome: String)
    +getUsuario(login: String): Usuario
}

class SessionManager {
    -sessoes: Map~String, String~
    +abrirSessao(login: String, senha: String): String
    +getUsuarioPorSessao(idSessao: String): Usuario
}

class CommunityManager {
    -comunidades: Map~String, Comunidade~
    +criarComunidade(dono: Usuario, nome: String, descricao: String)
    +adicionarMembro(usuario: Usuario, nomeComunidade: String)
```

```
    }

    class RelationshipManager {
        +adicionarAmigo(usuario: Usuario, loginAmigo: String)
        +ehAmigo(login1: String, login2: String): boolean
    }

    class PersistenceManager {
        +carregarUsuarios(userManager: UserManager)
        +salvarDados(userManager: UserManager, communityManager:
CommunityManager)
    }

    %% Relationships
    Facade --> UserManager
    Facade --> SessionManager
    Facade --> CommunityManager
    Facade --> RelationshipManager
    Facade --> PersistenceManager

    UserManager --> Usuario: manages
    CommunityManager --> Comunidade: manages

    Usuario "1" *-- "1" Perfil: has
    Usuario "1" --> "*" Usuario: friends
    Usuario "1" --> "*" Usuario: idols
    Usuario "1" --> "*" Usuario: crushes
    Usuario "1" --> "*" Usuario: enemies

    Comunidade "1" --> "1" Usuario: owner
    Comunidade "1" --> "*" Usuario: members
```



4. CONCLUSÃO

O desenvolvimento do Jackut foi uma experiência valiosa para aplicar conceitos de programação orientada a objetos, padrões de design e boas práticas de codificação. O sistema atende aos requisitos propostos, com funcionalidades completas para gerenciamento de usuários, relacionamentos e comunidades.

4.1 AVALIAÇÃO OBJETIVA

- Qualidade da Documentação: 8,5 (JavaDoc presente, mas alguns métodos poderiam ser mais detalhados).
- Qualidade do Design: 9,0 (Arquitetura modular e baixo acoplamento).
- Qualidade do Código: 9,0 (Código limpo, mas com oportunidades para reduzir redundâncias).

Assinatura:

Carla Evellyn Ferreira dos Santos || 05/05/2025