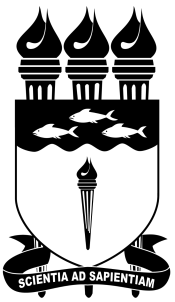
UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO



UNIVERSIDADE FEDERAL
DE ALAGOAS

IURY KAUANN DAVID NOGUEIRA

Relatorio Milestone 2 - Projeto: Rede de Relacionamentos Jackut

POO / JAVA

Maceió/AL

2025

IURY KAUANN DAVID NOGUEIRA

Relatorio Milestone 2 - Projeto: Rede de Relacionamentos Jackut

POO / JAVA

Sistema Jackut - Rede Social

Apresentado no Instituto de Computação da Universidade Federal de Alagoas, Campus A. C. Simões, como parte dos requisitos para obtenção de nota na disciplina de Programação 2 (P2) ou Programação Orientada a Objetos (POO).

Orientador: Prof. Dr. Mario Hozano Lucas de Souza

Maceió/AL

2025

SUMÁRIO

1.INTRODUÇÃO

Jackut é uma rede social inspirada em modelos clássicos de relacionamentos online, desenvolvida para demonstrar a aplicação de princípios de design de software e arquitetura modular. Este documento detalha a implementação das User Stories 5 a 9, correspondentes ao **segundo** milestone do projeto, que abrange funcionalidades extras em relacionamentos, comunidades e outros.

1.1 Objetivos

A segunda etapa do projeto teve como objetivo principal a **evolução da implementação desenvolvida na Milestone 1**, incorporando as **críticas, sugestões e indicações recebidas durante a avaliação anterior**. Além disso, foram implementadas novas funcionalidades para atender às User Stories 5.1 a 9.2, conforme documento de requisitos em anexo.

1.2 Escopo do Milestone 2

- **US5 –** Relacionamentos complexos (ídolos, fãs, paqueras, inimigos)
- **US6 –** Gestão de comunidades (criação, associação de membros)
- **US7 –** Envio de mensagens em massa para comunidades
- **US8 –** Leitura de mensagens e recados com controle de fila
- **US9 –** Persistência completa do sistema (estado salvo entre execuções)

A Validação de persistência foi testada em cada UX.2, considerando os arquivos de teste exemplo (us1_1.txt) e (us1_2 txt).

1.3 Metodologia

Foi adotada uma abordagem baseada em **Test-Driven Development (TDD)**, utilizando a biblioteca **EasyAccept** para validar o comportamento do sistema frente a cenários de teste pré-definidos.

A arquitetura foi projetada com base em princípios de engenharia de software que favorecem a **separação de responsabilidades**, resultando em uma organização modular do código:

- **Jackut**: núcleo da lógica de negócio e coordenação geral
- **Facade**: interface de comunicação com os testes
- **Exceptions**: pacote de exceções personalizadas para controle de erros
- **Models**: entidades principais do sistema, como `Usuario` e `Comunidade`
- **Services (Gerenciadores)**: componentes especializados na gestão de usuários, comunidades e relacionamentos

2.Estrutura do Projeto

2.1 Hierarquia de Arquivos

```
├── src/
│   └── br.ufal.ic.p2.jackut/
│       ├── exceptions/
│       │   ├── AmigoJaAdicionadoException.java
│       │   ├── AtributoNaoPreenchidoException.java
│       │   ├── AutoAmizadeException.java
│       │   ├── AutoMensagemException.java
│       │   ├── AutoRelacaoException.java
│       │   ├── ComunidadeJaExisteException.java
│       │   ├── ComunidadeNaoEncontradaException.java
│       │   ├── ConviteNaoEncontradoException.java
│       │   ├── InimigoException.java
│       │   ├── LoginInvalidoException.java
```

```
│   │   ├── LoginOuSenhaInvalidosException.java
│   │   ├── PaqueraExistenteException.java
│   │   ├── PersistenciaException.java
│   │   ├── RelacaoExistenteException.java
│   │   ├── SemMensagensException.java
│   │   ├── SemRecadosException.java
│   │   ├── SenhaInvalidaException.java
│   │   ├── UsuarioJaExisteException.java
│   │   └── UsuarioNaoEncontradoException.java
│   ├── models/
│   │   ├── Community.java
│   │   ├── package-info.java
│   │   ├── Perfil.java
│   │   ├── Recado.java
│   │   └── Usuario.java
│   ├── services/
│   │   ├── GerenciadorComunidades.java
│   │   ├── GerenciadorSessoes.java
│   │   ├── GerenciadorUsuarios.java
│   │   └── package-info.java
│   ├── Facade.java
│   ├── Jackute.java
│   └── package-info.java
├── Main/
```

Novas Exceções :

- AutoRelacaoException.java: Relação com o próprio usuário não é permitida.
- ComunidadeJaExisteException.java: Tentativa de criar comunidade já existente.
- ComunidadeNaoEncontradaException.java: Comunidade não localizada no sistema.

- InimigoException.java: Interação não permitida com usuário marcado como inimigo.
- PaqueraExistenteException.java: Relação de paquera já registrada.
- RelacaoExistenteException.java: Relação entre os usuários já existe.
- SemMensagensException.java: Nenhuma mensagem disponível para exibição.

Novos Serviços :

- GerenciadorComunidades.java: Gerencia criação, exclusão e listagem de comunidades.

Novo Model :

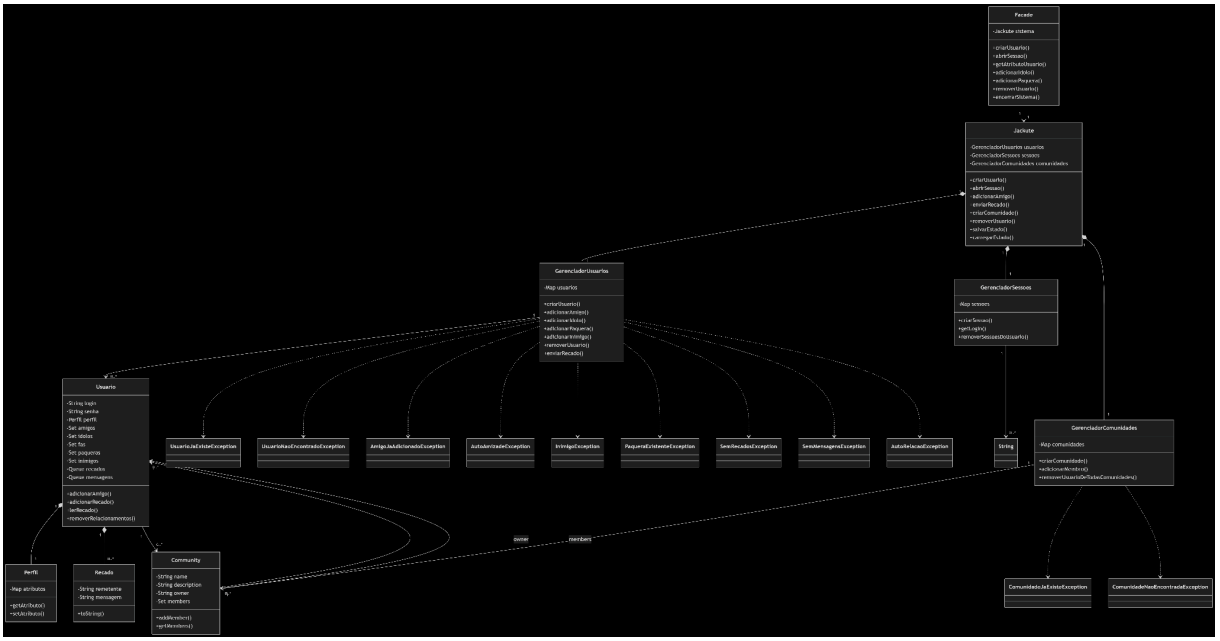
- Community.java: Representa uma comunidade com nome, descrição e lista de membros.

Atualizações :

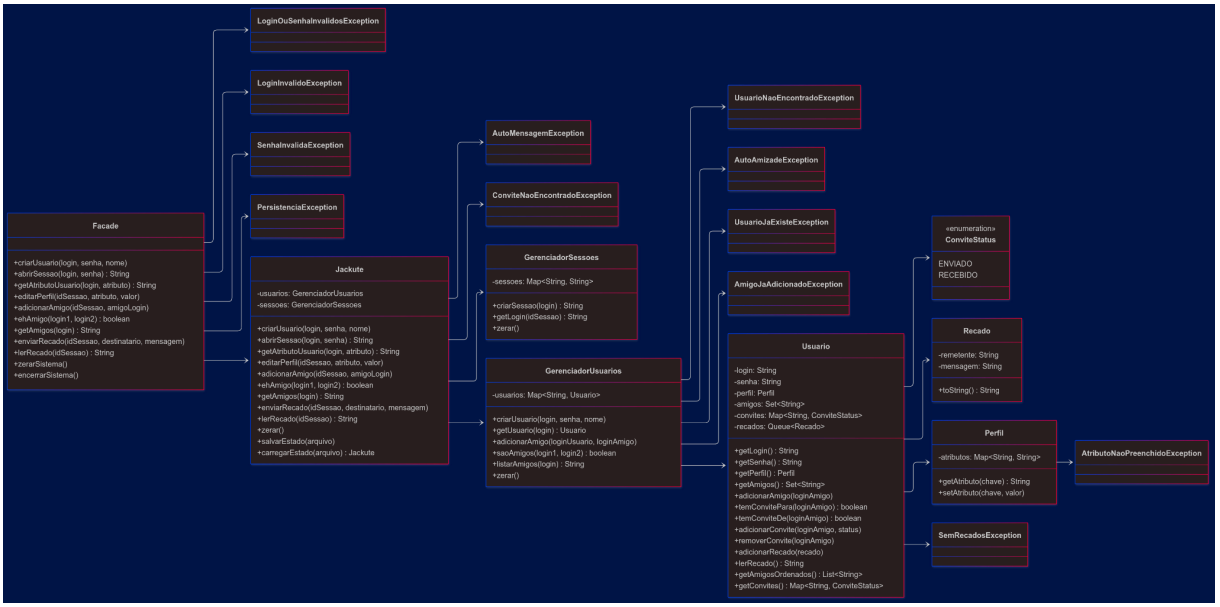
- Usuario.java: Novos campos para representar relações (amizade, paquera, inimizade).
- GerenciadorUsuarios.java: Novos métodos para gerenciamento de relações entre usuários.
- Jackute.java: Integração com serviços de comunidades e novos relacionamentos.

2.2 Diagrama de Classes

- Milestone 2



(Imagem com melhor qualidade está na pasta raiz do projeto e no README.md / Milestone-2)



(Imagem também está na pasta raiz no projeto e no README.md / Milestone-1)

2.3 Padrões de Projeto Aplicados

- **2.3.1 Singleton (Gerenciadores) :**

Os gerenciadores (`GerenciadorUsuarios`, `GerenciadorSessoes`, `GerenciadorComunidades`) seguem o padrão **Singleton** de fato, pois são instanciados apenas uma vez dentro da classe `Jackute`. Dessa forma, garantem:

Consistência: Uma única fonte de verdade para usuários, sessões e comunidades.

Controle de Estado: Simplificam a serialização e desserialização global do sistema.

- **2.3.2 Facade (Camada de Interface) :**

A classe `Facade` implementa o padrão **Facade**, provendo uma interface simples para todas as operações do Jackut:

Abstração de Complexidade: Oculta detalhes de instâncias de gerenciadores, modelos e persistência.

Interface Unificada: Métodos como `criarComunidade()`, `adicionarPaquera()` e `removerUsuario()` combinam várias etapas internas em chamadas únicas.

- **2.3.3 Observer (Notificações Automáticas) :**

Empregado indiretamente no fluxo de paqueras mútuas:

Mecanismo: Ao confirmar que A e B são paqueras, chama-se `enviarRecadoSistema()`, que dispara recados para ambos.

Vantagem: Desacopla a lógica de notificação da regra de negócio principal, facilitando adição de novos tipos de eventos.

- **2.3.4 Strategy (Validações Dinâmicas)**

As validações de relacionamentos (amizade, paquera, ídolo, inimigo) usam uma abordagem de **Strategy**:

Políticas Reutilizáveis: Cada checagem auto-relacionamento, inimizade, convites pendentes pode ser extraída para componentes independentes.

Extensibilidade: Novas regras (por exemplo, bloqueios temporários) podem ser introduzidas sem alterar métodos existentes.

- **2.3.5 Factory Method (Criação de Objetos Complexos)**

A classe `Usuario` atua como uma fábrica implícita para criar seu estado interno:

Inicialização Controlada: Construtor de `Usuario` cria `Perfil` e inicializa coleções para amigos, convites, recados etc.

Encapsulamento de Lógica: `GerenciadorUsuarios.criarUsuario()` encapsula toda a lógica de validação e construção.

Exemplo:

```
public void criarUsuario(String login, String senha, String nome) {
   validarCredenciais(login, senha);
   usuarios.put(login, new Usuario(login, senha, nome));
}
```

3.DESIGN DE CLASSES

(O JAVADOC FOI GERADO NA RAIZ DO PROJETO PARA ANÁLISE DE DETALHES ESPECIFICOS)

3.1 Jackute: Coordenação Expandida

A classe principal Jackute teve sua função de orquestração central ampliada, passando a gerenciar os novos módulos do sistema de forma coesa e eficiente.

Principais alterações:

- **Gestão de Comunidades:** Integração com a nova classe GerenciadorComunidades, responsável por criar e remover comunidades, além de permitir envio de mensagens em massa.

- **Persistência Aprimorada:** Inclusão de serialização de relacionamentos adicionais (fãs, paqueras, inimigos) e comunidades, utilizando LinkedHashSet para preservar a ordem de inserção.

- **Remoção em Cascata:** O método removerUsuario() agora remove dados associados em cascata (comunidades, relacionamentos, sessões), garantindo a integridade do sistema.

Escolhas de design:

- **Separação de Responsabilidades:** Cada funcionalidade foi delegada a gerenciadores específicos, promovendo coesão.

- **Persistência Atômica:** O estado completo do sistema é salvo e restaurado a partir de um único arquivo (dados_jackut.dat), simplificando o processo de persistência.

3.2 GerenciadorUsuarios: Relacionamentos Complexos

O GerenciadorUsuarios foi expandido para lidar com múltiplos tipos de relacionamentos, além das amizades convencionais.

Novas funcionalidades:

- **Relacionamentos Hierárquicos:**

    - *Fãs/Ídolos:* Utilização de HashSet para garantir buscas eficientes.

    - *Paqueras:* Utilização de LinkedHashSet para preservar ordem de inserção e permitir notificações mútuas.

    - *Inimigos:* Implementação de bloqueio automático de interações, com verificações em métodos críticos.

Otimizações aplicadas:

- **Desempenho:** Uso de HashMap garante complexidade O(1) para acesso a usuários.

- **Consistência:** Remoção em cascata de usuários de todas as estruturas relacionadas.

3.3 GerenciadorComunidades: Gestão de Grupos

A nova classe GerenciadorComunidades foi criada para lidar exclusivamente com comunidades, promovendo escalabilidade e organização do código.

Características principais:

- **Estrutura de Dados:** Uso de HashMap<String, Community> para acesso rápido às comunidades por nome.

- **Membros Ordenados:** Cada Community utiliza LinkedHashSet para preservar a ordem de entrada dos membros.

- **Persistência de Donos:** O dono da comunidade é armazenado separadamente para garantir restauração correta após a desserialização.

Métodos relevantes:

- enviarMensagem(): Distribui mensagens a todos os membros de forma otimizada.

- removerUsuarioDeTodasComunidades(): Remove o usuário de todas as comunidades e apaga comunidades das quais era dono.

3.4 Usuario: Novos Atributos e Relacionamentos

A classe Usuario foi estendida com novos campos para representar os diversos tipos de relacionamentos sociais.

Novos atributos:

```
private Set<String> fas = new HashSet<>();
private Set<String> idolos = new HashSet<>();
private Set<String> paqueras = new LinkedHashSet<>();
private Set<String> inimigos = new HashSet<>();
```

Decisões técnicas:

- **Segurança:** Campos críticos como login e senha permanecem final, garantindo imutabilidade.

- **Organização de Mensagens:** Separação entre filas de *recados* (públicos) e *mensagens* (privadas/comunitárias), garantindo clareza no escopo da comunicação.

3.5 Exceções Personalizadas

Foi criado um conjunto de exceções específicas para lidar com os novos cenários introduzidos nas User Stories 5 a 9.

Novas exceções:

- ComunidadeJaExisteException: Evita duplicação de comunidades.
- UsuarioJaMembroException: Impede adição redundante de usuários a comunidades.
- InimigoException: Bloqueia interações com usuários marcados como inimigos.
- AutoRelacaoException: Impede que um usuário se relacione consigo mesmo.

Estratégia adotada:

- **Herança de RuntimeException:** As exceções herdam de RuntimeException, evitando a obrigatoriedade de tratamentos com try-catch.
- **Mensagens Contextualizadas:** As mensagens fornecem contexto claro, como por exemplo: *"Função inválida: [Nome] é seu inimigo."*

3.6 Persistência e Serialização

A persistência do sistema foi fortalecida para garantir a continuidade do funcionamento após reinicializações ou falhas.

Melhorias implementadas:

- **Compatibilidade com Versões:** O método readObject em Jackute foi ajustado para inicializar campos ausentes em versões anteriores.
- **Tolerância a Falhas:** O sistema é recriado automaticamente se o arquivo de dados estiver corrompido ou ausente (FileNotFoundException).

Fluxo de persistência:

- **Salvar:** O objeto Jackute serializa todos os gerenciadores e dados no arquivo dados_jackut.dat.
- **Carregar:** A desserialização reconstrói toda a hierarquia de objetos do sistema.

4. Conclusão da Arquitetura

As decisões arquiteturais desta etapa refletem um compromisso com a evolução sustentável do sistema. Os principais objetivos foram atingidos:

- **Extensibilidade:** Novas funcionalidades foram adicionadas de forma modular, sem comprometer os componentes existentes.

- **Eficiência:** Adoção de estruturas de dados adequadas (como HashMap e LinkedHashSet) garante desempenho mesmo com grandes volumes de dados.

- **Robustez:** O uso de exceções personalizadas, persistência confiável e validações rigorosas assegura consistência e estabilidade da aplicação.

Essas escolhas demonstram a aplicação de princípios de engenharia de software, como SOLID, além do uso consciente de padrões como Fachada e Gerenciadores, viabilizando a escalabilidade e manutenção contínua do Jackut.