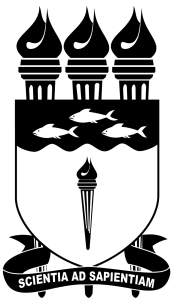
UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO



UNIVERSIDADE FEDERAL
DE ALAGOAS

IURY KAUANN DAVID NOGUEIRA

Relatorio Milestone 1 - Projeto: Rede de Relacionamentos Jackut

POO / JAVA

Maceió/AL

2025

IURY KAUANN DAVID NOGUEIRA

Relatorio Milestone 1 - 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 da AB1 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 1 a 4, correspondentes ao primeiro milestone do projeto, que abrange funcionalidades essenciais como criação de contas, gerenciamento de perfis, sistema de amizades e troca de recados.

1.1 Objetivos do Projeto

- Implementar um sistema robusto e escalável para gerenciar usuários e suas interações.
- Garantir aderência total aos requisitos funcionais por meio de testes de aceitação automatizados.
- Demonstrar boas práticas de arquitetura de software, como separação de camadas e encapsulamento.

1.2 Escopo do Milestone 1

- **Criação de Contas (US1)**
- **Edição de Perfis (US2)**
- **Sistema de Amizades (US3)**
- **Troca de Recados (US4)**

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

O desenvolvimento seguiu uma abordagem orientada a testes (*TDD*), utilizando a biblioteca EasyAccept para validar cada funcionalidade contra cenários pré-definidos. A arquitetura foi projetada para:

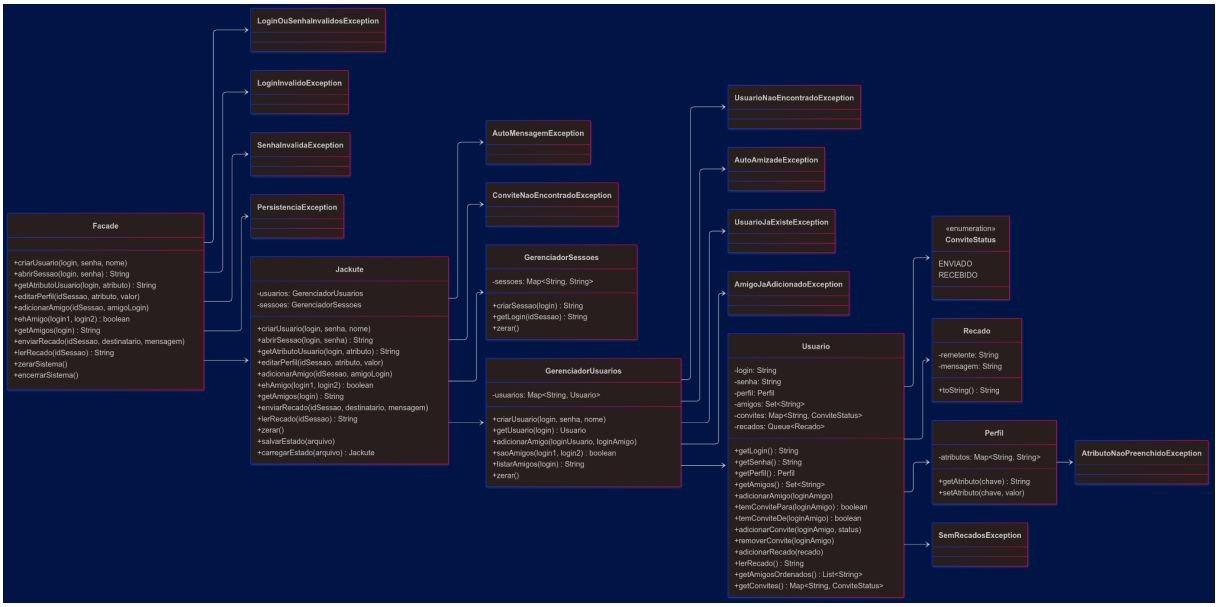
- **Separar preocupações:** Divisão clara entre l**ógica de negócio (*Jackut*)**, **interface (*Facade*)**, **expceptions**, **models**, **services**.

2.Estrutura do Projeto

2.1 Hierarquia de Arquivos

```
├── src/
│   └── br.ufal.ic.p2.jackut/
│       ├── exceptions/
│       │   ├── AmigoJaAdicionadoException.java
│       │   ├── AtributoNaoPreenchidoException.java
│       │   ├── AutoAmizadeException.java
│       │   ├── AutoMensagemException.java
│       │   ├── ConviteNaoEncontradoException.java
│       │   ├── LoginInvalidoException.java
│       │   ├── LoginOuSenhaInvalidosException.java
│       │   ├── PersistenciaException.java
│       │   ├── SemRecadosException.java
│       │   ├── SenhaInvalidaException.java
│       │   ├── UsuarioJaExisteException.java
│       │   └── UsuarioNaoEncontradoException.java
│       ├── models/
│       │   ├── package-info.java
│       │   ├── Perfil.java
│       │   ├── Recado.java
│       │   └── Usuario.java
│       ├── services/
│       │   ├── GerenciadorSessoes.java
│       │   ├── GerenciadorUsuarios.java
│       │   └── package-info.java
│       ├── Facade.java
│       ├── Jackute.java
│       └── package-info.java
├── Main/
```

2.2 Diagrama de Classes



(Imagem também está na pasta raiz no projeto e no README.md)

2.3 Padrões de Projeto Aplicados

O projeto **Jackut** adota padrões de design de software para garantir modularidade, reutilização de código e separação de responsabilidades. O principal padrão aplicado é :

Facade :

O padrão **Facade** foi utilizado para fornecer uma interface simplificada e unificada para o núcleo do sistema. A classe **(Facade.java)** expõe métodos de alto nível que encapsulam a lógica interna do sistema, permitindo que os clientes interajam com funcionalidades complexas sem precisar conhecer os detalhes da implementação.

- **Motivação**: Facilita a integração com outras partes do sistema, reduzindo o acoplamento entre a interface do usuário e a lógica de negócios.

- **Benefícios**: Oculta a complexidade da implementação interna, fornece um ponto central de acesso às funcionalidades do sistema e melhora a testabilidade ao permitir a substituição de componentes internos.

3.DESIGN DE CLASSES

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

3.1 Facade

A classe Facade, como já abordada no topico anterior, mas reforçando, serve como a interface unificada do sistema, simplificando a interação do usuário com as funcionalidades complexas internas. Ela encapsula as chamadas à lógica de negócio, gerenciando a comunicação com **o núcleo do sistema (Jackute)** e realizando operações de persistência.

- **Encapsulamento**

  Ao expor métodos de alto nível (como criarUsuario, abrirSessao, getAtributoUsuario, etc.), a Facade oculta detalhes internos, permitindo que o cliente não precise conhecer as implementações dos gerenciadores ou do sistema de persistência.

- **Persistência:**

  A classe possui métodos auxiliares (`salvarDados` e `carregarDados`) que realizam a serialização e desserialização do estado do sistema, garantindo que os dados sejam preservados entre execuções. Essa abordagem reforça a robustez do sistema.

3.2 Jackute

A classe **Jackute** concentra a lógica central do sistema, coordenando as operações de criação de usuários, gerenciamento de sessões, envio de recados e relacionamento entre usuários.

- **Centralização da Lógica de Negócio:**

  Ao concentrar as operações principais, a classe Jackute delega tarefas específicas para os gerenciadores (usuários e sessões), facilitando a manutenção e evolução do código.

- **Persistência e Serialização:**

  Implementando a interface `Serializable`, Jackute permite que o estado total do sistema seja salvo e restaurado, o que é fundamental para a continuidade do serviço após reinicializações.

- **Validações e Tratamento de Erros:**

  Métodos como `criarUsuario` e `abrirSessao` incluem validações (como verificar se o login e senha são válidos) e lançam exceções apropriadas, garantindo a integridade dos dados.

- **Delegação:**

  A existência de métodos auxiliares (por exemplo, `getUsuarioPorSessao`) demonstra a prática de delegar responsabilidades e evitar duplicação de código.

3.3 GerenciadorUsuarios

A classe **GerenciadorUsuarios** é responsável por gerenciar todas as operações relacionadas aos usuários, desde a criação e armazenamento até o gerenciamento de convites e amizades. Ela utiliza um `HashMap` para mapear logins a objetos do tipo **Usuario**.

Eficiência e Justificativas:

- **Acesso Rápido com HashMap:** O uso de `HashMap` permite buscas e inserções em tempo constante (O(1)), ideal para sistemas com alta demanda e necessidade de resposta rápida.

3.4 GerenciadorSessoes

A classe **GerenciadorSessoes** gerencia a criação, armazenamento e validação das sessões de usuários. Ela utiliza um `HashMap` para mapear os IDs de sessão aos logins correspondentes.

- **Geração de IDs Únicos:** A utilização de `UUID.randomUUID()` assegura que cada sessão possua um identificador único, minimizando a possibilidade de colisões.

- **Acesso Eficiente:** A estrutura de `HashMap` proporciona acesso rápido e direto aos dados das sessões, essencial para validações de segurança e gerenciamento dinâmico das sessões ativas.

- **Facilidade na Reinicialização:** O método `zerar` permite limpar todas as sessões com simplicidade, o que facilita testes e reinicializações sem comprometer a consistência do sistema.

3.5 Usuario

A classe **Usuario** representa os usuários do sistema, encapsulando informações como perfil, convites, amigos e recados. Ela foi projetada para garantir imutabilidade dos dados sensíveis e eficiência no gerenciamento dos relacionamentos.

- **Imutabilidade de Dados Críticos:** Os atributos `login` e `senha` são definidos como finais, garantindo que informações fundamentais não sejam alteradas após a criação, aumentando a segurança do sistema.

- **Estrutura de Dados Otimizada:**

    - **LinkedHashSet para Amigos:** Utilizado para evitar duplicatas e preservar a ordem de inserção, o que é importante para a apresentação e histórico dos relacionamentos.

    - **HashMap para Convites:** Permite gerenciar convites de amizade de forma rápida e organizada, distinguindo entre convites enviados e recebidos por meio de uma enumeração.

    - **Queue para Recados:** Garante que os recados sejam processados na ordem correta, respeitando o fluxo natural de mensagens.

3.6 Perfil

A classe **Perfil** gerencia os atributos pessoais dos usuários de maneira dinâmica, permitindo a personalização do perfil sem a necessidade de alterações na estrutura da classe.

- **Flexibilidade com Map:** O uso de um `Map<String, String>` possibilita a adição e atualização de atributos de forma dinâmica, facilitando a extensão do perfil conforme novas necessidades surgem.

- **Validação de Atributos:** O método `getAtributo` implementa uma verificação que garante a existência da chave, lançando uma exceção caso o atributo não esteja definido, o que aumenta a segurança e a integridade dos dados.

3.7 Recado

A classe **Recado** representa uma mensagem enviada entre usuários, armazenando o remetente e o conteúdo da mensagem. Ela foi mantida simples para cumprir seu propósito com eficiência.

- **Simplicidade e Foco:** Ao conter apenas os atributos essenciais (remetente e mensagem) e implementar o método `toString` para retornar o conteúdo, a classe evita complexidade desnecessária, facilitando seu uso e manutenção.

- **Integração com Persistência:** A implementação de `Serializable` permite que os recados sejam persistidos juntamente com o estado do sistema, garantindo que a troca de mensagens seja mantida mesmo após reinicializações.

3.8 - Exceptions

Para garantir a robustez do sistema Jackut, foram implementadas diversas exceções personalizadas, centralizadas no pacote `br.ufal.ic.p2.jackut.exceptions`. Essas exceções estendem `RuntimeException`, permitindo um tratamento de erros mais limpo e coeso, sem a necessidade de verificações excessivas em tempo de compilação.