UNIVERSIDADE FEDERAL DE ALAGOAS –
UFAL CAMPUS A. C. SIMÕES

CURSO DE CIÊNCIA DA COMPUTAÇÃO 2024.2

RIAN ANTONIO DA SILVA GAIÃO

PROGRAMAÇÃO ORIENTADA A OBJETOS:PROJETO JACKUT

MILESTONE 1

ABRIL, 2025

UNIVERSIDADE FEDERAL DE ALAGOAS –
UFAL CAMPUS A. C. SIMÕES

CURSO DE CIÊNCIA DA COMPUTAÇÃO 2024.2

RIAN ANTONIO DA SILVA GAIÃO

Relatório referente ao experimento: Ex.: Um projeto de programação orientada a objetos , solicitado pelo professor Mário Horanzo. De cunho avaliativo para composição da média final.

PROGRAMAÇÃO ORIENTADA A OBJETOS:PROJETO JACKUT

MILESTONE 1.

RESUMO

O projeto Jackut consiste em uma rede de relacionamentos inspirada no Orkut, desenvolvida em Java e seguindo princípios de POO. O sistema permite cadastro de usuários, gestão de perfis, amizades mútuas, envio de mensagens e participação em comunidades. Foi implementado uma arquitetura modular com facade para integração com testes automatizados (EasyAccept), garantindo 100% de aprovação nos testes de aceitação. Usando estrutura de dados e design de projetos orientados a objetos para maior eficiência e elegância do serviço. Resultados validaram a robustez do sistema em operações básicas e complexas.

Palavras chave: programação orientada a objetos, java, relatório, projeto

INTRODUÇÃO

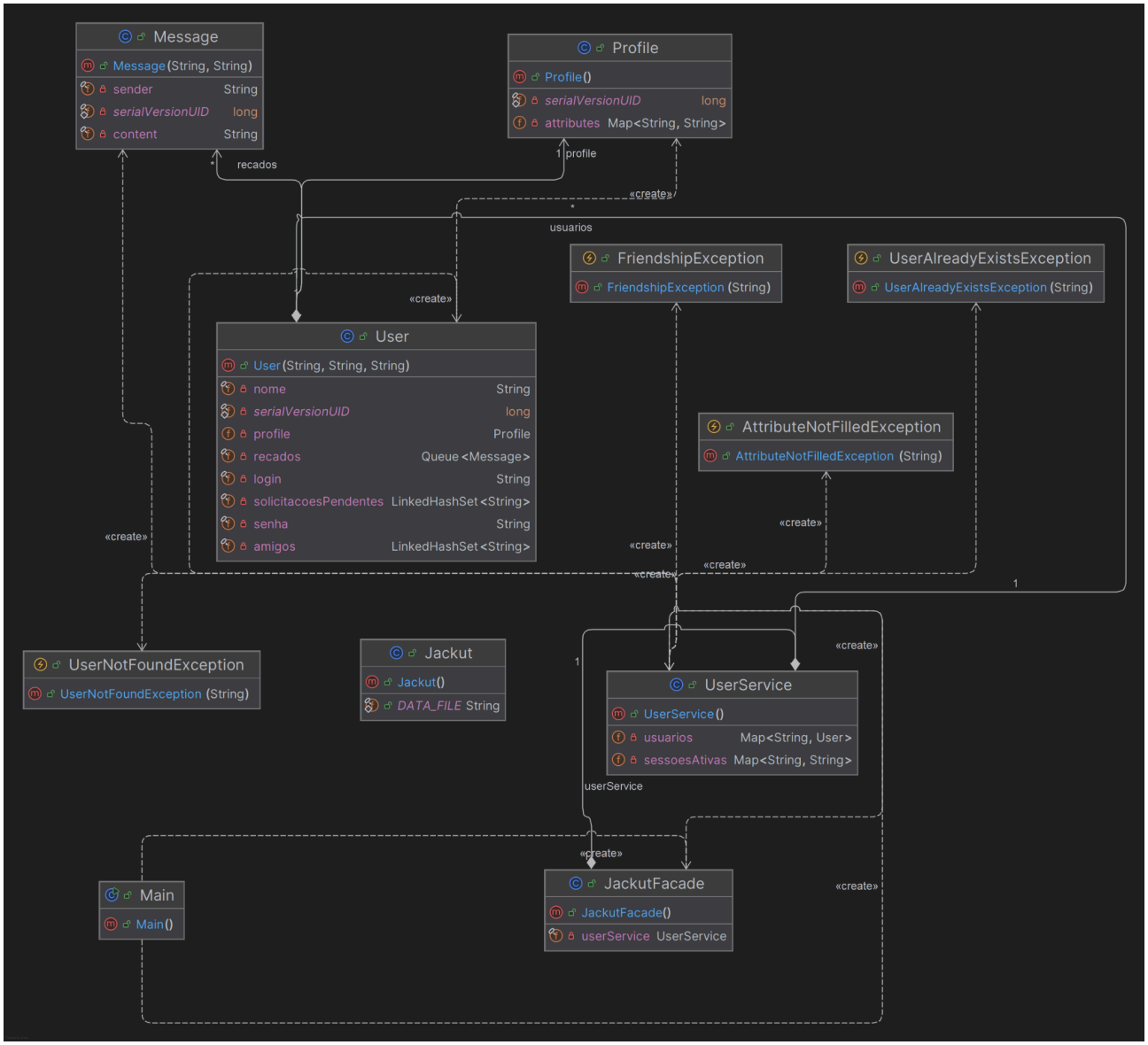
Este documento descreve o Milestone 1 do projeto Jackut, uma rede de relacionamentos desenvolvida em Java como parte de um trabalho acadêmico em POO (Programação Orientada a Objetos). O projeto será entregue em etapas (milestones), cada uma incorporando novas funcionalidades definidas em User Stories providenciados pelo orientador do projeto. Neste papel há breve elaborações das entidades escritas/adicionadas nesta milestone, focando mais no processo de decisões de design e diagrama de classes UML.

A milestone 1 do projeto Jackut cobre os user stories: US1 até o US4. (esses arquivos são achados no projeto, junto com um javadoc com descrições de cada função e classe relevante.)

Link do repositório no github: https://github.com/ryanthecomic/Projeto-Jackut

VISÃO GERAL

DIAGRAMA UML



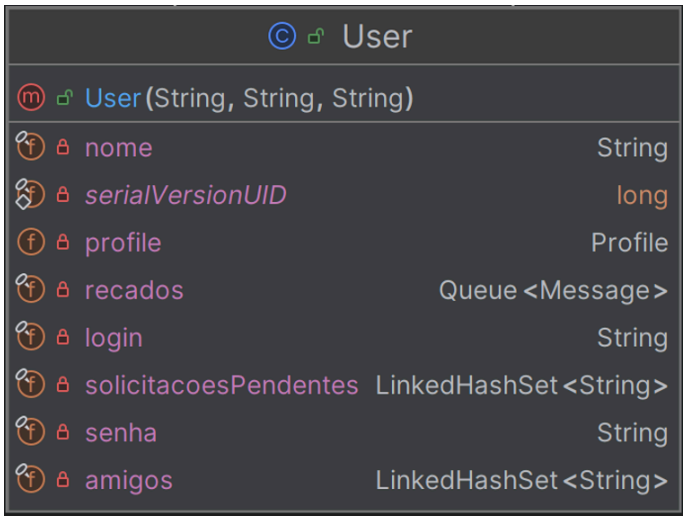
(Diagrama UML gerado a partir da ferramenta diagramas do Intellij, no centro do diagrama está a entidade User.)

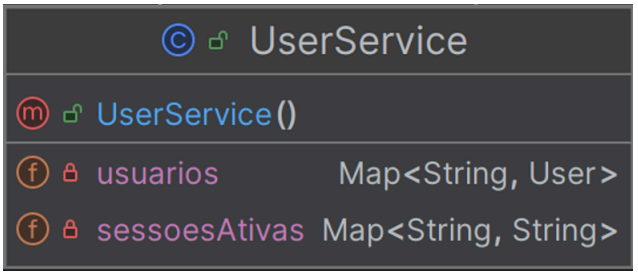
Função Main

A função Main gera uma interface simples que foi essencial para o desenvolvimento contínuo do projeto, como estamos na etapa de milestone 1, achei apropriado deixar na entrega da primeira milestone para testes mais simples dos comandos do easyaccept.jar

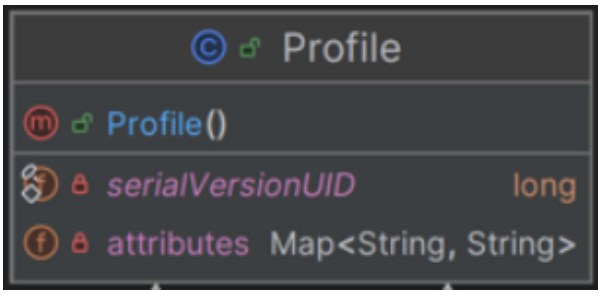
SEPARAÇÃO DE RESPONSABILIDADES

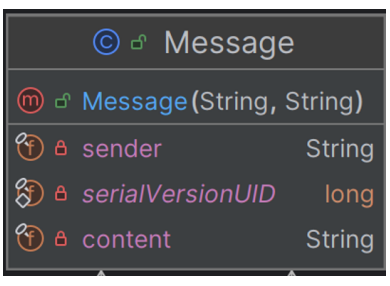
Como pode-se ver acima, a entidade User é o centro do programa, faz sentido por ser um serviço de rede inspirado no Orkut, onde usuários interagem entre si o tempo todo. Então foi decidido que User precisaria de um serviço juntamente com a sua entidade, para modularizar as várias partes do projeto que gerenciaria as interações do usuário com o sistema, essa decisão pode ser alterada em futuras milestones, uma vez que a maioria dos serviços como sessões, mensagens e edição de perfis são relacionadas a usuários, porém poderiam ser separados para melhor eficiência, mas já que todos os user stories providos nessa milestone foram relacionados ao usuário, acabou que o único arquivo de serviço foi o UserService..

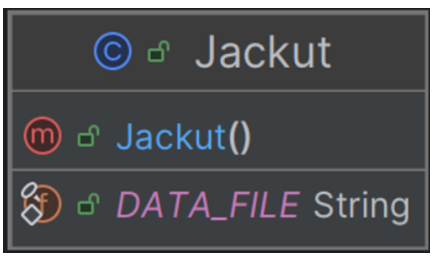




As entidades Jackut, Message, Profile, e User também foram criadas para melhor estruturar o sistema de forma modular, seguindo os princípios de Orientação a Objetos (POO) e Separação de Responsabilidades. Jackut em específico tem potêncial de ter o próprio serviço em futuras milestones, quando funções forem adicionadas para gerir o site invés de usuários, mas até agora a entidade só salva arquivos.







DECISÕES, SERIALIZAÇÃO E ESTRUTURAS

Já que os dados do usuário precisarão ser salvos, eles precisam ser serializados, o mesmo vale para todos seus campos, nome, login, senha, e claro a classe profile.

Motivo de usar mapa dinâmico ao invés de formar varios atributos na entidade Profile.java:

```
package entities;

public class Profile {

   private String descricao;
   private String estadoCivil;
   private String aniversario;
   private String filhos[];
   private String idiomas[];
   private String cidadeNatal;
   private String estilo;
   private String fumo;
   private String bebo;
   private String moro;

     // esse formato não permitiria preencher qualquer atributo
     // ou modificar um atributo inexistente e iria requerer
     // cada atributo de maneira fixa, o nosso usuário pode
     // preencher o que quiser e o que não quiser.

}
```

O uso de estruturas chave-valor (ou mapas) se destaca foi escolhido pois representava um banco de dados ideal, qualquer oportunidade apropriada para usar um mapa para armazenar usuários, tornando cada busca por esses atributos O(1) e sendo trivialmente fácil de implementar em Java.

Fluxo de Serialização:

```
1. Dados são salvos em jackut_data.ser via ObjectOutputStream
2. Recuperados no startup com o construtor do UserService
3. Backup automático em caso de corrupção
```

Uso de atributos Final:

O uso de final em atributos da classe User foi uma decisão baseada no princípios de imutabilidade controlada, os atributos marcados como final garantem que valores críticos não sejam alterados após a inicialização, prevenindo bugs encontrados no meio do desenvolvimento, isso pode ser mudado se for decidido se certos atributos como nome e senha poderão ser mudados pelo usuário a partir de autenticação.

```java
public class User implements Serializable {

   private static final long serialVersionUID = 1L;

   private final String login;

   private final String senha;

   private final String nome;
```

Usar esse método também assegura integridade de um sistema real de troca de mensagens.

Estruturas

As estruturas de dados selecionadas no projeto Jackut foram cuidadosamente escolhidas para garantir respostas precisas aos testes dos User Stories.

LinkedHashSet

```
private final LinkedHashSet<String> amigos = new LinkedHashSet<>();

private final LinkedHashSet<String> solicitacoesPendentes = new LinkedHashSet<>();
```

LinkedHashSets são usados para manter a ordem de amigos adicionados, para não interferir na ordem de resposta nas funções pedidas pelo Jackut e escalar dinamicamente (O primeiro amigo aceitado é o primeiro adicionado ao banco de dados.

Exemplo encontrado: O teste us3_1.txt exige que getAmigos retorne {oabath,jdoe} (nesta ordem). O LinkedHashSet garante isso preservando a sequência em que confirmarAmizade() foi chamado, enquanto um HashSet comum falharia por perder a ordem.

Lista encadeada

```
private final Queue<Message> recados = new LinkedList<>();
```

Pelo mesmo motivo, LinkedList é usado para filas dinâmicas com escalamento de dados e evitar quebra de ordem.