Universidade Federal de Alagoas
Instituto de Computação
Graduação em Ciência da Computação
Disciplina: Programação 2

Aluna: Isabelle de Lima Xavier

Relatório do Milestone 2

Maceió, 2 de Maio de 2025

1. Introdução

Este relatório descreve as decisões de projeto, a estrutura de classes e os padrões de projeto adotados na implementação do sistema Jackut. Desenvolvido como parte das atividades da disciplina de Programação 2, o sistema tem como objetivo simular o funcionamento de uma rede social educacional, contemplando funcionalidades como gerenciamento de usuários, envio de mensagens, criação de comunidades, adição de amigos e remoção completa de contas do sistema. O projeto foi concebido com base em princípios da programação orientada a objetos, priorizando a coesão entre as classes, o encapsulamento dos dados e a separação clara entre responsabilidades.

2. Objetivos do Projeto

O sistema Jackut foi desenvolvido com o propósito de permitir que usuários possam se cadastrar, autenticar-se por meio de sessões, interagir com outros usuários por meio de recados, participar de comunidades, estabelecer laços de amizade e, quando desejado, remover sua conta de maneira que todas as informações relacionadas a ela sejam definitivamente excluídas do sistema. Durante a implementação, buscou-se garantir que a arquitetura permanecesse escalável e de fácil manutenção.

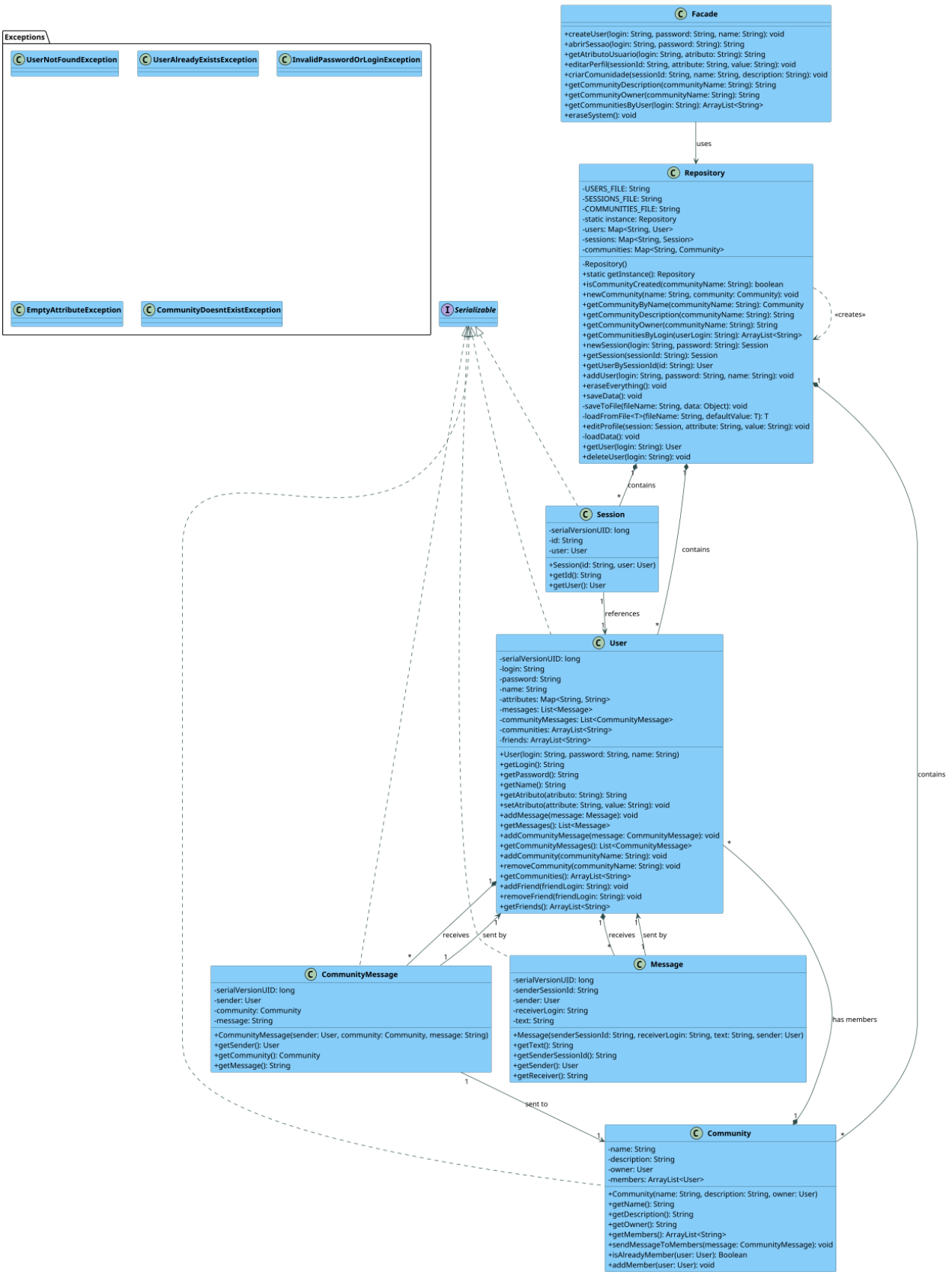
3. Estrutura do Projeto

A estrutura do sistema foi organizada em camadas bem definidas. A camada de fachada é representada pela classe **Jackut**, que expõe uma interface única e simplificada para os casos de uso do sistema, sendo o principal ponto de acesso para os testes e clientes externos. A lógica de armazenamento e recuperação de dados foi encapsulada na classe **Repository**, responsável por gerenciar usuários, sessões e comunidades. Essa classe centraliza o estado do sistema, garantindo consistência e desacoplamento entre os componentes. O núcleo do modelo de domínio é composto pelas classes **User**, **Community**, **Message**, **CommunityMessage** e **Session**, que representam entidades do mundo real e encapsulam tanto os dados quanto os comportamentos relacionados.

Além disso, o sistema conta com um conjunto de exceções específicas que definem mensagens de erro claras e precisas, contribuindo para o tratamento adequado de situações inválidas, como tentativas de envio de mensagens para si mesmo ou operações com usuários inexistentes.

4. Diagrama de Classes

O diagrama de classes do sistema representa visualmente a organização das entidades e suas relações. As principais classes são `Jackut`, `Repository`, `User`, `Community`, `Message`, `CommunityMessage`, `Session` e as exceções personalizadas. O relacionamento entre elas evidencia a separação entre interface, lógica de negócio e dados. O diagrama encontra-se anexado ao repositório do projeto, tal qual encontra-se abaixo, e pode ser consultado para melhor compreensão da estrutura interna.


Exceptions
UserNotFoundException
UserAlreadyExistsException
InvalidPasswordOrLoginException
EmptyAttributeException
CommunityDoesntExistException
Serializable
Facade
+createUser(login: String, password: String, name: String): void
+abrirSessao(login: String, password: String): String
+getAtributoUsuario(login: String, atributo: String): String
+editarPerfil(sessionId: String, attribute: String, value: String): void
+criarComunidade(sessionId: String, name: String, description: String): void
+getCommunityDescription(communityName: String): String
+getCommunityOwner(communityName: String): String
+getCommunitiesByUser(login: String): ArrayList<String>
+eraseSystem(): void
uses
Repository
-USERS_FILE: String
-SESSIONS_FILE: String
-COMMUNITIES_FILE: String
-static instance: Repository
-users: Map<String, User>
-sessions: Map<String, Session>
-communities: Map<String, Community>
-Repository()
+static getInstance(): Repository
+isCommunityCreated(communityName: String): boolean
+newCommunity(name: String, community: Community): void
+getCommunityByName(communityName: String): Community
+getCommunityDescription(communityName: String): String
+getCommunityOwner(communityName: String): String
+getCommunitiesByLogin(userLogin: String): ArrayList<String>
+newSession(login: String, password: String): Session
+getSession(sessionId: String): Session
+getUserBySessionId(id: String): User
+addUser(login: String, password: String, name: String): void
+eraseEverything(): void
+saveData(): void
-saveToFile(fileName: String, data: Object): void
-loadFromFile<T>(fileName: String, defaultValue: T): T
+editProfile(session: Session, attribute: String, value: String): void
-loadData(): void
+getUser(login: String): User
+deleteUser(login: String): void
«creates»
contains
Session
-serialVersionUID: long
-id: String
-user: User
+Session(id: String, user: User)
+getId(): String
+getUser(): User
contains
references
User
-serialVersionUID: long
-login: String
-password: String
-name: String
-attributes: Map<String, String>
-messages: List<Message>
-communityMessages: List<CommunityMessage>
-communities: ArrayList<String>
-friends: ArrayList<String>
+User(login: String, password: String, name: String)
+getLogin(): String
+getPassword(): String
+getName(): String
+getAtributo(atributo: String): String
+setAtributo(attribute: String, value: String): void
+addMessage(message: Message): void
+getMessages(): List<Message>
+addCommunityMessage(message: CommunityMessage): void
+getCommunityMessages(): List<CommunityMessage>
+addCommunity(communityName: String): void
+removeCommunity(communityName: String): void
+getCommunities(): ArrayList<String>
+addFriend(friendLogin: String): void
+removeFriend(friendLogin: String): void
+getFriends(): ArrayList<String>
receives
sent by
receives
sent by
CommunityMessage
-serialVersionUID: long
-sender: User
-community: Community
-message: String
+CommunityMessage(sender: User, community: Community, message: String)
+getSender(): User
+getCommunity(): Community
+getMessage(): String
Message
-serialVersionUID: long
-senderSessionId: String
-sender: User
-receiverLogin: String
-text: String
+Message(senderSessionId: String, receiverLogin: String, text: String, sender: User)
+getText(): String
+getSenderSessionId(): String
+getSender(): User
+getReceiver(): String
has members
sent to
Community
-name: String
-description: String
-owner: User
-members: ArrayList<User>
+Community(name: String, description: String, owner: User)
+getName(): String
+getDescription(): String
+getOwner(): String
+getMembers(): ArrayList<String>
+sendMessageToMembers(message: CommunityMessage): void
+isAlreadyMember(user: User): Boolean
+addMember(user: User): void
contains

5. Padrões de Projeto Utilizados

Durante a implementação do Jackut, foram aplicados alguns padrões de projeto com o intuito de melhorar a manutenibilidade e a clareza do código. O padrão *Facade* é representado pela classe `Jackut`, que concentra a exposição dos serviços do sistema para os clientes externos, escondendo a complexidade interna. Já o padrão *Repository* é utilizado na classe `Repository`, que centraliza a persistência dos dados e oferece métodos para manipulação das entidades. A própria instância do repositório segue o padrão *Singleton*, garantindo que uma única instância seja compartilhada por todo o sistema. Por fim, as entidades do domínio, como `User`, `Community` e `Message`, seguem o padrão *Domain Model*, encapsulando tanto os dados quanto os comportamentos, o que facilita a evolução do sistema sem quebra de contratos.

6. Justificativas das Decisões de Design

A adoção de uma arquitetura em camadas com divisão clara de responsabilidades foi fundamental para garantir que o sistema fosse extensível, testável e organizado. A fachada centralizada em `Jackut` reduz o acoplamento entre o cliente e as classes de domínio. A implementação do repositório como singleton assegura consistência na manipulação de dados compartilhados entre diferentes componentes. Além disso, a escolha por encapsular as listas de mensagens e comunidades com cópias (`new LinkedList<>(...)`) evita efeitos colaterais indesejados, protegendo o estado interno das entidades. A remoção de um usuário, por exemplo, aciona mecanismos internos que removem todas as associações, como mensagens recebidas e enviadas, participação em comunidades e laços de amizade, assegurando a integridade do sistema.

7. Considerações Finais

A construção do sistema Jackut permitiu a aplicação de importantes conceitos da orientação a objetos e o uso consciente de padrões de projeto. As escolhas realizadas visaram manter um código limpo, coeso e preparado para futuras extensões. O sistema apresenta uma base sólida para representar funcionalidades de redes sociais de forma didática e organizada. A aplicação dos padrões *Facade*, *Repository*, *Singleton* e *Domain Model* se mostrou adequada ao problema proposto, promovendo uma arquitetura compreensível e modular.