Plataforma de Soluções Colaborativas - Documentação do Projeto

Disciplina: Programação Orientada a Objetos
Professor(a): Talita Ribeiro
Data de Entrega: 11/06/2025
Data de Apresentação: 12/06/2025
Grupo: Guilherme Dias, Guilherme Pardelhas, Guilherme Resende

Sumário

1. Introdução e Objetivo

Este projeto tem como objetivo consolidar os conhecimentos adquiridos na disciplina de **Programação Orientada a Objetos** por meio da criação de uma aplicação prática dentro do tema **crowdsourcing**. Os alunos deverão aplicar os conceitos de orientação a objetos e realizar a persistência de dados em banco relacional.

1.1 Motivação

A motivação para o desenvolvimento desta plataforma colaborativa, inspirada em modelos como o **Stack Overflow**, reside na crescente necessidade de ambientes digitais onde usuários podem compartilhar conhecimento, colaborar em projetos e buscar ou oferecer soluções para desafios específicos. O foco em crowdsourcing permite a criação de um sistema dinâmico e auto-sustentável, impulsionado pela contribuição da comunidade.

1.2 Requisitos Principais

- Aplicação dos **pilares de OO**.
- Pelo menos uma **classe abstrata** com métodos abstratos e concretos.
- Pelo menos uma **interface** implementada por duas ou mais classes.
- Diversidade de **cardinalidades**: 1:1, 1:N, N:N.
- Diversidade de **direcionamento**: unidirecional e bidirecional (ao menos um).
- Evidência de **composição ou agregação**.
- Uso de **List, Map, Set** etc. com operações de adição, busca e remoção.
- Conexão com banco relacional via **JDBC**.
- Uso obrigatório do **padrão DAO**.

- Operações **CRUD** para as entidades principais.

2. Descrição do Sistema

A Plataforma de Soluções Colaborativas permite que usuários interajam em um ecossistema de projetos, tarefas e soluções. As principais funcionalidades incluem:

- **Gerenciamento de Usuários:** Cadastro, atualização e desativação de perfis de usuário, incluindo informações como biografia, foto de perfil e habilidades.
- **Criação e Gerenciamento de Projetos:** Usuários podem criar projetos, definir títulos, descrições e status, além de associá-los a um responsável.
- **Gestão de Tarefas:** Dentro de cada projeto, tarefas podem ser criadas com título, descrição, status, prioridade e um usuário responsável.
- **Proposição de Soluções:** Usuários podem propor soluções para tarefas específicas, incluindo título, descrição e status da submissão.
- **Sistema de Avaliação:** Soluções propostas podem ser avaliadas por outros usuários com uma nota (de 0 a 5) e um comentário, impactando a qualidade das soluções.
- **Relacionamentos e Colaboração:** O sistema suporta relacionamentos complexos entre entidades, como um usuário ser responsável por múltiplos projetos e tarefas, e várias soluções serem propostas para uma única tarefa.

3. Estrutura do Projeto

O projeto está estruturado em pacotes que separam as responsabilidades, seguindo um design modular e facilitando a manutenção.

3.1 Organização de Pacotes

```
src/
├── bd/                 # Configs do banco (ConnectionFactory.java)
│   └── ConnectionFactory.java
├── dao/                # Classes de acesso a dados (padrão DAO)
│   ├── AvaliacaoDAO.java
│   ├── BaseDAO.java
│   ├── PerfilUsuarioDAO.java
│   ├── ProjetoDAO.java
│   ├── SolucaoDAO.java
│   ├── TarefaDAO.java
│   └── UsuarioDAO.java
├── modelo/             # Classes de domínio (modelos de entidades)
│   ├── Avaliacao.java
```

```
│   ├── EntidadeBase.java
│   ├── InterfaceEntidadeBase.java
│   ├── PerfilUsuario.java
│   ├── Projeto.java
│   ├── Solucao.java
│   ├── Tarefa.java
│   └── Usuario.java
└── Main.java               # Classe principal para execução e testes
```

3.2 Entidades Principais

As principais entidades do sistema e seus atributos são:

- **PerfilUsuario:**
  - id: String
  - biografia: String
  - fotoPerfilUri: String
  - habilidades: Set
  - usuarioId: String (Relacionamento 1:1 com Usuario)
- **Usuario:**
  - id: String
  - nome: String
  - email: String
  - senhaCriptografada: String
  - ativo: boolean
  - dataCadastro: Timestamp
  - perfil: PerfilUsuario (Relacionamento bidirecional 1:1)
- **Projeto:**
  - id: String
  - titulo: String
  - descricao: String
  - status: String
  - usuarioId: String (Relacionamento 1:N com Usuario, criador do projeto)
  - dataCriacao: Timestamp
  - dataConclusao: Timestamp
- **Tarefa:**
  - id: String
  - titulo: String
  - descricao: String

    - prioridade: String
    - dataCriacao: Timestamp
    - dataConclusao: Timestamp
    - projetoId: String (Relacionamento 1:N com Projeto)
    - usuarioResponsavelId: String (Relacionamento 1:N com Usuario)
- **Solucao:**
    - id: String
    - titulo: String
    - descricao: String
    - status: String
    - dataSubmissao: Timestamp
    - tarefaId: String (Relacionamento 1:N com Tarefa)
    - usuarioId: String (Relacionamento 1:N com Usuario, autor da solução)
- **Avaliacao:**
    - id: String
    - nota: int
    - comentario: String
    - dataAvaliacao: Timestamp
    - solucaoId: String (Relacionamento 1:N com Solucao)
    - usuarioAvaliadorId: String (Relacionamento 1:N com Usuario)

4. Aplicação dos Pilares de Orientação a Objetos

4.1 Abstração

A **abstração** foi utilizada para modelar as entidades do sistema, focando nas características e comportamentos essenciais de cada conceito.

- **EntidadeBase:** Abstrai as propriedades comuns a todas as entidades persistíveis, como um identificador único (id).
- **Usuario:** Representa um usuário com atributos como nome, email e senha, e comportamentos como autenticação e cálculo de estatísticas.
- **Projeto:** Abstrai a ideia de um projeto com título, descrição, status e responsável, e comportamentos como calcular progresso.
- **Tarefa:** Representa uma unidade de trabalho com título, descrição, status, prioridade e responsável, com métodos para verificar atraso e encontrar a melhor solução.
- **Solucao:** Modelagem de uma solução proposta para uma tarefa, com título, descrição e status, permitindo avaliações e cálculo de qualidade.
- **Avaliacao:** Abstração de uma avaliação de solução, com nota, comentário e referências ao avaliador e à solução.

- **PerfilUsuario:** Abstrai informações complementares do usuário, como biografia e habilidades.

4.2 Encapsulamento

O **encapsulamento** é aplicado através do uso de modificadores de acesso (private para atributos e public para métodos de acesso e manipulação), garantindo que o estado interno dos objetos seja acessado e modificado de forma controlada.

- Em Usuario.java, nome, email e senhaCriptografada são private e acessados via get e set métodos.
- A classe PerfilUsuario controla a adição e remoção de habilidades através de métodos específicos, protegendo o acesso direto ao Set interno.
- Avaliacao.java possui validação da nota no método setNota, impedindo a atribuição de valores inválidos (nota deve estar entre 0 e 5).

4.3 Herança

A **herança** foi utilizada para promover a reutilização de código e estabelecer uma hierarquia de classes clara.

- Todas as classes de modelo (Usuario, Projeto, Tarefa, Solucao, Avaliacao, PerfilUsuario) herdam de EntidadeBase. Isso garante que todas as entidades possuam um id e implementem métodos como getDescricaoEntidade() e isValid().

4.4 Polimorfismo

O **polimorfismo** é demonstrado através da sobrescrita de métodos e sobrecarga de construtores/métodos.

- **Sobrescrita de métodos abstratos:** As classes que herdam de EntidadeBase (ex: Usuario, Projeto) implementam os métodos abstratos getDescricaoEntidade() e isValid() de maneiras específicas para cada entidade.
- **Sobrecarga de construtores:** Múltiplos construtores são fornecidos em classes como Usuario e Projeto, permitindo diferentes formas de instanciar objetos.
- **Sobrecarga de métodos:**
    - Avaliacao.java: Possui dois métodos atualizarAvaliacao() (um com apenas a nota, outro com nota e comentário). Possui dois métodos isRecomendacao() (um com o nível de exigência).
    - Projeto.java: Possui dois métodos atualizarStatus() (um com limiar de conclusão, outro sem).
    - Solucao.java: Possui dois métodos aprovar() (um simples, outro que pode concluir a tarefa associada). Possui dois métodos calcularQualidade() (um padrão, outro com critério de notas).
    - Tarefa.java: Possui dois métodos concluirTarefa() (um simples, outro com

comentário).
    - Usuario.java: Possui dois métodos verificarSenha() (um padrão, outro com algoritmo específico).

5. Modelagem de Classes e Banco de Dados

5.1 Diagrama de Classes

```
@startuml
!theme plain
skinparam classAttributeIconSize 0

interface InterfaceEntidadeBase {
   + getId(): String
   + setId(id: String): void
}

abstract class EntidadeBase {
   # id: String
   --
   # EntidadeBase()
   # EntidadeBase(id: String)
   + getId(): String
   + setId(id: String): void
   + {abstract} getDescricaoEntidade(): String
   + {abstract} isValid(): boolean
   + equals(o: Object): boolean
   + hashCode(): int
}

class Usuario {
   - nome: String
   - email: String
   - senhaCriptografada: String
   - ativo: boolean
   - dataCadastro: Timestamp
   - perfil: PerfilUsuario
   - projetos: List<Projeto>
   - tarefasResponsavel: List<Tarefa>
   - solucoes: List<Solucao>
```

```
    - avaliacoesRealizadas: List<Avaliacao>
    --
    + Usuario()
    + Usuario(id: String, nome: String, email: String, senhaCriptografada: String)
    + getNome(): String
    + setNome(nome: String): void
    + getEmail(): String
    + setEmail(email: String): void
    + getSenhaCriptografada(): String
    + setSenhaCriptografada(senhaCriptografada: String): void
    + isAtivo(): boolean
    + setAtivo(ativo: boolean): void
    + getDataCadastro(): Timestamp
    + setDataCadastro(dataCadastro: Timestamp): void
    + getPerfil(): PerfilUsuario
    + setPerfil(perfil: PerfilUsuario): void
    + getProjetos(): List<Projeto>
    + adicionarProjeto(projeto: Projeto): void
    + removerProjeto(projeto: Projeto): void
    + getTarefasResponsavel(): List<Tarefa>
    + adicionarTarefaResponsavel(tarefa: Tarefa): void
    + removerTarefaResponsavel(tarefa: Tarefa): void
    + getSolucoes(): List<Solucao>
    + adicionarSolucao(solucao: Solucao): void
    + removerSolucao(solucao: Solucao): void
    + getAvaliacoesRealizadas(): List<Avaliacao>
    + adicionarAvaliacaoRealizada(avaliacao: Avaliacao): void
    + removerAvaliacaoRealizada(avaliacao: Avaliacao): void
    + contarTarefasConcluidas(): int
    + calcularMediaAvaliacoesRecebidas(): double
    + verificarSenha(senhaTexto: String): boolean
    + verificarSenha(senhaTexto: String, algoritmo: String): boolean
    + getDescricaoEntidade(): String
    + isValid(): boolean
    + toString(): String
}

class PerfilUsuario {
    - biografia: String
```

```
    - fotoPerfilUri: String
    - habilidades: Set<String>
    - usuarioId: String
    - usuario: Usuario
    --
    + PerfilUsuario()
    + PerfilUsuario(id: String, biografia: String, fotoPerfilUri: String)
    + getBiografia(): String
    + setBiografia(biografia: String): void
    + getFotoPerfilUri(): String
    + setFotoPerfilUri(fotoPerfilUri: String): void
    + getHabilidades(): Set<String>
    + adicionarHabilidade(habilidade: String): void
    + removerHabilidade(habilidade: String): void
    + setHabilidades(habilidades: Set<String>): void
    + getUsuarioId(): String
    + setUsuarioId(usuarioId: String): void
    + getUsuario(): Usuario
    + setUsuario(usuario: Usuario): void
    + temHabilidade(habilidade: String): boolean
    + temHabilidade(habilidade: String, nivelMinimo: int): boolean
    + isPerfilCompleto(): boolean
    + calcularCompletudePercentual(): int
    + calcularCompatibilidade(outroPerfil: PerfilUsuario): int
    + getDescricaoEntidade(): String
    + isValid(): boolean
    + toString(): String
}

class Projeto {
    - titulo: String
    - descricao: String
    - status: String
    - usuarioId: String
    - dataCriacao: Timestamp
    - dataConclusao: Timestamp
    - tarefas: List<Tarefa>
    - responsavel: Usuario
    --
```

```
    + Projeto()
    + Projeto(id: String, titulo: String, descricao: String, usuarioId: String)
    + getTitulo(): String
    + setTitulo(titulo: String): void
    + getDescricao(): String
    + setDescricao(descricao: String): void
    + getStatus(): String
    + setStatus(status: String): void
    + getUsuarioId(): String
    + setUsuarioId(usuarioId: String): void
    + getDataCriacao(): Timestamp
    + setDataCriacao(dataCriacao: Timestamp): void
    + getDataConclusao(): Timestamp
    + setDataConclusao(dataConclusao: Timestamp): void
    + getTarefas(): List<Tarefa>
    + adicionarTarefa(tarefa: Tarefa): void
    + removerTarefa(tarefa: Tarefa): void
    + getResponsavel(): Usuario
    + setResponsavel(responsavel: Usuario): void
    + calcularProgresso(): double
    + estaConcluido(): boolean
    + atualizarStatus(limiarConclusao: double): String
    + atualizarStatus(): String
    + getDescricaoEntidade(): String
    + isValid(): boolean
    + toString(): String
}

class Tarefa {
    - titulo: String
    - descricao: String
    - status: String
    - prioridade: String
    - projetoId: String
    - usuarioResponsavelId: String
    - dataCriacao: Timestamp
    - dataConclusao: Timestamp
    - solucoes: List<Solucao>
    - responsavel: Usuario
```

```
    - projeto: Projeto
    --
    + Tarefa()
    + Tarefa(id: String, titulo: String, descricao: String, projetoId: String,
usuarioResponsavelId: String)
    + getTitulo(): String
    + setTitulo(titulo: String): void
    + getDescricao(): String
    + setDescricao(descricao: String): void
    + getStatus(): String
    + setStatus(status: String): void
    + getPrioridade(): String
    + setPrioridade(prioridade: String): void
    + getProjetoId(): String
    + setProjetoId(projetoId: String): void
    + getUsuarioResponsavelId(): String
    + setUsuarioResponsavelId(usuarioResponsavelId: String): void
    + getDataCriacao(): Timestamp
    + setDataCriacao(dataCriacao: Timestamp): void
    + getDataConclusao(): Timestamp
    + setDataConclusao(dataConclusao: Timestamp): void
    + getSolucoes(): List<Solucao>
    + adicionarSolucao(solucao: Solucao): void
    + removerSolucao(solucao: Solucao): void
    + getResponsavel(): Usuario
    + setResponsavel(responsavel: Usuario): void
    + getProjeto(): Projeto
    + setProjeto(projeto: Projeto): void
    + estaConcluida(): boolean
    + estaAtrasada(): boolean
    + getMelhorSolucao(): Solucao
    + concluirTarefa(): void
    + concluirTarefa(comentario: String): void
    + calcularDiasRestantes(): long
    + getDescricaoEntidade(): String
    + isValid(): boolean
    + toString(): String
}
```

```
class Solucao {
    - titulo: String
    - descricao: String
    - status: String
    - tarefaId: String
    - usuarioId: String
    - dataSubmissao: Timestamp
    - avaliacoes: List<Avaliacao>
    - autor: Usuario
    - tarefa: Tarefa
    --
    + Solucao()
    + Solucao(id: String, titulo: String, descricao: String, tarefaId: String, usuarioId:
String)
    + getTitulo(): String
    + setTitulo(titulo: String): void
    + getDescricao(): String
    + setDescricao(descricao: String): void
    + getStatus(): String
    + setStatus(status: String): void
    + getTarefaId(): String
    + setTarefaId(tarefaId: String): void
    + getUsuarioId(): String
    + setUsuarioId(usuarioId: String): void
    + getDataSubmissao(): Timestamp
    + setDataSubmissao(dataSubmissao: Timestamp): void
    + getAvaliacoes(): List<Avaliacao>
    + adicionarAvaliacao(avaliacao: Avaliacao): void
    + removerAvaliacao(avaliacao: Avaliacao): void
    + getAutor(): Usuario
    + setAutor(autor: Usuario): void
    + getTarefa(): Tarefa
    + setTarefa(tarefa: Tarefa): void
    + calcularMediaAvaliacoes(): double
    + estaAprovada(): boolean
    + aprovar(): void
    + aprovar(concluirTarefa: boolean): void
    + calcularQualidade(): String
    + calcularQualidade(criterioNotas: int): String
```

```
    + getDescricaoEntidade(): String
    + isValid(): boolean
    + toString(): String
}

class Avaliacao {
    - nota: int
    - comentario: String
    - solucaoId: String
    - usuarioAvaliadorId: String
    - dataAvaliacao: Timestamp
    - usuarioAvaliador: Usuario
    - solucaoAvaliada: Solucao
    --
    + Avaliacao()
    + Avaliacao(id: String, nota: int, comentario: String, solucaoId: String,
usuarioAvaliadorId: String)
    + getNota(): int
    + setNota(nota: int): void
    + getComentario(): String
    + setComentario(comentario: String): void
    + getSolucaoId(): String
    + setSolucaoId(solucaoId: String): void
    + getUsuarioAvaliadorId(): String
    + setUsuarioAvaliadorId(usuarioAvaliadorId: String): void
    + getDataAvaliacao(): Timestamp
    + setDataAvaliacao(dataAvaliacao: Timestamp): void
    + getUsuarioAvaliador(): Usuario
    + setUsuarioAvaliador(usuarioAvaliador: Usuario): void
    + getSolucaoAvaliada(): Solucao
    + setSolucaoAvaliada(solucaoAvaliada: Solucao): void
    + isAvaliaoPositiva(): boolean
    + atualizarAvaliacao(novaNota: int): void
    + atualizarAvaliacao(novaNota: int, novoComentario: String): void
    + isRecomendacaoPremium(): boolean
    + isRecomendacao(nivel: int): boolean
    + getDescricaoEntidade(): String
    + isValid(): boolean
    + toString(): String
```

```
}

EntidadeBase <|-- Usuario
EntidadeBase <|-- PerfilUsuario
EntidadeBase <|-- Projeto
EntidadeBase <|-- Tarefa
EntidadeBase <|-- Solucao
EntidadeBase <|-- Avaliacao

InterfaceEntidadeBase <|.. EntidadeBase

Usuario "1" -- "0..1" PerfilUsuario : tem >
Usuario "1" -- "0..N" Projeto : cria >
Usuario "1" -- "0..N" Tarefa : é responsável por >
Usuario "1" -- "0..N" Solucao : é autor de >
Usuario "1" -- "0..N" Avaliacao : realiza >

Projeto "1" -- "0..N" Tarefa : contém >

Tarefa "1" -- "0..N" Solucao : possui >

Solucao "1" -- "0..N" Avaliacao : recebe >
@enduml
```

5.2 Modelo ER do Banco de Dados

O modelo Entidade-Relacionamento abaixo representa a estrutura do banco de dados que suporta o sistema, com base no script SQL fornecido.

```
@startuml
!theme plain
hide circle
skinparam linetype ortho

entity usuario {
   id: VARCHAR(36) <<PK>>
   nome: VARCHAR(100)
   email: VARCHAR(100)
   senha_criptografada: VARCHAR(100)
```

```
    data_cadastro: TIMESTAMP
    ativo: BOOLEAN
}

entity perfil_usuario {
    id: VARCHAR(36) <<PK>>
    biografia: TEXT
    foto_perfil_uri: VARCHAR(255)
    habilidades: TEXT
    usuario_id: VARCHAR(36) <<FK>>
}

entity projetos {
    id: VARCHAR(36) <<PK>>
    titulo: VARCHAR(100)
    descricao: TEXT
    data_criacao: TIMESTAMP
    data_conclusao: TIMESTAMP
    usuario_id: VARCHAR(36) <<FK>>
    status: VARCHAR(20)
}

entity tarefas {
    id: VARCHAR(36) <<PK>>
    titulo: VARCHAR(100)
    descricao: TEXT
    status: VARCHAR(20)
    data_criacao: TIMESTAMP
    data_conclusao: TIMESTAMP
    projeto_id: VARCHAR(36) <<FK>>
    usuario_responsavel_id: VARCHAR(36) <<FK>>
    prioridade: VARCHAR(20)
}

entity solucoes {
    id: VARCHAR(36) <<PK>>
    titulo: VARCHAR(100)
    descricao: TEXT
    data_submissao: TIMESTAMP
```

```
    tarefa_id: VARCHAR(36) <<FK>>
    usuario_id: VARCHAR(36) <<FK>>
    status: VARCHAR(20)
}

entity avaliacoes {
    id: VARCHAR(36) <<PK>>
    nota: INT
    comentario: TEXT
    data_avaliacao: TIMESTAMP
    solucao_id: VARCHAR(36) <<FK>>
    usuario_avaliador_id: VARCHAR(36) <<FK>>
}

usuario ||--o{ perfil_usuario : "1:1"
usuario ||--o{ projetos : "1:N"
usuario ||--o{ tarefas : "1:N"
usuario ||--o{ solucoes : "1:N"
usuario ||--o{ avaliacoes : "1:N"

projetos ||--o{ tarefas : "1:N"

tarefas ||--o{ solucoes : "1:N"

solucoes ||--o{ avaliacoes : "1:N"
@enduml
```

6. Uso de Classes Abstratas e Interfaces

6.1 Classe Abstrata EntidadeBase

A classe abstrata EntidadeBase serve como a base para todas as entidades persistíveis no sistema.

```
package modelo;

import java.util.Objects;
import java.util.concurrent.atomic.AtomicInteger;
```

```
public abstract class EntidadeBase implements InterfaceEntidadeBase {
    private String id;
    // Contador estático compartilhado por todas as entidades para gerar IDs
incrementais
    private static final AtomicInteger contador = new AtomicInteger(1);

    protected EntidadeBase() {
        // Gera um ID incremental automaticamente
        this.id = String.valueOf(contador.getAndIncrement());
    }

    protected EntidadeBase(String id) {
        this.id = id != null ? id : String.valueOf(contador.getAndIncrement());
    }

    @Override
    public String getId() {
        return id;
    }

    @Override
    public void setId(String id) {
        if (id == null || id.trim().isEmpty()) {
            throw new IllegalArgumentException("O ID não pode estar vazio");
        }
        this.id = id;
    }

    /**
     * Método abstrato que deve ser implementado por todas as subclasses
     * para fornecer uma descrição significativa da entidade.
     * @return String com a descrição da entidade
     */
    public abstract String getDescricaoEntidade();

    /**
     * Verifica se a entidade é válida para persistência
     * @return true se a entidade estiver válida, false caso contrário
     */
```

```
    public abstract boolean isValid();

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        EntidadeBase that = (EntidadeBase) o;
        return Objects.equals(id, that.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
}
```

Esta classe abstrata:

- Define um atributo id de forma protegida.
- Gera IDs automaticamente para novas instâncias.
- Implementa equals() e hashCode() com base no id.
- Declara métodos abstratos getDescricaoEntidade() e isValid() que devem ser implementados pelas subclasses.

6.2 Interface InterfaceEntidadeBase

A interface InterfaceEntidadeBase define os métodos básicos de acesso ao identificador de uma entidade.

```
package modelo;

public interface InterfaceEntidadeBase {
    String getId();

    void setId(String id);
}
```

Esta interface é implementada por EntidadeBase, garantindo que todas as entidades do sistema (direta ou indiretamente) sigam esse contrato para manipulação de seus

IDs.

7. Uso de Collections

O sistema faz uso de diversas estruturas de coleções para gerenciar dados e relacionamentos entre as entidades.

7.1 Uso de List

As **listas** são amplamente utilizadas para representar relacionamentos 1:N (um-para-muitos), onde a ordem dos elementos pode ser relevante ou onde duplicações podem ser aceitas.

- Projeto contém uma List<Tarefa>: Um projeto pode ter múltiplas tarefas associadas.
- Tarefa contém uma List<Solucao>: Uma tarefa pode receber várias soluções propostas.
- Solucao contém uma List<Avaliacao>: Uma solução pode ser avaliada múltiplas vezes.
- Usuario mantém listas de Projetos, Tarefas, Solucaos e Avaliacoes que lhe estão relacionadas.

Exemplo (Projeto.java):

```
private List<Tarefa> tarefas; // Inicializada como new ArrayList<>();
public List<Tarefa> getTarefas() {
   return new ArrayList<>(tarefas); // Retorna uma cópia da lista para evitar
modificações externas
}
public void adicionarTarefa(Tarefa tarefa) {
   if (tarefa == null) {
      throw new IllegalArgumentException("A tarefa não pode ser nula");
   }
   this.tarefas.add(tarefa); // Adição de elementos
}
public void removerTarefa(Tarefa tarefa) {
   this.tarefas.remove(tarefa); // Remoção de elementos
}
```

7.2 Uso de Set

A classe PerfilUsuario utiliza um Set<String> para armazenar as **habilidades**,

garantindo que não haja habilidades duplicadas para um mesmo perfil, pois a ordem não é relevante.

Exemplo (PerfilUsuario.java):

```
private Set<String> habilidades; // Inicializada como new HashSet<>();
public Set<String> getHabilidades() {
   return new HashSet<>(habilidades); // Retorna uma cópia do conjunto
}
public void adicionarHabilidade(String habilidade) {
   if (habilidade == null || habilidade.trim().isEmpty()) {
      throw new IllegalArgumentException("A habilidade não pode estar vazia");
   }
   this.habilidades.add(habilidade.trim().toLowerCase()); // Adição de elementos e
garantia de unicidade
}
public void removerHabilidade(String habilidade) {
   if (habilidade != null) {
      this.habilidades.remove(habilidade.trim().toLowerCase()); // Remoção de
elementos
   }
}
```

7.3 Uso de Map

Os DAOs (**Data Access Objects**) utilizam **Maps** para retornar estatísticas e agrupamentos de dados de forma organizada.

- AvaliacaoDAO: Métodos como obterEstatisticasGerais() e obterDistribuicaoNotas() retornam Map<String, Double> ou Map<String, Long> para agrupar dados por chaves (status, prioridade, etc.).
- TarefaDAO: Métodos como obterEstatisticasPorPrioridade() e obterEstatisticasPorStatus() retornam Map<String, Long>.
- ProjetoDAO: O método obterEstatisticasProjetos() retorna um Map<String, Long>.
- UsuarioDAO: O método obterEstatisticasUsuario() retorna um Map<String, Object> com diversas métricas.

Exemplo (TarefaDAO.java):

```
public Map<String, Long> obterEstatisticasPorPrioridade() {
   Map<String, Long> estatisticas = new HashMap<>(); // Uso de HashMap para
```

```
armazenar estatísticas
    String sql = """
        SELECT
            prioridade,
            COUNT(*) as quantidade
        FROM tarefas
        GROUP BY prioridade
    """;
    // ... lógica de preenchimento do mapa
    return estatisticas;
}
```

7.4 Operações com Collections

Além das operações básicas de adição e remoção, o sistema utiliza funcionalidades mais avançadas das Collections, como **Streams** para operações de busca e agregação de dados, como o cálculo de médias e contagens.

- Usuario.java: contarTarefasConcluidas() e calcularMediaAvaliacoesRecebidas() utilizam streams e filtros.
- Projeto.java: calcularProgresso() utiliza stream e filtro para contar tarefas concluídas.
- Tarefa.java: getMelhorSolucao() usa streams e max() com um comparador para encontrar a melhor solução.
- Solucao.java: calcularMediaAvaliacoes() utiliza streams para calcular a média das notas.

8. Persistência com JDBC e Padrão DAO

8.1 Padrão DAO (Data Access Object)

O sistema implementa o padrão **DAO** para separar a lógica de negócio da lógica de acesso a dados, seguindo o requisito do projeto. Este padrão proporciona:

- **Isolamento de Responsabilidades:** A lógica de manipulação de dados do banco fica isolada em classes DAO, sem impactar as classes de modelo ou de negócio.
- **Facilidade de Manutenção e Evolução:** Alterações no esquema do banco de dados ou no provedor de persistência afetam apenas as classes DAO, minimizando o impacto no restante da aplicação.
- **Reusabilidade:** As operações CRUD são padronizadas e reutilizadas para cada entidade.

8.1.1 Interface BaseDAO

Define as operações básicas de CRUD (Create, Read, Update, Delete) que todos os DAOs devem implementar.

```
package dao;

import java.util.ArrayList;

public interface BaseDAO<T> {
   void salvar(T objeto);
   Object buscarPorId(String id); // Note o tipo String para o ID
   ArrayList<T> listarTodosLazyLoading();
   void atualizar(T objeto);
   void excluir(String id); // Note o tipo String para o ID
}
```

A interface BaseDAO é genérica (<T>), permitindo que os DAOs específicos trabalhem com seus respectivos tipos de entidade.

8.1.2 Classes DAO Específicas

AvaliacaoDAO, PerfilUsuarioDAO, ProjetoDAO, SolucaoDAO, TarefaDAO e UsuarioDAO implementam a interface BaseDAO para suas respectivas entidades.

Exemplo de implementação em ProjetoDAO.java:

```
package dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import modelo.Projeto;
import modelo.Tarefa;
import modelo.Usuario;
```

```
public class ProjetoDAO implements BaseDAO<Projeto> {

    private Connection connection;

    public ProjetoDAO(Connection connection) {
        this.connection = connection;
    }

    public void criarTabela() throws SQLException {
        String sql = """
            CREATE TABLE IF NOT EXISTS projetos (
                id VARCHAR(36) PRIMARY KEY,
                titulo VARCHAR(100) NOT NULL,
                descricao TEXT,
                data_criacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                data_conclusao TIMESTAMP,
                usuario_id VARCHAR(36),
                status VARCHAR(20) DEFAULT 'EM_ANDAMENTO',
                FOREIGN KEY (usuario_id) REFERENCES usuario(id)
            )
        """;

        try (var stmt = connection.createStatement()) {
            stmt.execute(sql);
        }
    }

    @Override
    public void salvar(Projeto projeto) {
        String sql = "INSERT INTO projetos (id, titulo, descricao, usuario_id, status)
VALUES (?, ?, ?, ?, ?)";

        try (PreparedStatement pstm = connection.prepareStatement(sql)) {
            pstm.setString(1, projeto.getId());
            pstm.setString(2, projeto.getTitulo());
            pstm.setString(3, projeto.getDescricao());
            pstm.setString(4, projeto.getUsuarioId());
            pstm.setString(5, projeto.getStatus());
```

```
            pstm.execute();
        } catch (SQLException e) {
            throw new RuntimeException("Erro ao salvar projeto: " + e.getMessage());
        }
    }

    // Métodos buscarPorId, listarTodosLazyLoading, atualizar, excluir, etc.
    // ...

    private Projeto criarProjeto(ResultSet rs) throws SQLException {
        Projeto projeto = new Projeto();
        projeto.setId(rs.getString("id"));
        projeto.setTitulo(rs.getString("titulo"));
        projeto.setDescricao(rs.getString("descricao"));
        projeto.setUsuarioId(rs.getString("usuario_id"));
        projeto.setStatus(rs.getString("status"));
        projeto.setDataCriacao(rs.getTimestamp("data_criacao"));
        projeto.setDataConclusao(rs.getTimestamp("data_conclusao"));
        return projeto;
    }
}
```

8.1.3 Padrões de Carregamento (Lazy Loading)

O sistema principalmente utiliza o conceito de **Lazy Loading** ao carregar entidades. Por exemplo, os métodos listarTodosLazyLoading() em cada DAO carregam apenas os dados diretos da tabela da entidade, sem buscar automaticamente seus relacionamentos. Os objetos relacionados (ex: Usuario em Projeto, Tarefa em Projeto) são populados nas classes de modelo quando são explicitamente buscados ou definidos, evitando a carga desnecessária de dados.

8.2 Conexão com o Banco de Dados (JDBC)

A conexão com o banco de dados MySQL é realizada via **JDBC**, conforme os requisitos.

A classe ConnectionFactory é responsável por fornecer a conexão.

```
public class ConnectionFactory {
  public Connection recuperaConexao() {
```

```
    try {
      String sgbd = "mysql";
      String endereco = "localhost";
      String bd = "plataforma_de_solucoes_colaborativas";
      String usuario = "root"; // <<-- Credenciais hardcoded
      String senha = "Bnk03112005@"; // <<-- Credenciais hardcoded

      Connection connection = DriverManager.getConnection(
          "jdbc:" + sgbd + "://" + endereco + "/" + bd, usuario, senha);

      return connection;
    } catch (SQLException e) {
      throw new RuntimeException(e);
    }
  }
}
```

- **Gerenciamento da Conexão:** A classe ConnectionFactory centraliza a lógica de conexão, utilizando DriverManager.getConnection() para estabelecer a conexão com o banco de dados MySQL.
- **Credenciais Hardcoded:** Atualmente, as credenciais de banco de dados (usuario e senha) estão diretamente no código. Em um ambiente de produção, é recomendável usar variáveis de ambiente ou arquivos de configuração seguros para estas informações.

8.3 Operações CRUD

Todas as entidades principais (Usuario, PerfilUsuario, Projeto, Tarefa, Solucao, Avaliacao) possuem operações **CRUD** completas implementadas em seus respectivos DAOs, conforme o requisito.

- **Create** (salvar): Insere um novo registro no banco de dados.
- **Read** (buscarPorId, listarTodosLazyLoading, e métodos de busca específicos): Recupera registros do banco.
- **Update** (atualizar): Modifica registros existentes.
- **Delete** (excluir): Remove registros do banco (ou realiza um "soft delete" como no caso de Usuario).

9. Funcionalidades Implementadas

A aplicação implementa um conjunto robusto de funcionalidades para a gestão

colaborativa de projetos e tarefas, com um foco claro na interação do usuário e na gestão da informação.

9.1 Gerenciamento de Usuários e Perfis

- **Criação de Usuário:** Permite o cadastro de novos usuários com nome, email e senha.
- **Criação de Perfil de Usuário:** Perfis são associados aos usuários, contendo biografia, foto de perfil e habilidades.
- **Atualização de Usuário e Perfil:** Nomes e e-mails de usuários podem ser atualizados, assim como informações do perfil.
- **Soft Delete de Usuário:** Usuários não são excluídos permanentemente, mas marcados como inativo, preservando o histórico de suas contribuições.
- **Autenticação:** Verificação de credenciais de login (email e senha).
- **Busca por Habilidades:** Permite encontrar usuários com habilidades específicas.
- **Estatísticas por Usuário:** Retorna o total de projetos, tarefas, soluções e avaliações de um usuário, além da média de avaliações recebidas.

9.2 Gerenciamento de Projetos

- **Criação de Projeto:** Usuários podem iniciar novos projetos com título, descrição e status inicial.
- **Busca de Projetos:** Projetos podem ser listados por usuário, por período de criação, ou por status (ativos).
- **Conclusão de Projeto:** Altera o status de um projeto para CONCLUIDO e registra a data de conclusão.
- **Cálculo de Progresso:** Determina o percentual de conclusão de um projeto com base nas tarefas finalizadas.
- **Estatísticas de Projetos:** Fornece o total de projetos, quantos estão em andamento e quantos foram concluídos.
- **Equipe do Projeto:** Lista todos os usuários responsáveis por tarefas dentro de um projeto específico.

9.3 Gerenciamento de Tarefas

- **Criação de Tarefa:** Adiciona tarefas a projetos existentes, com título, descrição, status, prioridade e um responsável.
- **Busca de Tarefas:** Tarefas podem ser buscadas por projeto, por usuário responsável, por prioridade, por status (pendentes) ou por proximidade da data de entrega.
- **Conclusão de Tarefa:** Altera o status de uma tarefa para CONCLUIDA e registra a data de conclusão.

- **Atualização de Prioridade:** Permite ajustar a prioridade de uma tarefa.
- **Tarefas em Atraso:** Identifica tarefas que estão atrasadas e ainda não foram concluídas.
- **Estatísticas de Tarefas:** Oferece contagens de tarefas por prioridade e por status.
- **Desempenho de Usuários em Tarefas:** Calcula o total de tarefas, tarefas concluídas e a média de dias para conclusão por usuário.

9.4 Gerenciamento de Soluções

- **Criação de Solução:** Permite que usuários submetam soluções para tarefas específicas.
- **Busca de Soluções:** Soluções podem ser encontradas por tarefa, por usuário autor, por status, ou por popularidade (média de avaliações).
- **Atualização de Status:** Modifica o status de uma solução (por exemplo, para aprovada ou rejeitada).
- **Qualidade da Solução:** Calcula a qualidade de uma solução com base na média das avaliações recebidas.
- **Soluções Recentes:** Lista as soluções mais recentemente submetidas.

9.5 Gerenciamento de Avaliações

- **Criação de Avaliação:** Usuários podem avaliar soluções com uma nota e um comentário.
- **Busca de Avaliações:** Avaliações podem ser buscadas por solução ou por usuário avaliador.
- **Cálculo da Média de Avaliações:** Calcula a média das notas para uma solução específica ou para um período.
- **Estatísticas de Avaliações:** Fornece a média geral, nota mínima, máxima e total de avaliações no sistema.
- **Ranking de Soluções:** Apresenta um ranking das soluções com melhor média de notas, exigindo um número mínimo de avaliações.
- **Distribuição de Notas:** Mostra a contagem de avaliações para cada nota (0 a 5).

10. Evidências de Execução

Esta seção deve ser preenchida com as capturas de tela da execução do sistema.

- Cadastro e Listagem de Usuários:
  [Espaço para inserir captura de tela]
- Criação de Projetos e Tarefas:
  [Espaço para inserir captura de tela]
- Submissão de Soluções e Avaliações:

[Espaço para inserir captura de tela]
- Buscas e Relatórios de Estatísticas:
[Espaço para inserir captura de tela]

11. Conclusão e Desafios

11.1 Aprendizados

O desenvolvimento desta plataforma colaborativa proporcionou a aplicação prática e o aprofundamento nos seguintes conceitos de Programação Orientada a Objetos e desenvolvimento de software:

- **Modelagem de Classes e Relacionamentos:** Experiência na criação de um modelo de domínio complexo com diversas cardinalidades (1:1, 1:N) e direcionamentos, demonstrando agregação e composição.
- **Pilares da Orientação a Objetos:** Fortalecimento da compreensão e aplicação de abstração, encapsulamento, herança e polimorfismo em um projeto real.
- **Padrões de Projeto:** Implementação e entendimento do padrão DAO para desacoplamento da camada de persistência.
- **Persistência de Dados:** Uso de JDBC para interação com bancos de dados relacionais e execução de operações CRUD.
- **Gerenciamento de Coleções:** Utilização estratégica de List e Set para diferentes cenários de dados, incluindo operações de filtragem e agregação com Streams.
- **Tratamento de Exceções:** Gerenciamento robusto de erros em operações de banco de dados e validações de negócio.

11.2 Desafios Enfrentados

Durante o desenvolvimento, surgiram alguns desafios notáveis:

- **Gerenciamento de Dependências em Cascata:** A exclusão de entidades com relacionamentos (ex: um Projeto ter Tarefas, que por sua vez têm Solucoes e Avaliacoes) exigiu um planejamento cuidadoso das operações de exclusão em cascata dentro dos DAOs para garantir a integridade dos dados (ex: ProjetoDAO excluindo tarefas relacionadas, TarefaDAO excluindo soluções relacionadas, SolucaoDAO excluindo avaliações relacionadas).
- **Conceitos de Lazy Loading:** O desafio foi garantir que os objetos pudessem ser carregados de forma eficiente (lazy loading) e que os relacionamentos fossem estabelecidos corretamente no modelo após a recuperação do banco de dados, sem carregar dados desnecessários.
- **Mapeamento entre Modelo e Banco:** A conversão de objetos do modelo para registros de banco de dados e vice-versa exigiu atenção à formatação de datas, booleanos e coleções (Set<String> de habilidades em PerfilUsuario).

11.3 Melhorias Futuras

Para aprimorar a Plataforma de Soluções Colaborativas, as seguintes melhorias poderiam ser implementadas:

- **Interface Gráfica (GUI):** Desenvolver uma interface de usuário amigável (e.g., com JavaFX ou Swing) para uma melhor experiência de interação.
- **Segurança (Criptografia Avançada):** Implementar algoritmos de hash mais robustos (ex: BCrypt) para senhas e um sistema de autenticação e autorização mais completo.
- **Buscas Avançadas:** Aprimorar as funcionalidades de busca com filtros combinados por múltiplos critérios (ex: buscar tarefas por projeto, status e responsável simultaneamente).
- **Notificações:** Desenvolver um sistema de notificações para alertar usuários sobre novas soluções em suas tarefas, novas avaliações, etc.
- **Geração de Relatórios:** Exportar dados e estatísticas para formatos como CSV, PDF ou JSON para análise externa.
- **Moderação de Conteúdo:** Implementar um sistema de moderação mais sofisticado para gerenciar conteúdo inadequado em descrições, comentários e soluções.
- **Otimização de Consultas:** Para grandes volumes de dados, refinar as consultas SQL e índices do banco para otimizar o desempenho.

12. Tecnologias Utilizadas

- **Java:** Linguagem de programação principal.
- **MySQL:** Sistema de gerenciamento de banco de dados relacional.
- **JDBC:** API para conexão e manipulação de banco de dados Java.
- **Maven:** Ferramenta para gerenciamento de projetos e dependências.

13. Instruções de Execução

A classe principal para iniciar a aplicação e executar os testes é: src/Main.java.

Pré-requisitos:

- Java Development Kit (JDK) 21 ou superior.
- MySQL Server 8.0 ou superior.
- Maven.

Configuração do Banco de Dados:

Crie o banco de dados MySQL:

CREATE SCHEMA plataforma_de_solucoes_colaborativas;

Ajuste as credenciais de conexão diretamente no arquivo
src/bd/ConnectionFactory.java:

```
// ... dentro da classe ConnectionFactory
public class ConnectionFactory {
  public Connection recuperaConexao() {
    try {
      String sgbd = "mysql";
      String endereco = "localhost";
      String bd = "plataforma_de_solucoes_colaborativas"; // Nome do banco de dados
      String usuario = "[SEU_USUARIO_MYSQL]"; // <<-- ALtere aqui
      String senha = "[SUA_SENHA_MYSQL]";    // <<-- ALtere aqui

      Connection connection = DriverManager.getConnection(
          "jdbc:" + sgbd + "://" + endereco + "/" + bd, usuario, senha);

      return connection;
    } catch (SQLException e) {
      throw new RuntimeException(e);
    }
  }
}
```

O script SQL para criação das tabelas será executado automaticamente pelo
Main.java na primeira execução do sistema, através dos DAOs.

Execução da Aplicação:

1. Clone o repositório (se ainda não o fez):
   git clone https://github.com/gl-dias/plataforma-colaborativa.git

2. Entre no diretório do projeto (ajuste o nome se for diferente):
   cd plataforma-colaborativa # Ou cd minha-agenda, se o git clone criou com esse nome

3. Compile o projeto com Maven (isso resolverá as dependências e compilará o código):

mvn clean install

4. Execute a aplicação via Maven:
   mvn exec:java -Dexec.mainClass="Main"

Isso executará o método main da classe Main, que irá criar as tabelas e rodar os testes de CRUD demonstrados no código.

14. Referências

- Documentação Java: https://docs.oracle.com/en/java/
- MySQL Documentation: https://dev.mysql.com/doc/
- Requisitos do Projeto: requisitos.md
- Script SQL de Criação do Banco: script.sql
- Maven Project Object Model: pom.xml
- Classe Principal: src/Main.java
- Classe de Conexão com o Banco: src/bd/ConnectionFactory.java
- Interface Base DAO: src/dao/BaseDAO.java
- DAOs Específicos:
  - src/dao/AvaliacaoDAO.java
  - src/dao/PerfilUsuarioDAO.java
  - src/dao/ProjetoDAO.java
  - src/dao/SolucaoDAO.java
  - src/dao/TarefaDAO.java
  - src/dao/UsuarioDAO.java
- Modelos de Entidades:
  - src/modelo/Avaliacao.java
  - src/modelo/EntidadeBase.java
  - src/modelo/InterfaceEntidadeBase.java
  - src/modelo/PerfilUsuario.java
  - src/modelo/Projeto.java
  - src/modelo/Solucao.java
  - src/modelo/Tarefa.java
  - src/modelo/Usuario.java