



**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA
CAMPUS - SANTO ANTÔNIO DE JESUS - BAHIA**

**CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISITEMAS**

MARCELO DE JESUS

RONALDO CORREIA

FRAKLIN FELIX

**ATIVIDADE PADRÕES DE PROJETO- PROJETO FINAL –
PLATAFORMA COLABORATIVA DE APRENDIZAGEM
GAMIFICADA**

SANTO ANTONIO DE JESUS – BA 2025

MARCELO DE JESUS

RONALDO CORREIA

FRAKLIN FELIX

**ATIVIDADE PADRÕES DE PROJETO: PROJETO FINAL –
PLATAFORMA COLABORATIVA DE APRENDIZAGEM
GAMIFICADA**

Relatório técnico da
atividade final de padrões de
projeto- Plataforma Colaborativa
de Aprendizagem Gamificada,
elaborado como requisito parcial
de avaliação para a disciplina
de Padrões de Projeto, ministrada
pelo Prof. Felipe Silva.

SANTO ANTONIO DE JESUS – BA

2025

1. INTRODUÇÃO

Este relatório tem como objetivo demonstrar na prática e de forma descritiva o desenvolvimento de uma plataforma de aprendizagem gamificada, realizada como projeto final da disciplina de Padrões de Projeto, com a junção de todas práticas padronizadas de acordo com os princípios SOLID e padrões GOF. O sistema tem como finalidade simular um ambiente educacional interativo, no qual alunos e visitantes podem participar de desafios, acumular pontos, conquistar medalhas e interagir em um mural coletivo, lembramos que os desafios, serão descritos por professor em formato de quizz/códigos.

A proposta buscou unir conceitos de gamificação com boas práticas de engenharia de software, de modo a aplicar, de forma prática, os padrões de projeto **GOF (Gang of Four)** e os princípios **SOLID**, garantindo modularidade, extensibilidade e coesão entre os componentes.

O escopo do sistema contempla:

- **Gerenciamento de usuários e perfis**, com autenticação e controle de sessão.
- **Módulo de desafios**, no qual é possível criar, responder e registrar atividades avaliativas.
- **Sistema de pontuação e conquistas**, responsável por premiar boas práticas e estimular a continuidade.
- **Histórico de ações e notificações**, possibilitando rastreabilidade das interações dos usuários.
- **Relatórios de desempenho**, exportados em diferentes formatos.

O projeto foi implementado em **Java**, adotando uma arquitetura modular dividida em pacotes, contemplando os requisitos funcionais definidos no enunciado. A aplicação utiliza interface de console como camada de interação, atendendo ao critério de simplicidade sugerido, mas mantendo espaço para futuras extensões em ambiente gráfico ou web.

Além da implementação prática, este relatório apresenta a documentação técnica, os diagramas UML, a análise de aplicação dos padrões de projeto e dos princípios SOLID, bem como a discussão de limitações e possíveis extensões futuras.

2. JUSTIFICATIVA DOS PADRÕES UTILIZADOS

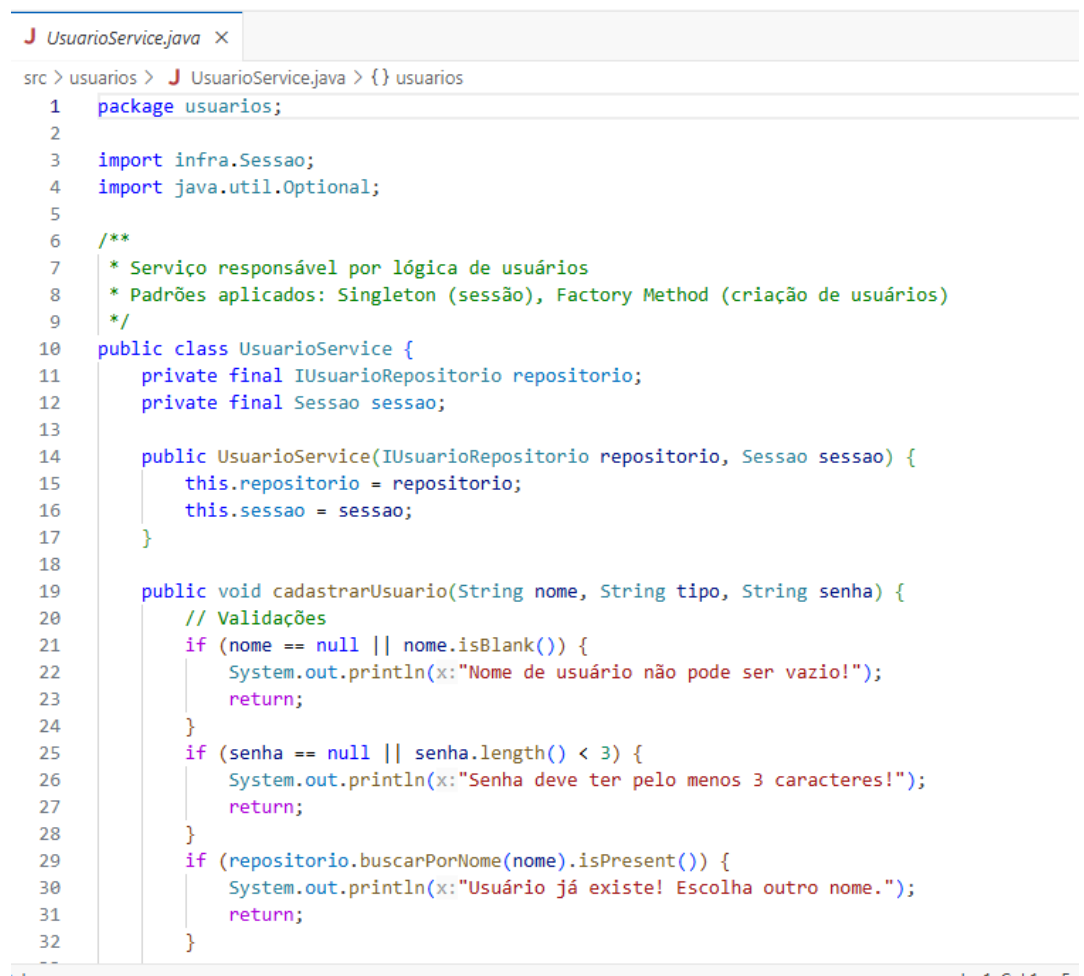
A construção do sistema utilizou diversos padrões de projeto de acordo com cada necessidade vista, princípios e estruturas, principalmente com ênfase na classificação das três categorias: **criação, estruturais e comportamentais**. A seguir, descrevemos cada padrão aplicado, sua justificativa e função dentro do sistema.

2.1 Padrão de Criação

- **Singleton (Controle de Sessão)**

Foi utilizado para garantir que exista apenas uma instância responsável por controlar a sessão ativa no sistema. Esse padrão evita inconsistências, já que somente um usuário pode estar logado por vez no console.

- Exemplo:

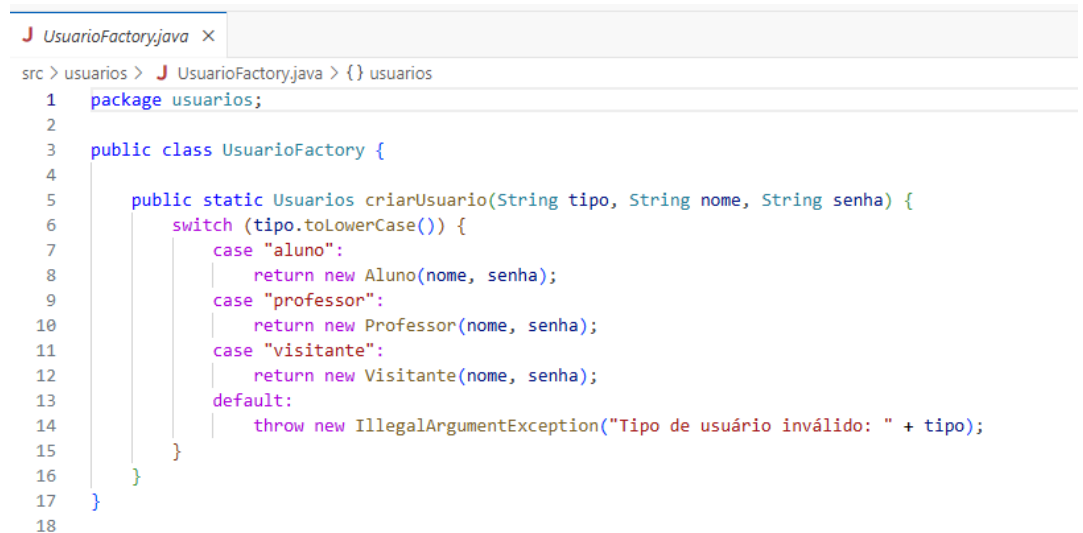


```
J UsuarioService.java X
src > usuarios > J UsuarioService.java > {} usuarios
1 package usuarios;
2
3 import infra.Sessao;
4 import java.util.Optional;
5
6 /**
7  * Serviço responsável por lógica de usuários
8  * Padrões aplicados: Singleton (sessão), Factory Method (criação de usuários)
9  */
10 public class UsuarioService {
11     private final IUsuarioRepositorio repositorio;
12     private final Sessao sessao;
13
14     public UsuarioService(IUsuarioRepositorio repositorio, Sessao sessao) {
15         this.repositorio = repositorio;
16         this.sessao = sessao;
17     }
18
19     public void cadastrarUsuario(String nome, String tipo, String senha) {
20         // Validações
21         if (nome == null || nome.isBlank()) {
22             System.out.println(x:"Nome de usuário não pode ser vazio!");
23             return;
24         }
25         if (senha == null || senha.length() < 3) {
26             System.out.println(x:"Senha deve ter pelo menos 3 caracteres!");
27             return;
28         }
29         if (repositorio.buscarPorNome(nome).isPresent()) {
30             System.out.println(x:"Usuário já existe! Escolha outro nome.");
31             return;
32         }
33     }
34 }
```

- **Factory Method (Perfis de Usuário)**

Permite a criação de diferentes tipos de usuários (Aluno, Professor e Visitante) sem acoplar a lógica de instanciamento diretamente ao código principal. Assim, o sistema pode ser estendido facilmente para novos perfis no futuro.

- Exemplo:



```
src > usuarios > J UsuarioFactory.java > {} usuarios
1 package usuarios;
2
3 public class UsuarioFactory {
4
5     public static Usuarios criarUsuario(String tipo, String nome, String senha) {
6         switch (tipo.toLowerCase()) {
7             case "aluno":
8                 return new Aluno(nome, senha);
9             case "professor":
10                 return new Professor(nome, senha);
11             case "visitante":
12                 return new Visitante(nome, senha);
13             default:
14                 throw new IllegalArgumentException("Tipo de usuário inválido: " + tipo);
15         }
16     }
17 }
18
```

2.2 Padrões Comportamentais

- **Strategy (Pontuação e Relatórios)**

Implementado para definir múltiplas estratégias de pontuação (por tempo, dificuldade, acertos). A escolha ocorre em tempo de execução, tornando o sistema aberto para expansão de novas regras sem modificar o código existente. Nos relatórios foi utilizado para garantir a exportação em múltiplos formatos (CSV, JSON e PDF) facilitando a manipulação de formatos diferentes alterando apenas um parâmetro na chamada do método.

- **Observer** (Notificações de Conquistas)
Utilizado para monitorar ações do usuário e notificar quando este desbloqueia conquistas (ex.: primeira resposta correta, primeira medalha). Tal abordagem desacopla a lógica de premiação do núcleo de execução de desafios.
- **Command** (Histórico e Undo)
Aplicado para registrar todas as ações dos usuários (responder desafios, desbloquear conquistas). Além do registro, esse padrão viabiliza a reversão de determinadas ações, simulando a funcionalidade de *undo*.

2.3 Padrões Estruturais

- **Decorator** (Bônus de Recompensas)
Usado para adicionar dinamicamente benefícios às conquistas, como *streak* (sequência de acertos) ou *double XP*. O padrão evita duplicação de código e permite compor diferentes combinações de bônus.
- **Composite** (Hierarquia de Medalhas)
Permite organizar conquistas individuais (medalhas) em conjuntos maiores (coleções de medalhas). Dessa forma, mantém-se a uniformidade no tratamento de conquistas simples e complexas.
- **Facade** + **Adapter** (Relatórios)
O Facade oferece uma interface simplificada para centralizar a geração de relatórios, enquanto o Adapter traduz o formato de dados uma API externa para o formato utilizado internamente. Assim, o sistema fornece flexibilidade sem sobrecarregar o usuário com detalhes técnicos.

3. ANÁLISE DE COMO OS PRINCÍPIOS SOLID FORAM CONTEMPLADOS

- **S** (Single Responsibility Principle)
Cada classe foi projetada para possuir apenas uma responsabilidade clara. Por exemplo,

a classe Usuario cuida das informações pessoais, enquanto a classe Sessao controla apenas o gerenciamento de login/logout.

- **O (Open/Closed Principle)**
O sistema está aberto para extensão e fechado para modificação. Novos tipos de desafios, estratégias de pontuação ou conquistas podem ser adicionados sem alterar classes existentes, apenas criando novas implementações.
- **L (Liskov Substitution Principle)**
As subclasses de Usuario (Aluno, Professor, Visitante) podem substituir a superclasse sem quebrar a lógica do sistema. Isso reforça a consistência do modelo de herança aplicado.
- **I (Interface Segregation Principle)**
Interfaces foram segmentadas para atender a responsabilidades específicas, como responder desafios ou exportar relatórios. Essa separação evita que classes implementem métodos que não utilizam.
- **D (Dependency Inversion Principle)**
O sistema depende de abstrações e não de implementações concretas. Exemplo: o módulo de pontuação opera sobre a interface EstrategiaPontuacao, permitindo injetar dinamicamente diferentes algoritmos.

4. TABELA DE TRADE-OFFS NOS PADRÕES USADOS

Durante o desenvolvimento, algumas escolhas técnicas trouxeram benefícios, mas também limitações, em quesito de usabilidade por exemplo, console como interface simplificada, fácil para protótipo, mas limitada, persistência apenas em memória (não há banco de dados) e o uso de padrões pode aumentar a complexidade para algo que, em um sistema pequeno, poderia ser resolvido de forma direta. Mas além dessas características segue abaixo na tabela benefícios e limitações identificadas de acordo com os padrões.

Padrão/Decisão	Benefício	Limitação/Trade-off
----------------	-----------	---------------------

Singleton (Sessão)	Garante controle único da sessão, evita inconsistência de múltiplos logins no console.	Restringe a possibilidade de múltiplos usuários simultâneos, inviável para web ou multiusuário.
Factory Method (Usuários)	Facilita criação de novos tipos de usuários sem modificar código existente; promove extensibilidade.	Aumenta o número de classes e complexidade da hierarquia, podendo ser excessivo para poucos tipos de usuários.
Strategy (Pontuação)	Permite múltiplas estratégias de pontuação, extensível e flexível.	Introduz camadas extras de abstração; aumenta complexidade do código.
Observer (Notificações)	Desacopla lógica de notificação de conquistas do núcleo de execução.	Pode gerar overhead se houver muitos observadores ou eventos em sistemas maiores.
Command (Histórico/Undo)	Permite desfazer ações e rastrear histórico do usuário; aumenta robustez do sistema.	Requer armazenamento de estado de comandos; complexidade adicional para simples protótipo.
Decorator (Bônus de Conquistas)	Permite combinar dinamicamente diferentes bônus sem modificar classes existentes.	Pode dificultar depuração e rastreamento se houver múltiplos decorators encadeados.
Composite (Hierarquia de Medalhas)	Uniformiza tratamento de conquistas simples e compostas.	Estrutura de árvore adiciona complexidade; pouco impacto em sistemas pequenos.

Facade + Adapter (Relatórios)	Simplifica a interface de exportação e permite múltiplos formatos sem alterar núcleo.	Adiciona camada extra de abstração; overhead para funcionalidades simples de exportação.
--	---	--

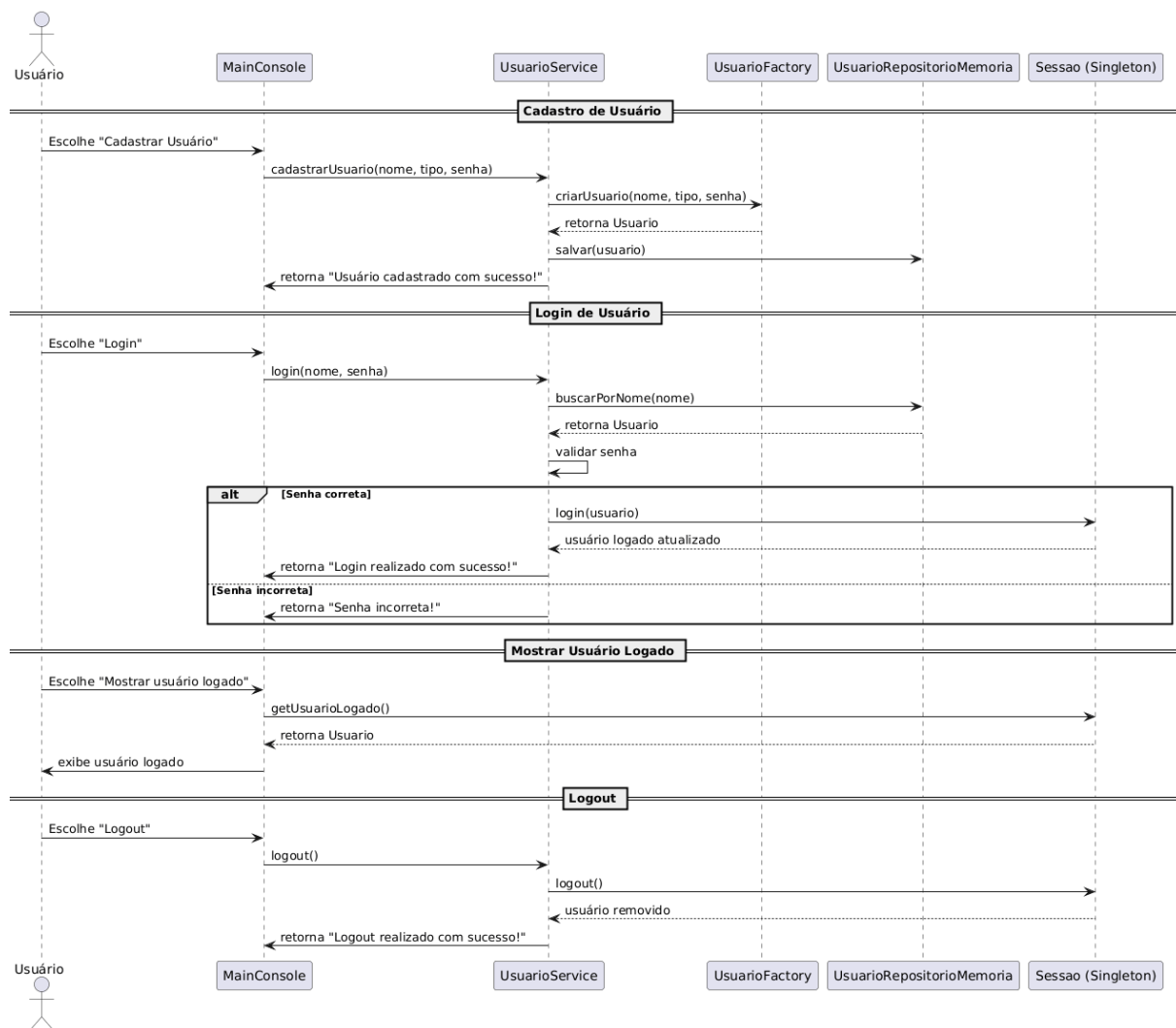
5. POSSÍVEIS EXTENSÕES FUTURAS

A arquitetura proposta foi desenhada com extensibilidade em mente, permitindo evoluções como:

- Migração para plataforma web ou aplicativo mobile, aumentando a acessibilidade.
- Implementação de banco de dados relacional para persistência de usuários, desafios e conquistas.
- Inclusão de novos tipos de desafios, como programação ou simulações práticas.
- Integração com APIs externas de relatórios e exportação avançada.
- Aplicação de recursos de inteligência artificial para sugerir desafios personalizados.

6. DIAGRAMAS UML

6.1 Diagrama Fluxo de gerenciamento de usuários e perfis.



7. CONCLUSÃO

Através do desenvolvimento da plataforma gamificada de aprendizagem foi possível consolidar praticas fundamentais para uma boa padronização de software, como destacado durante a construção deste relatório, incluindo padrões de projeto GOF e princípios SOLID. O trabalho em equipe, com a definição das tarefas de forma especifica, em módulos, possibilitou a especialização e que cada parte foi implementada de forma coesa e alinhada.

Lembrando que, a adoção de uma interface em console, embora simples, permitiu demonstrar todas funcionalidades essenciais: cadastros de usuários, autenticação, criação e resposta de desafios, registro de histórico e conquistas e exportação de relatórios. No entanto

foram identificadas limitações referente a persistência em memória, interface, restrição de múltiplos usuários simultaneamente por conta do Singleton dentre outras limitações.

Entretanto, as limitações identificadas foram aceitáveis, tendo em vista que o objetivo da atividade foi contemplado bem como o aprendizado e implementação dos padrões destacados. Como perspectivas futuras, o sistema pode ser estendido para interfaces gráficas ou web, integração com bancos de dados e APIs externas, múltiplas sessões simultâneas e novos tipos de desafios e conquistas. Tais extensões permitirão transformar a plataforma em um ambiente educacional completo, escalável e mais próximo da realidade de uso em contextos reais de aprendizagem.

Portanto, o projeto consolidou o conhecimento teórico em projeto de software orientado a objetos, demonstrando a importância de padrões de projeto bem aplicados, princípios SOLID e organização modular no desenvolvimento de sistemas complexos, mesmo em um protótipo de console.