

# Proposta de Exercício (avaliado) — TDD para Concessão de Badges por Missão

Disciplina: POO – 2025/2

Foco: **Regra de domínio** de concessão de badges no contexto acadêmico/gamificado (**ClassHero**), implementada por **TDD** (Red → Green → Refactor)

Linguagem/Runtimee: C# 12 / .NET 9

Entrega: repositório Git com testes + domínio + README (guia rápido)

## Objetivos de Aprendizagem



### Modelar por TDD

Modelar por **TDD** a regra de **concessão de badges por missão** (gamificação acadêmica).



### Garantir Invariantes

Garantir **invariantes**: *idempotência, unicidade por missão e estudante, janelas de bônus, pré-requisitos e rastreamento (auditoria).*



### Aplicar Boas Práticas

Aplicar **navegabilidade mínima, tipos nuláveis/não-nuláveis, exceções de domínio e design por intenção** (métodos do agregado/serviço de domínio).

## Contexto do Problema (domínio ClassHero)

Quando um estudante conclui uma **missão** de um **tema**, o sistema pode conceder uma **badge** (insígnia) e opcionalmente **XP bônus** conforme janelas temporais do tema. Esta concessão deve ser **determinística, atômica e registrada** para fins de ranking e histórico (classes rewards\_badge e rewards\_xp que devem ser implementadas).

# Regras de Alto Nível

01

## Unicidade

A mesma *badge* não pode ser concedida mais de uma vez para o **mesmo estudante no mesmo tema/missão**.

03

## Janelas de Bônus

Usar campos do tema que podem adotados no projeto: `bonusStartDate`, `bonusFullWeightEndDate`, `bonusFinalDate` (memória do projeto).

05

## Atomicidade

Badge e XP (quando houver) são **persistidos juntos** ou nada é gravado (transação).

02

## Elegibilidade

Só é elegível quem **concluiu a missão** (status de conclusão verdadeiro).

04

## Idempotência

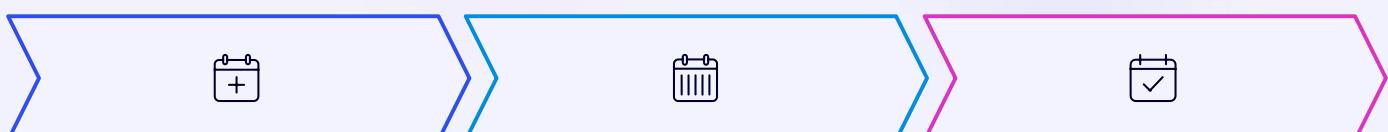
Repetição de requisições idênticas **não duplica** concessões (usar `requestId` opcional ou checagem por chave natural).

06

## Auditoria

Registrar **quem, o quê, quando, por quê** (ex.: fonte = "conclusão de missão").

# Política de Janelas de Bônus



### Bônus Integral

Até  
`bonusFullWeightEndDate`:  
concede **bônus integral**

### Bônus Reduzido

Entre  
`bonusFullWeightEndDate` e  
`bonusFinalDate`: concede  
**bônus reduzido**

### Sem Bônus

Após `bonusFinalDate`: **sem bônus** (a badge ainda pode ser concedida, se elegível)

- ❑ **Política de Rejeição:** se não elegível ou já concedida, retornar **erro de domínio** claro (sem efeitos colaterais).

**Navegabilidade mínima:** expor apenas Student → AwardBadge(missionId, now, requestId?) (ou um **Domain Service**), evitando referências bidirecionais até que um teste justifique.

**Observação:** não implemente persistência real (EF/DB). Use *ports* (interfaces) falsificáveis para testes. O foco é o **domínio puro** e a validade das regras.

# Escopo Técnico

## Projetos

- Projeto **.Domain** (classe library) com entidades/agregados/serviço de domínio e exceções.
- Projeto **.Domain.Tests** (xUnit ou NUnit) com a suíte de testes.
- Nullable habilitado**, VOs imutáveis para valores como XpAmount e BadgeSlug.

## Ports (Interfaces)

- IAwardsReadStore (consulta: já ganhou? missão concluída? datas do tema?)
- IAwardsWriteStore (gravação atômica: badge + XP + log)
- Transação** modelada via IAwardsUnitOfWork (ou método atômico no IAwardsWriteStore).

## Backlog de Testes (mínimo obrigatório)

### Concessão básica

- ConcederBadge\_quando\_missao\_concluida\_concede\_uma\_unica\_vez()
- RepetirConcessao\_da\_mesma\_badge\_para\_mesmo\_estudante\_deve\_ser\_idempotente()

### Elegibilidade

- ConcederBadge\_sem\_concluir\_missao\_deve\_falhar()
- ConcederBadge\_para\_missao\_inexistente\_deve\_falhar()

### Janelas de bônus (datas do tema)

- ConcederBadge\_ate\_bonusFullWeightEndDate\_concede\_bonus\_integral()
- ConcederBadge\_entre\_fullWeight\_e\_finalDate\_concede\_bonus\_reduzido()
- ConcederBadge\_apos\_bonusFinalDate\_nao\_concede\_bonus\_mas\_concede\_badge()

## Atomicidade e auditoria

- ConcederBadge\_falha\_na\_gravacao\_nao\_deve\_gerar\_efeitos\_parciais()
- ConcederBadge\_registro\_auditoria\_com\_contexto\_da\_acao()

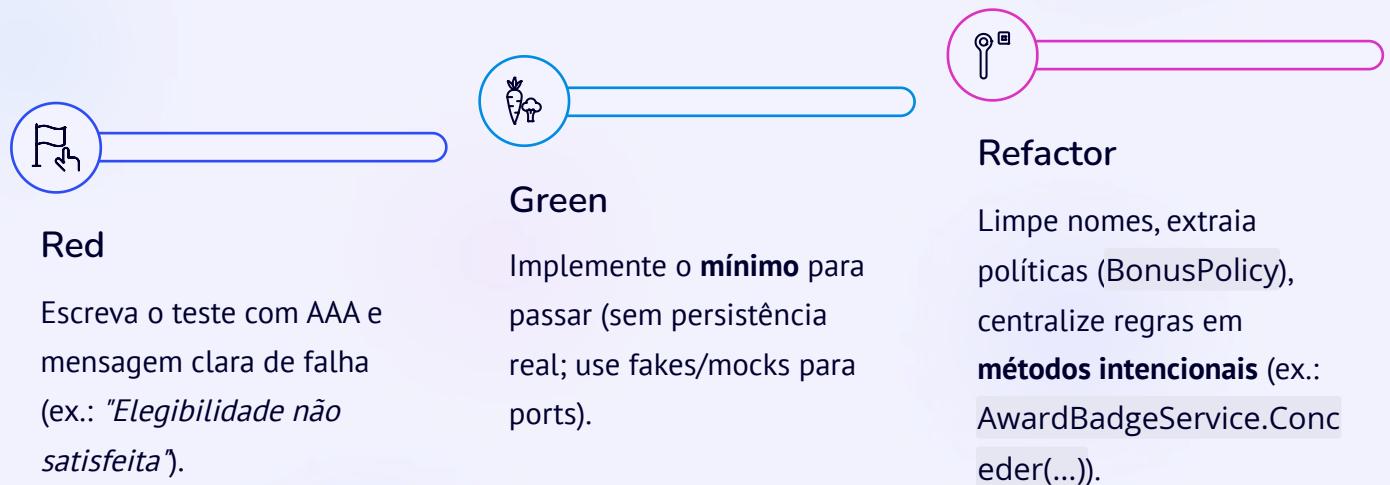
## Idempotência

- ConcederBadge\_requisicao\_repetida\_com\_requestId\_igual\_nao\_duplica()
- ConcederBadge\_requisicao\_sem\_requestId\_usa\_chave\_natural\_para\_evitar\_duplicacao()

## Naveabilidade mínima / design por intenção

- API\_deDominio\_nao\_expose\_setters\_perigosos\_para\_manipular\_recompensas()

## Roteiro TDD (por história)



## Checklist por teste:

- Nome descreve regra
- Um único motivo de falha
- Sem dependências externas
- Assert essencial + mensagens de exceção úteis
- Cenários de *edge cases* cobertos

# Regras e Políticas (detalhe)

## Chave Natural

**Chave natural** da concessão: (studentId, themeId, missionId, badgeSlug) garante unicidade.

## BonusPolicy (pura)

**Entrada:** now, bonusStartDate, bonusFullWeightEndDate, bonusFinalDate, xpBase, xpFullWeight, xpReducedWeight.

**Saída:** XpAmount (0, reduzido, integral) + *justificativa* para auditoria.

## Elegibilidade

IAwardsReadStore.MissaoConcluida(studentId, missionId) precisa ser **true**.

## Idempotência

Se requestId informado, registrar/checar em rewards\_badge (ou store idempotente) e retornar o primeiro resultado.

Sem requestId, usar a **chave natural** para prevenir duplicidade.

## Atomicidade

IAwardsWriteStore.SalvarConcessaoAtomicamente(badge, xp?, log); qualquer falha deve **anular tudo**.

## Auditoria

source = 'mission\_completion', reason = BonusPolicy.Justificativa.

# Casos-Limite (testes sugeridos)

- Datas Inconsistentes

Datas **nulas** ou inconsistentes (ex.: bonusStartDate > bonusFinalDate) → falha de configuração.

- Mudança de Relógio

**Mudança de relógio** (horário de verão): política deve aceitar DateTimeOffset e comparar por UTC.

- Conclusão Tardia

Missão concluída **após** bonusFinalDate: sem bônus.

- Múltiplas Badges

Tentativa de conceder **duas badges diferentes** para a **mesma missão**: permitido **se** as regras definirem badges distintas (cubra com teste se for um requisito futuro).

- Reprocessamento

Reprocessamento de eventos (webhook/retry) → **idempotência garante** ausência de duplicações.

## Estrutura de Código (sugestão)

### Diretório /src

```
/src
Gamification.Domain
Awards/
  AwardBadgeService.cs
  Policies/BonusPolicy.cs
  Ports/IAwardsReadStore.cs
  Ports/IAwardsWriteStore.cs
  Ports/IAwardsUnitOfWork.cs
Model/
  Badge.cs (slug, title)
  RewardLog.cs
  XpAmount.cs (VO)
```

### Diretório /tests

```
/tests
Gamification.Domain.Tests
AwardBadgeServiceTests.cs
BonusPolicyTests.cs
```

## Exemplos de Títulos de Teste (com Gherkin resumido)

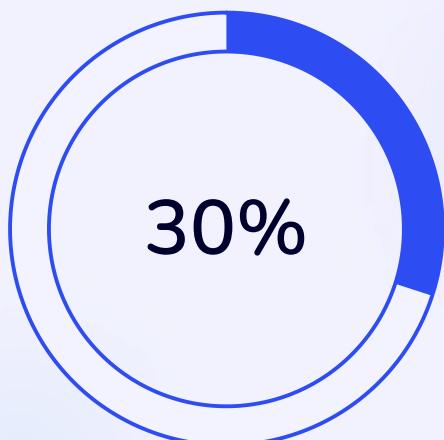
### Teste 1: Concessão Única

```
ConcederBadge_quando_missao_concluida_concede_uma_unica_vez
Given estudante concluiu missão
When solicitar concessão da badge
Then deve gravar badge uma única vez e retornar sucesso
```

## Teste 2: Bônus Integral

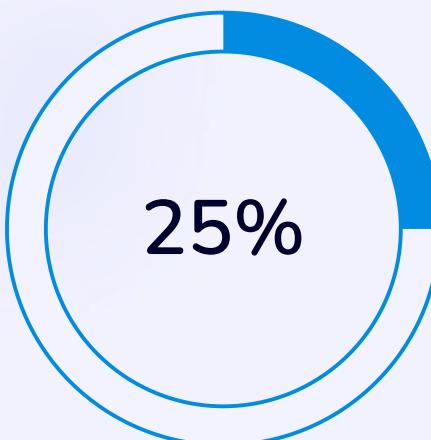
```
ConcederBadge_ate_bonusFullWeightEndDate_concede_bonus_integral  
Given agora < bonusFullWeightEndDate  
When conceder  
Then XP = xpFullWeight e log com razão "janela integral"
```

## Rubrica de Avaliação (0–100)



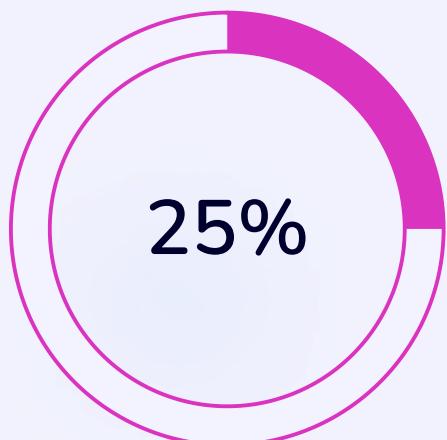
### Cobertura de Regras

Unicidade, elegibilidade, janelas, idempotência, atomicidade, auditoria



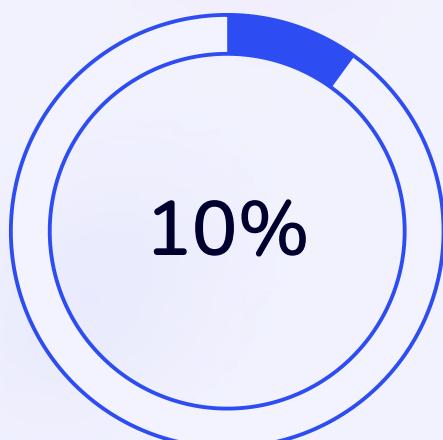
### Qualidade dos Testes

AAA/Gherkin, clareza, independência, asserts essenciais



### Domínio Correto

Políticas puras, portos/serviços, nullability, mensagens de erro significativas



### Qualidade de Código

Nomes, encapsulamento, imutabilidade útil, acoplamento baixo



### README

Como rodar testes, decisões de design, limitações

# Entregáveis e Validação (LLM-assistida)

**PR** com testes + domínio + README.

01

## Execução

Execução dotnet test.

02

## Check Automatizado

Check automatizado + **validação LLM** verificando:

- alinhamento entre **nomes dos testes e regras**;
- presença de cenários de janelas de bônus;
- idempotência e atomicidade descritas nos testes.

## Guia de Partida Rápida

1

```
mkdir gamification && cd gamification
```

2

```
mkdir src tests
```

3

```
dotnet new sln -n Gamification
```

4

```
dotnet new classlib -n  
Gamification.Domain -o  
src/Gamification.Domain framework  
net8.0
```

5

```
dotnet new xunit -n  
Gamification.Domain.Tests -o  
tests/Gamification.Domain.Tests
```

6

Referencie projetos e **habilite Nullable** no .csproj.

7

Comece pelo teste:  
ConcederBadge\_quando\_missao\_concluida\_concede\_uma\_unica\_vez →  
Red/Green/Refactor.

8

Em seguida, cubra **janelas de bônus** e **idempotência**.

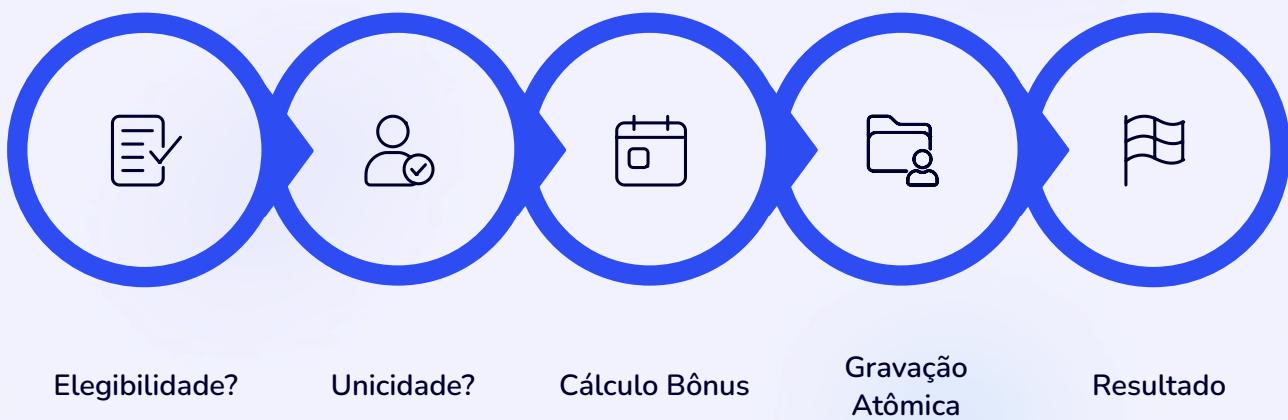
# Dica Final

## Commits pequenos por regra

Mensagens claras (ex.: feat(domain): bônus integral até FullWeightEndDate).



## Fluxo de Concessão de Badge



Este diagrama ilustra o processo completo desde a solicitação até a persistência atômica dos dados.

# Arquitetura de Ports e Adapters

## Domínio

- AwardBadgeService
- BonusPolicy
- Badge
- XpAmount
- RewardLog

## Ports (Interfaces)

- IAwardsReadStore
- IAwardsWriteStore
- IAwardsUnitOfWork

## Testes

- Fakes/Mocks
- Cenários AAA
- Asserts essenciais
- Independência

A arquitetura hexagonal permite testar o domínio de forma isolada, sem dependências de infraestrutura.

