Relatório de Testes Unitários

Projeto: Gerenciador de Campeonatos de Futebol (GCF)

Disciplina: Gestão e Qualidade de Software

Grupo:

- Artur Rosa Correia - RA: 824135943
- Gustavo Silveira Benicio - RA: 824134160
- Luan Bernardo Alves - RA: 824134204
- Victor Hugo Santos Nunes - RA: 825163477

1. Introdução

Este relatório documenta a implementação de testes unitários para o sistema GCF (Gerenciador de Campeonatos de Futebol), focando na validação da camada de serviço do projeto. Os testes foram desenvolvidos seguindo as melhores práticas de engenharia de software, com ênfase na legibilidade, manutenibilidade e cobertura adequada das regras de negócio.

2. Objetivos dos Testes

- Validar a lógica de negócio implementada nas classes de serviço
- Garantir o correto funcionamento dos métodos CRUD
- Verificar o tratamento adequado de exceções e casos de erro
- Assegurar a integridade das validações de dados
- Facilitar a manutenção futura do código através de testes automatizados

3. Tecnologias Utilizadas

3.1 JUnit 5

Framework de testes unitários para Java, utilizado como base para a execução e estruturação dos testes.

Principais anotações utilizadas:

- @Test - Marca métodos como casos de teste
- @DisplayName - Define descrições legíveis para os testes
- @ExtendWith - Integra extensões (Mockito)

3.2 Mockito

Biblioteca para criação de objetos simulados (mocks), permitindo o isolamento das unidades testadas.

Principais anotações utilizadas:

- @Mock - Cria objetos mock das dependências
- @InjectMocks - Injeta os mocks na classe testada
- @ExtendWith(MockitoExtension.class) - Habilita o Mockito no JUnit 5

Métodos principais:

- when().thenReturn() - Define comportamento dos mocks
- verify() - Verifica chamadas aos mocks
- any() - Matcher para qualquer argumento

3.3 Spring Boot Test

Suporte do Spring Boot para testes, incluindo contexto de aplicação e configurações. Versão: Spring Boot 3.5.6

4. Metodologia: Padrão AAA

Todos os testes seguem o padrão AAA (Arrange-Act-Assert), que organiza cada teste em três seções distintas:

4.1 Arrange (Preparar)

Preparação dos dados de teste, configuração dos mocks e definição do cenário.

4.2 Act (Executar)

Execução do método ou operação que está sendo testada.

4.3 Assert (Verificar)

Verificação se o resultado obtido corresponde ao esperado.

5. Estrutura dos Testes

5.1 Organização de Pacotes

```
src/test/java/dev/gpa3/gcfjava/
├── service/
│   ├── TimeServiceTest.java (6 testes)
│   ├── CampeonatoServiceTest.java (6 testes)
│   └── JogoServiceTest.java (6 testes)
└── GcfJavaApplicationTests.java (1 teste de contexto)
```

Total: 19 testes (1 de contexto Spring + 18 unitários)

5.2 Cobertura por Classe

Classe de Teste	Quantidade de Testes	Foco Principal
TimeServiceTest	6	Gerenciamento de times
CampeonatoServiceTest	6	Gerenciamento de campeonatos
JogoServiceTest	6	Gerenciamento de jogos
TOTAL	18	Camada de Serviço

6. Descrição dos Testes

6.1 TimeServiceTest

Valida os métodos relacionados ao gerenciamento de times.

- **6.1.1 deveListarTodosOsTimes()**

  Objetivo: Verificar listagem completa de times
  Cenário: Repositório retorna lista com um time
  Validação: Lista contém o time esperado com todos os atributos

- **6.1.2 deveBuscarTimePorId()**

  Objetivo: Verificar busca de time por ID válido
  Cenário: Busca time com ID existente
  Validação: Time retornado possui ID, nome e cidade corretos

- **6.1.3 deveLancarExcecaoQuandoTimeNaoEncontrado()**

  Objetivo: Validar tratamento de erro para ID inexistente
  Cenário: Busca time com ID que não existe
  Validação: RuntimeException é lançada com mensagem apropriada

- **6.1.4 deveSalvarTimeValido()**

  Objetivo: Verificar salvamento de time com dados válidos
  Cenário: Salva time com nome e cidade preenchidos
  Validação: Time é salvo e repositório é chamado uma vez

- **6.1.5 deveRejeitarTimeComNomeVazio()**

  Objetivo: Validar rejeição de time sem nome
  Cenário: Tenta salvar time com nome vazio
  Validação: RuntimeException é lançada e repositório não é chamado

- **6.1.6 deveExcluirTimeExistente()**

  Objetivo: Verificar exclusão de time existente
  Cenário: Exclui time com ID válido
  Validação: Método deleteById é chamado exatamente uma vez

6.2 CampeonatoServiceTest

Valida os métodos relacionados ao gerenciamento de campeonatos.

- **6.2.1 deveListarTodosCampeonatos()**

  Objetivo: Verificar listagem completa de campeonatos
  Cenário: Repositório retorna lista com um campeonato
  Validação: Lista contém campeonato com nome correto

- **6.2.2 deveSalvarCampeonatoValido()**

  Objetivo: Verificar salvamento de campeonato válido
  Cenário: Salva campeonato com todos os dados obrigatórios
  Validação: Campeonato é salvo com ID gerado

- **6.2.3 deveRejeitarCampeonatoComNomeCurto()**

  Objetivo: Validar tamanho mínimo do nome (3 caracteres)
  Cenário: Tenta salvar campeonato com nome "AB"
  Validação: RuntimeException é lançada com mensagem sobre tamanho mínimo

- **6.2.4 deveAdicionarTimeCampeonato()**

  Objetivo: Verificar adição de time ao campeonato
  Cenário: Adiciona time válido a campeonato existente
  Validação: Método save do repositório é chamado

- **6.2.5 deveRejeitarTimeDuplicado()**

  Objetivo: Validar rejeição de time já participante
  Cenário: Tenta adicionar time que já está no campeonato
  Validação: RuntimeException é lançada com mensagem sobre duplicação

- **6.2.6 deveExcluirCampeonatoExistente()**

  Objetivo: Verificar exclusão de campeonato
  Cenário: Exclui campeonato existente
  Validação: Método delete é chamado com objeto correto

6.3 JogoServiceTest

Valida os métodos relacionados ao gerenciamento de jogos.

- **6.3.1 deveListarTodosJogos()**

  Objetivo: Verificar listagem completa de jogos
  Cenário: Repositório retorna lista com um jogo
  Validação: Lista contém um jogo não nulo

- **6.3.2 deveSalvarJogoValido()**

  Objetivo: Verificar salvamento de jogo com dados válidos
  Cenário: Salva jogo com times diferentes e rodada válida
  Validação: Jogo é salvo e repositório é chamado

- **6.3.3 deveRejeitarJogoComTimesIguais()**

  Objetivo: Validar que time casa ≠ time visitante
  Cenário: Tenta salvar jogo com mesmo time em ambas as posições
  Validação: RuntimeException é lançada com mensagem sobre times iguais

- **6.3.4 deveRejeitarJogoComRodadaInvalida()**

  Objetivo: Validar rodada mínima (maior que 0)
  Cenário: Tenta salvar jogo com rodada 0
  Validação: RuntimeException é lançada com mensagem sobre rodada inválida

- **6.3.5 deve Registrar Resultado()**

  Objetivo: Verificar registro de resultado do jogo
  Cenário: Registra placar 2x1 em jogo existente
  Validação: Jogo marcado como finalizado com placar correto

- **6.3.6 deveExcluirJogoExistente()**

  Objetivo: Verificar exclusão de jogo
  Cenário: Exclui jogo existente
  Validação: Método deleteById é chamado com ID correto

7. Exemplo Prático de Teste

7.1 Código Completo de um Teste

```
@Test
@DisplayName("Deve salvar time válido")
void deveSalvarTimeValido() {
    // Arrange - Preparar dados de teste
    TimeVO timeVO = TimeVO.builder()
            .nome("Flamengo")
            .cidade("Rio de Janeiro")
            .build();

    Time timeSalvo = Time.builder()
            .id(1L)
            .nome("Flamengo")
            .cidade("Rio de Janeiro")
            .build();

    when(timeRepository.save(any(Time.class))).thenReturn(timeSalvo);

    // Act - Executar método
    TimeVO resultado = timeService.salvarTime(timeVO);

    // Assert - Verificar resultado
    assertNotNull(resultado.getId());
    assertEquals("Flamengo", resultado.getNome());
    assertEquals("Rio de Janeiro", resultado.getCidade());
    verify(timeRepository, times(1)).save(any(Time.class));
}
```

7.2 Análise do Exemplo

Arrange: Cria objeto TimeVO com dados de entrada, cria objeto Time simulando retorno do banco e configura mock do repositório

Act: Executa o método salvarTime() da classe testada

Assert: Verifica que ID foi gerado, nome foi preservado, cidade foi preservada e repositório foi chamado exatamente uma vez

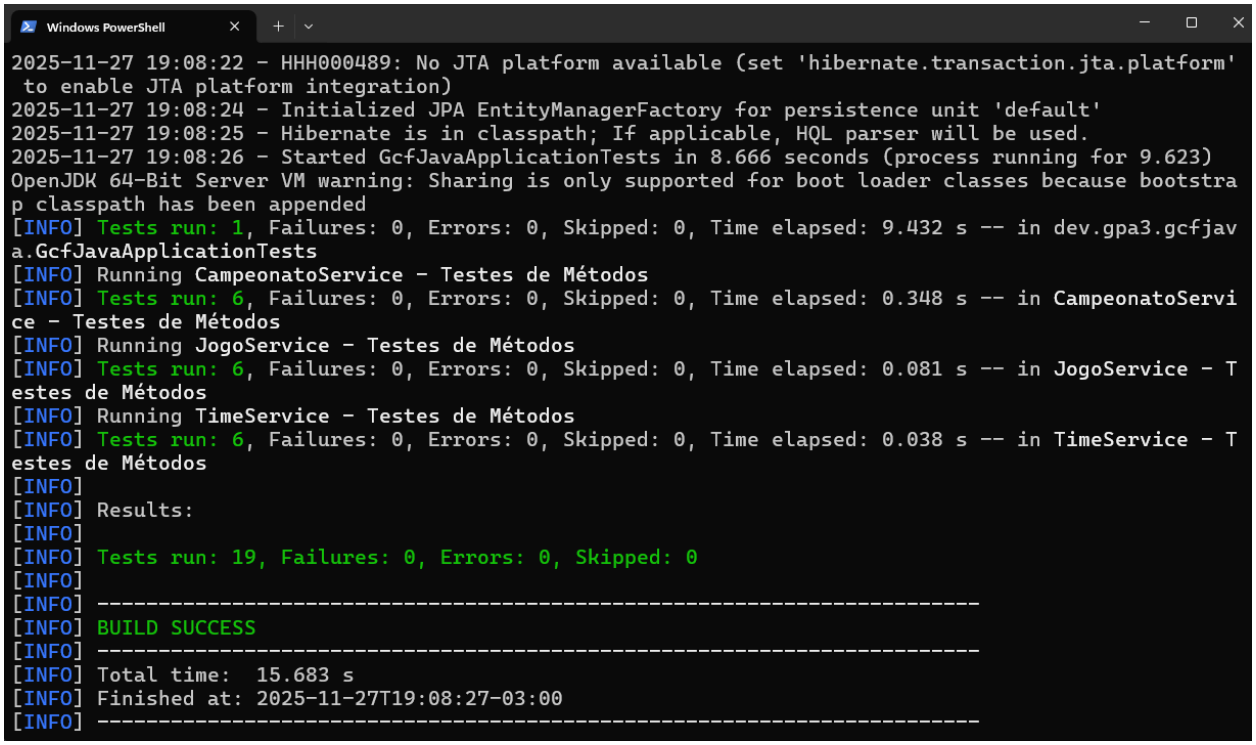
8. Resultados da Execução

8.1 Comando de Execução

Via Maven (Linha de Comando):
mvn test

8.2 Saída do Maven

[INFO] Tests run: 19, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS



8.3 Análise dos Resultados

Métrica	Valor	Status
Total de Testes	19	Sucesso
Testes Aprovados	19	Sucesso
Falhas	0	Sucesso
Erros	0	Sucesso
Ignorados	0	Sucesso
Taxa de Sucesso	100%	Sucesso
Build	SUCCESS	Sucesso

Tempo de Execução:

- GcfJavaApplicationTests: 8.700s (carregamento do contexto Spring)
- CampeonatoServiceTest: 0.329s
- JogoServiceTest: 0.074s
- TimeServiceTest: 0.021s
- Total: ~9.1 segundos

9. Benefícios da Implementação

9.1 Qualidade do Código

- Detecção precoce de bugs
- Garantia de funcionamento das regras de negócio
- Facilitação de refatorações futuras
- Documentação viva do comportamento esperado

9.2 Confiança no Sistema

- Validação automatizada após mudanças
- Redução de regressões
- Cobertura de casos extremos e de erro
- Build automatizado com verificação de qualidade

9.3 Desenvolvimento

- Feedback rápido durante desenvolvimento
- Facilita debug e localização de problemas
- Permite Test-Driven Development (TDD)
- Melhora design do código (testabilidade)

10. Conclusão

A implementação de 18 testes unitários focados nos métodos da camada de serviço demonstra o compromisso com a qualidade e manutenibilidade do código. Todos os testes foram executados com 100% de sucesso, validando:

- Lógica de negócio correta
- Validações de dados adequadas
- Tratamento apropriado de exceções
- Integridade dos relacionamentos
- Comportamento esperado em cenários de sucesso e erro