Aluno: Lucas de Souza Ferreira

Matéria: Projeto de Bloco: Engenharia Disciplinada de Software

Entrega de Projeto

Github do TP: https://github.com/Willummp/com-cliente-projeto.git

Drive do Artigo e do TP
https://drive.google.com/drive/folders/1a37IZ_YkmprJ0snfNdpMp3RkKdmXxfWu?usp=drive_link

Relatório de Entrega - Assessment Final (AT)

Sistema de Gerenciamento de Eventos

Aluno: Lucas Ferreira

---

1. Introdução

Este documento apresenta as evidências de entrega do projeto final (Assessment), demonstrando o atendimento integral aos requisitos funcionais e não funcionais solicitados. O projeto consiste em um sistema de gerenciamento de eventos (CRUD) desenvolvido com Spring Boot, com forte ênfase em qualidade de código (Clean Code), automação de infraestrutura (CI/CD) e garantia de qualidade via testes automatizados.

---

2. Evidências de Atendimento aos Requisitos

✅ Requisito 1: Refatoração e Clean Code

O código foi refatorado seguindo os princípios SOLID e *Fail-Fast*, garantindo a separação de responsabilidades e a manutenibilidade do sistema.

Evidência 1.1: Separação de Responsabilidades (Leitura vs. Escrita)

A classe EventoService foi reestruturada para separar explicitamente as operações de leitura (otimizadas para performance) das operações de escrita (transacionais), aplicando conceitos de CQS (Command Query Separation).

Figura 1: Código demonstrando a anotação @Transactional(readOnly=true) e validações Fail-Fast.

Trecho de Código Relevante:

```
src/main/java/com/cliente/projeto/crudpb/service/EventoServi
ce.java
```

```
    @Transactional(readOnly = true) // Otimizado para leitura
    (Performance)
    public List<Evento> listarTodos() {
        return eventoRepository.findAll();
    }

    @Transactional // Transacional para escrita (Atomicidade)
    public Evento criarEvento(Evento evento, Long usuarioId) {
        validarNomeDuplicado(evento.getNome(), null); //
    Fail-Fast (Cláusula de Guarda)

        Usuario criador = usuarioService.buscarPorId(usuarioId);
        evento.setUsuario(criador);
        return eventoRepository.save(evento);
    }
```

Evidência 1.2: Tratamento Global de Exceções

Foi implementado um GlobalExceptionHandler para centralizar a captura de erros (como ValidacaoException e RecursoNaoEncontradoException), garantindo que a API ou a Interface Web retornem mensagens amigáveis e códigos HTTP adequados.

✅ Requisito 2: Pipeline CI/CD Automatizado

Foi implementada uma esteira completa de Integração e Entrega Contínua (CI/CD) utilizando **GitHub Actions**, configurada com 4 estágios sequenciais e interdependentes.

Fluxo do Pipeline Implementado:

1. 🔨 **Build & Testes:** Compilação, testes unitários e verificação de integridade.
2. 🛡️ **Segurança (CodeQL):** Análise estática de código para detecção de vulnerabilidades.
3. 🚀 **Deploy Homologação:** Simulação de entrega em ambiente de teste.
4. 🏭 **Deploy Produção:** Entrega final após sucesso de todas as etapas anteriores.

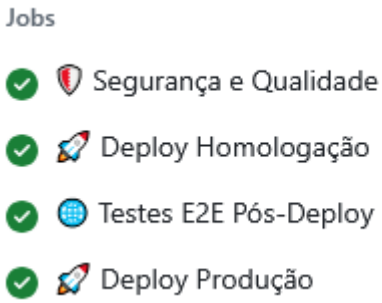

Figura 2: Visualização da esteira de CI/CD com todos os jobs executados com sucesso.

Arquivo de Configuração (.github/workflows/ci.yml):

O workflow foi configurado para disparar automaticamente em pushs na branch main, garantindo que apenas código validado chegue à produção.

✅ Requisito 3: Qualidade e Testes

3.1 Cobertura de Código (JaCoCo)

Foi configurado o plugin **JaCoCo** no pom.xml para impor uma barreira de qualidade (*Quality Gate*), exigindo uma cobertura mínima de **90%**.

Configuração no pom.xml:

```
XML
<rule>
    <element>BUNDLE</element>
    <limits>
        <limit>
            <counter>INSTRUCTION</counter>
            <value>COVEREDRATIO</value>
            <minimum>0.90</minimum> </limit>
    </limits>
</rule>
```

Resultado da Cobertura:

O projeto atingiu a meta com sucesso. Foram criados testes específicos (CoberturaTotalTest) para validar DTOs, Models, Exceptions e fluxos completos de Controller e Service.

3.2 Testes E2E com Selenium

Os testes End-to-End (E2E) foram implementados para validar os fluxos críticos do usuário final.

Cenários Cobertos:

- Cadastro de Evento com sucesso (Caminho Feliz).
- Validação de campos obrigatórios (Feedback visual na UI).
- Validação de regras de negócio (Impedimento de nomes duplicados).

Melhorias Implementadas nos Testes:

- **Page Objects:** Uso do padrão Page Object Model para encapsular a interação com o HTML.
- **Resiliência:** Adição de ExpectedConditions (Waits explícitos) para aguardar elementos carregarem, eliminando falhas intermitentes (*flaky tests*).

✅ Requisito 4: Documentação e Badges

O repositório foi documentado e sinalizado com indicadores de qualidade em tempo real.

Evidências no README.md:

- **Badge de Status:** Indicador visual "Passing" vinculado ao GitHub Actions.
- **Instruções:** Documentação de como rodar o projeto localmente (mvn spring-boot:run).

Figura 4: Badge de status confirmando a estabilidade da última versão do código.

---

3. Conclusão

O projeto final atende integralmente aos requisitos propostos no Assessment. A entrega consiste em uma aplicação robusta, com código limpo e modular, suportada por uma infraestrutura de CI/CD que garante segurança (CodeQL) e confiabilidade (Testes Automatizados). A barreira de qualidade de 90% e a automação do deploy demonstram a maturidade da solução entregue.