Aluno: Lucas de Souza Ferreira

Matéria: Projeto de Bloco: Engenharia Disciplinada de Software

TP2

Teste de Performance 2 (TP2) - Relatório de Evidências

Visão Geral

Este relatório documenta a verificação dos requisitos do TP2 (Interface Web + Selenium) com base na tentativa de execução dos testes automatizados.

1. Verificação dos Requisitos

1.1 Interface Web

- Requisito: Implementar interface web para CRUD.
- Status de Verificação: ✅ Implementado.

1.2 Automação com Selenium

- Requisito: Configurar Selenium WebDriver e criar testes automatizados.
- Evidência: O código foi atualizado para utilizador FirefoxDriver.
- Status: ✅ Implementado e Configurado.

1.3 Execução de Testes

- Requisito: Validar fluxo completo e usar Page Object Model (POM).
- Resultado da Execução:
- Ambiente: Firefox Driver configurado e executando.
- Resultados: Foram executados 33 testes.
- Falhas Identificadas:
    a. Exclusão: deveExcluirEventoComSucesso falhou. O evento não desapareceu da lista (possível erro de lógica ou delay na atualização da página).
    b. Timeout: deveCadastrarEventosValidosParametrizados falhou por timeout de 10s ao esperar a lista carregar.

2. Evidências de Código (Para o PDF)

- Testes CRUD (Selenium):
- src/test/java/org/example/tp2pb/tests/EventoCrudTest.java

```
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class EventoCrudTest extends BaseSeleniumTest {

    private EventoListPage eventoListPage;

    @BeforeEach
```

```
    public void setupTest() {
        driver.get(TestConfig.BASE_URL);
        eventoListPage = new EventoListPage(driver);
    }

    @AfterEach
    public void afterEach(TestInfo testInfo) {
        // Captura screenshot se o teste falhou
        if (testInfo.getTags().contains("screenshot-on-failure")) {
            ScreenshotHelper.captureScreenshot(driver,
testInfo.getDisplayName().replaceAll("[^a-zA-Z0-9]", "_"));
        }
    }

    /**
     * Testes de cadastro de eventos.
     */
    @Nested
    @DisplayName("Testes de Cadastro")
    class TestesCadastro {

        @Test
        @Order(1)
        @DisplayName("Deve cadastrar um novo evento com sucesso")
        public void deveCadastrarEventoComSucesso() {
            // Arrange
            var dadosEvento = TestDataBuilder.eventoValido();

            // Act
            var formPage = eventoListPage.clicarAdicionarNovoEvento();
            formPage.preencherFormulario(dadosEvento.nome, dadosEvento.data,
dadosEvento.local);
            var listPageResult = formPage.submeterFormulario();

            // Assert
            assertTrue(listPageResult.isEventoPresenteNaLista(dadosEvento.nome),
                    "O evento cadastrado não foi encontrado na lista.");
            assertThat(listPageResult.getQuantidadeEventos()).isGreaterThan(0);
        }

        @Test
        @Order(2)
        @DisplayName("Deve cadastrar múltiplos eventos")
        public void deveCadastrarMultiplosEventos() {
            // Arrange
            var eventos = TestDataBuilder.multiplosEventosValidos();
            int quantidadeInicial = eventoListPage.getQuantidadeEventos();

            // Act
            for (TestDataBuilder.EventoData evento : eventos) {
                var formPage = eventoListPage.clicarAdicionarNovoEvento();
                formPage.preencherFormulario(evento.nome, evento.data,
evento.local);
                eventoListPage = formPage.submeterFormulario();
            }

            // Assert
            int quantidadeFinal = eventoListPage.getQuantidadeEventos();
            assertThat(quantidadeFinal).isEqualTo(quantidadeInicial +
eventos.length);

            // Verifica que todos os eventos foram cadastrados
            var nomesEventos = eventoListPage.getTodosNomesEventos();
            for (TestDataBuilder.EventoData evento : eventos) {
                assertTrue(nomesEventos.contains(evento.nome),
                        "Evento " + evento.nome + " não encontrado na lista");
            }
        }
    }

    /**
```

```
    * Testes de edição de eventos.
    */
   @Nested
   @DisplayName("Testes de Edição")
   @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
   class TestesEdicao {

       private final String NOME_EVENTO_ORIGINAL = "Evento Para Editar";
       private final String NOME_EVENTO_EDITADO = "Evento Editado com Sucesso";

       @Test
       @Order(1)
       @DisplayName("Preparação: Criar evento para edição")
       public void prepararEventoParaEdicao() {
           var evento = TestDataBuilder.eventoCustomizado(NOME_EVENTO_ORIGINAL,
"2025-12-15", "Sala A");
           var formPage = eventoListPage.clicarAdicionarNovoEvento();
           formPage.preencherFormulario(evento.nome, evento.data, evento.local);
           formPage.submeterFormulario();
       }

       @Test
       @Order(2)
       @DisplayName("Deve editar um evento existente com sucesso")
       public void deveEditarEventoComSucesso() {
           // Arrange - evento já deve existir do teste anterior

assertTrue(eventoListPage.isEventoPresenteNaLista(NOME_EVENTO_ORIGINAL),
                   "Evento original não encontrado");

           // Act
           var formPage = eventoListPage.editarEvento(NOME_EVENTO_ORIGINAL);
           assertTrue(formPage.isEdicao(), "Formulário deveria estar em modo de
edição");

           formPage.preencherFormulario(NOME_EVENTO_EDITADO, "2025-12-20", "Sala
B");
           var listPageResult = formPage.submeterFormulario();

           // Assert

assertTrue(listPageResult.isEventoPresenteNaLista(NOME_EVENTO_EDITADO),
                   "Evento editado não encontrado na lista");

assertFalse(listPageResult.isEventoPresenteNaLista(NOME_EVENTO_ORIGINAL),
                   "Evento original ainda aparece na lista após edição");
       }
   }

   /**
    * Testes de exclusão de eventos.
    */
   @Nested
   @DisplayName("Testes de Exclusão")
   @TestMethodOrder(MethodOrderer.OrderAnnotation.class)
   class TestesExclusao {

       private final String NOME_EVENTO_PARA_EXCLUIR = "Evento Para Excluir";

       @Test
       @Order(1)
       @DisplayName("Preparação: Criar evento para exclusão")
       public void prepararEventoParaExclusao() {
           var evento =
TestDataBuilder.eventoCustomizado(NOME_EVENTO_PARA_EXCLUIR, "2025-12-25",
"Online");
           var formPage = eventoListPage.clicarAdicionarNovoEvento();
           formPage.preencherFormulario(evento.nome, evento.data, evento.local);
           formPage.submeterFormulario();
       }
```

```
        @Test
        @Order(2)
        @DisplayName("Deve excluir um evento existente")
        public void deveExcluirEventoComSucesso() {
            // Arrange

assertTrue(eventoListPage.isEventoPresenteNaLista(NOME_EVENTO_PARA_EXCLUIR),
                    "Evento para exclusão não encontrado");
            int quantidadeAntes = eventoListPage.getQuantidadeEventos();

            // Act
            eventoListPage.excluirEvento(NOME_EVENTO_PARA_EXCLUIR);

            // Assert

assertFalse(eventoListPage.isEventoPresenteNaLista(NOME_EVENTO_PARA_EXCLUIR),
                    "O evento ainda é exibido na lista após a exclusão");
            int quantidadeDepois = eventoListPage.getQuantidadeEventos();
            assertThat(quantidadeDepois).isEqualTo(quantidadeAntes - 1);
        }
    }

    /**
     * Testes de listagem e visualização.
     */
    @Nested
    @DisplayName("Testes de Listagem")
    class TestesListagem {

        @Test
        @DisplayName("Deve exibir todos os eventos cadastrados")
        public void deveExibirTodosEventos() {
            // Act
            var nomesEventos = eventoListPage.getTodosNomesEventos();
            int quantidade = eventoListPage.getQuantidadeEventos();

            // Assert
            assertThat(nomesEventos).hasSize(quantidade);
            assertThat(quantidade).isGreaterThanOrEqualTo(0);
        }

        @Test
        @DisplayName("Deve navegar para formulário de novo evento e voltar")
        public void deveNavegarParaFormularioEVoltar() {
            // Act
            EventoFormPage formPage = eventoListPage.clicarAdicionarNovoEvento();

            // Assert
            assertNotNull(formPage);
            assertFalse(formPage.isEdicao(), "Formulário não deveria estar em modo
de edição");

            // Volta para a lista
            EventoListPage listPageResult = formPage.clicarCancelar();
            assertNotNull(listPageResult);
        }
    }
}
```

- Configuração do Driver (Firefox):
- src/test/java/org/example/tp2pb/tests/BaseSeleniumTest.java

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT)
@ActiveProfiles("test")
```

```java
@DirtiesContext(classMode = DirtiesContext.ClassMode.AFTER_CLASS)
public abstract class BaseSeleniumTest {

 protected static WebDriver driver;

 @BeforeAll
 public static void setupClass() {
   WebDriverManager.firefoxdriver().setup();
   FirefoxOptions options = new FirefoxOptions();
   options.addArguments("--headless");
   driver = new FirefoxDriver(options);
 }

 @AfterAll
 public static void teardown() {
   if (driver != null) {
     driver.quit();
   }
 }
}
```

- Lógica Web (Controller):
- src/main/java/org/example/tp2pb/controller/EventoController.java

```java
@Controller
@RequestMapping("/eventos") // Todos os métodos nesta classe começarão com
/eventos
public class EventoController {

   @Autowired // O Spring vai injetar uma instância do repositório aqui
   private EventoRepository eventoRepository;

   /**
    * Lista todos os eventos cadastrados.
    *
    * @param model modelo para passar dados à view
    * @return nome da view de listagem
    */
   // Método para LISTAR todos os eventos
   @GetMapping
   public String listarEventos(Model model) {
       model.addAttribute("eventos", eventoRepository.findAll());
       return "index"; // Retorna o arquivo templates/index.html
   }

   /**
    * Exibe formulário para cadastro de novo evento.
```

```
     *
     * @param model modelo para passar dados à view
     * @return nome da view do formulário
     */
    // Método para mostrar o FORMULÁRIO de novo evento
    @GetMapping("/novo")
    public String mostrarFormularioNovo(Model model) {
        model.addAttribute("evento", new Evento());
        return "form-evento"; // Retorna o arquivo templates/form-evento.html
    }

    /**
     * Salva um evento (novo ou existente) com validação.
     *
     * @param evento objeto evento com dados do formulário
     * @param result resultado da validação
     * @return redirect para listagem ou retorna ao formulário se houver erros
     */
    // Método para SALVAR um evento (novo ou existente)
    @PostMapping("/salvar")
    public String salvarEvento(@Valid @ModelAttribute Evento evento, BindingResult
result) {
        if (result.hasErrors()) {
            return "form-evento";
        }
        eventoRepository.save(evento);
        return "redirect:/eventos"; // Redireciona para a lista após salvar
    }

    /**
     * Exibe formulário para edição de evento existente.
     *
     * @param id    identificador do evento
     * @param model modelo para passar dados à view
     * @return nome da view do formulário
     * @throws IllegalArgumentException se evento não for encontrado
     */
    // Método para mostrar o formulário de EDIÇÃO
    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Long id, Model model) {
        Evento evento = eventoRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("ID de evento
inválido: " + id));
        model.addAttribute("evento", evento);
        return "form-evento";
    }

    /**
     * Exclui um evento pelo ID.
     *
     * @param id identificador do evento a ser excluído
     * @return redirect para listagem
```

```
     */
    // Método para EXCLUIR um evento
    @GetMapping("/excluir/{id}")
    public String excluirEvento(@PathVariable Long id) {
        eventoRepository.deleteById(id);
        return "redirect:/eventos";
    }
}
```

- Modelo de Domínio:
- src/main/java/org/example/tp2pb/model/Evento.java

```
@Entity
@Data
public class Evento {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "Nome do evento é obrigatório")
    @Size(min = 3, max = 100, message = "Nome deve ter entre 3 e 100 caracteres")
    private String nome;

    @NotNull(message = "Data do evento é obrigatória")
    @FutureOrPresent(message = "Data do evento deve ser presente ou futura")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate data;
    @NotBlank(message = "Local do evento é obrigatório")
    @Size(min = 3, max = 150, message = "Local deve ter entre 3 e 150 caracteres")
    private String local;
}
```

- Dependências e Plugins:
- pom.xml

```
<dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>

        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.31.0</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>io.github.bonigarcia</groupId>
            <artifactId>webdrivermanager</artifactId>
            <version>5.8.0</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <version>3.24.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

3. Evidências de Execução (PRINTS)

Abaixo estão os logs REAIS da execução no ambiente.

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running org.example.tp2pb.tests.EventoCrudTest
...
[ERROR] Failures:
[ERROR]
EventoCrudTest$TestesExclusao.deveExcluirEventoComSucesso:166 O
evento ainda é exibido na lista após a exclusão ==> expected: <false>
but was: <true>
[ERROR] Errors:
[ERROR]
EventoParameterizedTest.deveCadastrarEventosValidosParametrizados:49
» Timeout Expected condition failed: waiting for visibility of
element located by By.xpath: //h1[text()='Lista de Eventos']
...
[INFO] Tests run: 33, Failures: 1, Errors: 1, Skipped: 0
[INFO]
------------------------------------------------------------------------
---
[INFO] BUILD FAILURE
[INFO]
------------------------------------------------------------------------
---
```

4. Conclusão TP2

O ambiente de testes está funcional e os testes automatizados foram capazes de exercitar a aplicação e **identificar falhas (bugs ou problemas de performance/timeout).** O relatório de evidências cumpre seu papel de demonstrar a execução da suíte de testes.