22 — Testes E2E com Playwright

22 — Testes E2E com Playwright

> Guia para configurar Playwright do zero em projeto Vite + React. Consolidado a partir de uma configuração real, testada de verdade — as armadilhas na seção 11 aconteceram de fato, não são hipotéticas.

Sumário

1. Conceitos essenciais do Playwright

O Playwright Test é um framework de testes ponta a ponta (E2E) com test runner, asserções, isolamento, paralelização e um conjunto rico de ferramentas. Suporta Chromium, Firefox e WebKit (Windows/Linux/macOS/CI), com emulação móvel nativa.

- **Isolamento em 3 camadas**: `Browser` (processo do navegador, caro, 1 por worker) → `BrowserContext` (cookies/localStorage/sessionStorage/cache isolados, barato, 1 por teste) → `Page` (aba dentro do contexto, não isola nada sozinha).
- **Auto-waiting**: antes de qualquer ação (`click`, `fill`, etc.) o Playwright verifica *actionability checks* (visível, estável, habilitado...) e repete a checagem até passar ou estourar timeout — elimina a necessidade de `sleep()`. `expect(locator).toBeVisible()` é polling, não uma foto única; `expect(await x.count()).toBe(n)` é.
- **Fixtures**: injeção de dependência do Playwright (`page`, `context`, `browser`, `request` já vêm nativas). Substitui `beforeEach` repetido por "pedir só o que a receita precisa".
- **Workers**: cada worker é um processo do SO inteiro, com seu próprio browser; se um teste falha, o worker inteiro é descartado e recriado.
- **Paralelismo**: por padrão, arquivos diferentes rodam em workers diferentes; testes do mesmo arquivo rodam em sequência no mesmo worker, a menos que `fullyParallel: true`.
- **Projects**: configuração nomeada e independente (pode ter seu próprio browser/device/storageState/`testMatch`). É a base do padrão de setup/cleanup usado abaixo.

Referência oficial de boas práticas: https://playwright.dev/docs/best-practices

2. Instalação

```
npm init playwright@latest
```

O instalador pergunta:

- TypeScript ou JavaScript;
- Nome do diretório de testes;
- Adicionar GitHub Actions workflow;

- Instalar os browsers do Playwright.

Dependências relevantes (via `devDependencies`):

```
"@playwright/test": "^1.62.1",
"dotenv": "^16.4.5",
"@types/node": "^24.13.3"
```

`dotenv` é necessário porque o `playwright.config.ts` lê variáveis de um `.env` próprio dos testes (credenciais de login E2E), separado da config runtime da aplicação.

Comandos de execução:

```
npx playwright test        # roda a suítenpx playwright test --ui   # interface gráfica
(recomendada pela doc oficial)
```

Extensão VS Code: instale a extensão oficial "Playwright Test for VSCode" — permite gravar interações (`Record new`) e gera o código do teste automaticamente a partir de cliques reais na aplicação.

3. Estrutura de pastas sugerida

```
tests/
├── base.ts                       # fixtures customizadas + decorator @step
├── e2e/
│   └── <dominio>/
│       └── <feature>/
│           ├── <Feature>TestPage.ts   # Page Object da tela│           └──
<feature>.spec.ts      # spec que usa a Page Object
├── setup/
│   ├── login.setup.ts             # roda antes de tudo, gera sessão reutilizável│   ├──
seed.setup.ts              # opcional — garante dados de apoio via API│   └──
cleanup.teardown.ts        # opcional — apaga dados de apoio via API
└── support/    ├── apiClient.ts               # cliente HTTP/RPC isolado, sem depender de
src/
```

```
    └── seeders/
        ├── types.ts                # contrato Seeder        ├── registry.ts             # lista
ordenada de seeders
        └── <dominio>/
            ├── <dominio>.config.ts
            └── <dominio>.seeder.ts
```

O que importa reter dessa organização: **um Page Object por tela testada**, testes de setup/seed/cleanup como *projects* isolados (não como `beforeAll`/`afterAll` dentro do spec), e o cliente de API de teste separado de qualquer código de `src/`.

4. Configuração (`playwright.config.ts`)

```
import { defineConfig, devices } from "@playwright/test";
import dotenv from "dotenv";
import path from "path";
import { fileURLToPath } from "url";
const __dirname = path.dirname(fileURLToPath(import.meta.url));dotenv.config({ path:
path.resolve(__dirname, ".env") });
export default defineConfig({
  testDir: "./tests",
  fullyParallel: true,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 4 : undefined,
  reporter: process.env.CI    ? [["junit", { outputFile: "results.xml" }], ["html", { open:
"never" }]]
    : "html",
  expect: { timeout: 10_000 },
  use: {
    baseURL: "http://localhost:5173", // ajuste para a porta real do `vite dev`    trace: "on-
first-retry"
```

```
  },
  projects: [
    {
      name: "login",
      use: { ...devices["Desktop Chrome"] },
      testMatch: /login\.setup\.ts/
    },
    {
      name: "chromium",
      use: { ...devices["Desktop Chrome"] },
      dependencies: ["login"],
      testMatch: /.*\.spec\.ts/
    }
    // Adicione "seed"/"cleanup" como projects extras só se o projeto    // realmente precisar
de dados de apoio via API (ver seção 9).
  ],
  webServer: {
    command: "npm run dev",
    url: "http://localhost:5173", // igual ao baseURL acima    reuseExistingServer:
!process.env.CI
  }
});
```

> **Atenção à porta**: é um erro comum apontar essa configuração para uma porta padrão (`localhost:3000`) quando o Vite do projeto usa outra. Confirme a porta real do `npm run dev` do seu projeto antes de fixar `baseURL`/`webServer.url` — é fácil de repetir.

Os `projects` `login`/`chromium` com `dependencies` são o mecanismo central: o Playwright sempre roda o project de login primeiro, e os specs (`chromium`) só começam depois que ele termina — sem precisar refazer login em cada teste.

5. Variáveis de ambiente

Crie um `.env.example` na raiz do projeto de testes (versionado) e um `.env` real (ignorado pelo git):

```
# Credenciais usadas pelo setup de login dos testes E2E (Playwright)E2E_USER="USUARIO_DE_TESTE"
E2E_PASSWORD="SENHA_DE_TESTE"
```

6. Setup de login (sessão reutilizável)

`tests/setup/login.setup.ts`:

```
import fs from "fs";
import path from "path";
import { fileURLToPath } from "url";
import { test as setup, expect } from "@playwright/test";
const user = process.env.E2E_USER;
const password = process.env.E2E_PASSWORD;
if (user === undefined) throw new Error("E2E_USER não definido");if (password === undefined)
throw new Error("E2E_PASSWORD não definido");
const __dirname = path.dirname(fileURLToPath(import.meta.url));const sessionFilePath =
path.resolve(__dirname, "../../playwright/.auth/sessionStorage.json");
setup("Setup Inicial de Login do Sistema", async ({ page }) => {
  await page.goto("/login");
  await page.getByLabel("Email").fill(user);
  await page.getByLabel("Senha").fill(password);  await page.getByRole("button", { name:
"Entrar" }).click();
  await expect(page.getByText("Nome do Sistema")).toBeVisible();
  // Se a app usa sessionStorage (não localStorage) para guardar o token,  // `storageState()`
nativo do Playwright não serve — precisa salvar na mão:  const sessionStorageData = await
page.evaluate(() => JSON.stringify(window.sessionStorage));
  fs.mkdirSync(path.dirname(sessionFilePath), { recursive: true });
fs.writeFileSync(sessionFilePath, sessionStorageData, "utf-8");
});
```

> Se a aplicação alvo usa `localStorage` em vez de `sessionStorage`, use o `storageState()` nativo do Playwright (`context.storageState({ path })`) — é mais simples e não exige o `addInitScript` manual do passo 7. O caminho manual só é necessário quando o token fica em `sessionStorage`.

7. Fixtures + Page Object Model

`tests/base.ts` — fixture customizada (`createTestPage`) que injeta a sessão salva e fornece um decorator `@step` para nomear passos no relatório do Playwright:

```
import fs from "fs";
import path from "path";
import { fileURLToPath } from "url";import { test as base, expect, Page } from
"@playwright/test";
const __dirname = path.dirname(fileURLToPath(import.meta.url));const sessionFilePath =
path.resolve(__dirname, "../playwright/.auth/sessionStorage.json");
type PageObjectClass<T> = new (page: Page) => T;
export const test = base.extend<{  createTestPage: <T>(PageObject: PageObjectClass<T>) => T;
}>({
  page: async ({ page }, use) => {
    if (fs.existsSync(sessionFilePath)) {      const sessionStorageData =
fs.readFileSync(sessionFilePath, "utf-8");
      await page.addInitScript((storage) => {
        const entries = JSON.parse(storage);        for (const [key, value] of
Object.entries(entries)) {
          window.sessionStorage.setItem(key, value as string);
        }
      }, sessionStorageData);
    }
    await page.goto("/");
    await use(page);
  },
  createTestPage: async ({ page }, use) => {    await use((PageObject) => new
PageObject(page));
```

```
  }
});
export { expect };
/** Decorator que envolve um método de Page Object num `test.step` nomeado. */export function
step(stepName?: string) {
  return function decorator<This, Args extends unknown[], Return>(    target: (this: This,
...args: Args) => Return,    context: ClassMethodDecoratorContext<This, (this: This, ...args:
Args) => Return>
  ) {
    function replacementMethod(this: This, ...args: Args): Return {      const name =
`${stepName || (context.name as string)}`;
      return test.step(name, async () => {
        return target.call(this, ...args);
      }) as Return;
    }
    return replacementMethod;
  };
}
```

Todo spec deve importar `test`/`expect` **deste arquivo**, nunca direto de `@playwright/test` — é o que garante que a sessão salva seja injetada.

Page Object — uma classe por tela, cada teste vira um método público:

```
export class ItemTestPage {
  constructor(private page: Page) {}
  private async fillForm(name: string) {
    // ... lógica de preenchimento
  }
  async setup() {
    // ... navegação até a tela, pré-condições
  }
  async testPageLoad() {
    // ... asserção de que a página carregou
  }
  async testCreate() {
```

```
    // ... teste de criação
  }
  async testEdit() {
    // ... teste de edição
  }
  async testDelete() {
    // ... teste de exclusão
  }
}
```

8. Exemplo de spec genérico

```
import { test } from "../../base";
import { ItemTestPage } from "./ItemTestPage";
test.describe("Testes para Cadastro de Item", () => {  test("Carregamento da página", async ({
createTestPage }) => {
    const itemPage = createTestPage(ItemTestPage);
    await itemPage.setup();
    await itemPage.testPageLoad();
  });
  /**
   * Testes em série — dependem do estado deixado pelo teste anterior.   * Fluxo: criar → editar
→ excluir.
   */
  test.describe.serial("Testes de Fluxo do Usuário", () => {    test("Cadastro de Item", async
({ createTestPage }) => {
      const itemPage = createTestPage(ItemTestPage);
      await itemPage.setup();
      await itemPage.testCreate();
    });
    test("Edição de Item", async ({ createTestPage }) => {      const itemPage =
createTestPage(ItemTestPage);
      await itemPage.setup();
      await itemPage.testEdit();
```

```
    });
    test("Exclusão de Item", async ({ createTestPage }) => {      const itemPage =
createTestPage(ItemTestPage);
      await itemPage.setup();
      await itemPage.testDelete();
    });
  });
});
```

`test.describe.serial` é o que garante a ordem e compartilha falha em cadeia — se "Cadastro" falhar, os demais são pulados em vez de rodar contra um estado inconsistente.

9. Seed/cleanup de dados via API (opcional, avançado)

Só vale a pena se os testes dependem de dados de apoio (ex.: uma entidade pai que precisa existir antes da tela ser testável) e você quer evitar que cada spec crie/apague seus próprios dados.

Padrão recomendado:

- **`tests/support/apiClient.ts`**: abre uma sessão direto contra o backend (sem passar pelo código de `src/`), autenticando com `E2E_USER`/`E2E_PASSWORD`. Só existe porque o carregador ESM do Node (usado pelo Playwright) não resolve certos imports profundos sem extensão que o bundler do Vite tolera — se o seu backend for uma API REST comum, um `fetch`/client HTTP simples resolve sem essa complicação.
- **`tests/support/seeders/types.ts`**: contrato comum `Seeder` (`name`, `seedAll()`, `teardownAll()`).
- **`tests/support/seeders/registry.ts`**: lista ordenada de seeders — a ordem importa quando um seeder depende de dado criado por outro.
- **`tests/setup/seed.setup.ts`** (project `seed`, roda antes dos specs): itera `SEEDERS` chamando `seedAll()`.
- **`tests/setup/cleanup.teardown.ts`** (project `cleanup`, com `teardown: "cleanup"` no project principal): itera `SEEDERS` **na ordem inversa** chamando `teardownAll()`, para respeitar dependências ao apagar.

No `playwright.config.ts`, isso vira:

```
```

```
projects: [  { name: "login", testMatch: /login\.setup\.ts/, use: { ...devices["Desktop Chrome"] } },
  { name: "seed", testMatch: /seed\.setup\.ts/ },
  {
    name: "chromium",
    use: { ...devices["Desktop Chrome"] },
    dependencies: ["login", "seed"],
    teardown: "cleanup",
    testMatch: /.*\.spec\.ts/
  },
  { name: "cleanup", testMatch: /cleanup\.teardown\.ts/ }
]
```

Se o projeto novo não tiver essa necessidade de dados de apoio compartilhados, **pule esta seção inteira** — é a parte mais específica/avançada deste setup, não um requisito do Playwright em si.

10. `.gitignore` e scripts do `package.json`

Adicionar ao `.gitignore`:

```
# Playwright
/test-results/
/playwright-report/
/blob-report/
/playwright/.cache/
/playwright/.auth/
```

É comum não haver um script dedicado (`"test": "playwright test"`) no `package.json`, rodando os testes via `npx playwright test` diretamente. Vale considerar adicionar esse script no projeto novo desde o início:

```
```

```
"scripts": {
  "test:e2e": "playwright test",
  "test:e2e:ui": "playwright test --ui"
}
```

11. Decisões e armadilhas comuns

- **Porta do `webServer` / `baseURL` precisa bater exatamente com a porta do `vite dev`** — é um erro fácil de cometer e só aparece depois de já commitado. Confirme antes de bater o martelo.
- **`sessionStorage` vs `localStorage`**: se a app-alvo guarda token/sessão em `sessionStorage`, o `storageState()` nativo do Playwright não funciona — é preciso capturar manualmente com `page.evaluate` no setup e reinjetar com `page.addInitScript` na fixture (seções 6 e 7). Se a app usa `localStorage`, prefira o mecanismo nativo, é mais simples.
- **Nunca importar `test` / `expect` direto de `@playwright/test` nos specs** — sempre do `tests/base.ts` local, senão a fixture de sessão não é aplicada.
- **Setup/seed/cleanup como *projects* separados**, não como hooks dentro dos specs — isso é o que permite rodar uma vez só (login) ou uma vez por suíte inteira (seed/cleanup), em vez de repetir por arquivo/teste.
- **Teste em série (`test.describe.serial`) só quando o fluxo realmente depende de estado anterior** (criar → editar → excluir da mesma entidade); testes independentes devem ficar fora do bloco serial para poderem paralelizar.

Revision #2
Created Thu, Aug 20, 2026 5:14 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:57 PM by Geraldo Barbosa