Guia Interno de Testes Automatizados (Playwright)

Este guia explica, em linguagem simples, como funciona o sistema de testes deste projeto, quais são os componentes, como ele foi criado, como se conecta ao GitHub e como pode se integrar ao Qase.io. O objetivo é qualquer pessoa, mesmo sem experiência técnica, conseguir entender o fluxo e colaborar no projeto.

---

1) O que este projeto faz (visão simples)

Imagine um robô que usa o navegador para clicar, digitar e publicar conteúdos, exatamente como uma pessoa faria. Esse robô é o Playwright.

Os testes são “roteiros” que dizem ao robô o que fazer. Quando você roda os testes, ele entra no site de QA, preenche as telas, publica posts e confirma se tudo ocorreu corretamente.

---

2) Componentes do sistema (pastas e arquivos principais)

- `tests/`
- Onde ficam os testes.
- Cada arquivo `*.spec.ts` é um conjunto de testes.
- `helpers/`
- Funções reutilizáveis para não repetir código.
- Exemplo: clicar em “Próximo”, preencher título, escolher editoria.
- `pages/`
- “Page Objects”, que são classes para interagir com telas específicas.
- `data/`
- Dados fixos (ex.: usuário) e mídias usadas nos testes.
- `storage/`
- Sessão salva de login (`auth.json`) para evitar MFA em todo teste.
- `scripts/`
- Scripts auxiliares (ex.: gerar `auth.json`).
- `.github/workflows/`
- Configurações do GitHub Actions para rodar testes automaticamente no GitHub.

---

3) Como o login funciona (sem precisar MFA toda hora)

O login usa Microsoft SSO, que normalmente pede MFA. Para evitar isso, o projeto usa um arquivo de sessão salvo:

- Script que gera a sessão:

```
npm run auth:setup
```

O script abre o navegador, você faz login manualmente e ele salva a sessão em `storage/auth.json`.

Depois disso, os testes usam essa sessão automaticamente.

---

4) Fluxo dos testes de criação de post

Os testes seguem sempre o mesmo fluxo básico:

1. Entrar na tela de criação.
2. Selecionar canais (Aplicativo e TV quando aplicável).
3. Selecionar editoria (sempre “Einstein - Institucional - Padrão”).
4. Preencher Título e Texto.
5. (Se necessário) enviar mídia.
6. Configurar opções de priorização.
7. Publicar.
8. Confirmar que apareceu um novo card na tela “Comunicações”.

Para posts com vídeo: - Depois de clicar em “Publicar”, o teste aguarda o fim do upload (barra de progresso) antes de validar o card.

---

5) Como os títulos são gerados

Para facilitar a identificação no ambiente de QA, os títulos ficam assim:

```
Teste automatizado_Froes – <tipo de post> – <data e hora>
```

Exemplo:

```
Teste automatizado_Froes – Post simples – com imagem – 2026-02-
```

```
12 11:58:21.824 UTC
```

---

6) Arquivos e funções importantes (explicação simples)

`helpers/ui.ts` - Funções genéricas para clicar ou preencher campos sem quebrar. - Exemplo: `tryClick` tenta clicar em vários seletores até achar um visível.

`helpers/auth.ts` - Verifica se a sessão está válida. - Se o `auth.json` estiver inválido, avisa para gerar novamente.

`helpers/communications.ts` - Contém funções específicas para criar comunicações. - Exemplos: - `selectEditoria`: escolhe a editoria correta e espera o botão “Próximo” habilitar. - `uploadMediaFile`: envia imagem, gif ou vídeo. - `dismissCropDialogIfPresent`: fecha o modal de crop.

`tests/communications/creation.spec.ts` - Arquivo principal de testes de criação. - Contém o fluxo completo de publicação simples e os fluxos iniciais de outros tipos.

---

7) Como rodar os testes localmente

- Rodar todos:

```
npm test
```

- Rodar com navegador visível:

```
node scripts/playwright.js test --headed
```

- Rodar só um teste específico:

```
node scripts/playwright.js test
tests/communications/creation.spec.ts -g "Post simples - com
imagem"
```

---

8) Como se conecta ao GitHub

No GitHub, o projeto é armazenado em um repositório. Para enviar alterações:

```
git add .
git commit –m "Mensagem"
git push origin main
```

Se estiver usando HTTPS, você precisa de um **Personal Access Token (PAT)** como senha.

Para salvar credenciais no macOS:

```
git config ––global credential.helper osxkeychain
```

---

9) Como o GitHub Actions roda os testes

Os workflows ficam em `.github/workflows/`.

Fluxo básico: - Baixa o código. - Instala Node. - Instala dependências. - Instala Playwright + browsers. - Executa os testes.

O relatório é salvo como artefato no GitHub Actions.

---

10) Como integrar com Qase.io (visão simples)

Qase é uma plataforma de gerenciamento de testes. A integração permite enviar resultados automaticamente.

Passo a passo (modo recomendado)

1. Instalar o reporter do Qase:

```
npm i –D playwright–qase–reporter
```

1. Criar um arquivo `qase.config.json` na raiz do projeto:

```
{
  "mode": "testops",
  "testops": {
    "api": {
      "token": "<SEU_TOKEN_QASE>"
    },
    "project": "<CODIGO_DO_PROJETO>"
  }
}
```

1. Adicionar o reporter no `playwright.config.ts`:

```
reporter: [
  ['html', { open: 'never' }],
  ['github'],
  ['playwright-qase-reporter']
],
```

1. Rodar testes normalmente. O Qase receberá os resultados.

Boas práticas - Nunca commitar o token. Use variáveis de ambiente ou GitHub Secrets.

---

11) Como integrar Qase no GitHub Actions

1. No GitHub, crie os secrets:
2. `QASE_API_TOKEN`
3. `QASE_PROJECT`
4. No workflow, adicione antes do passo de testes:

```
- name: Set Qase env
  run: |
    echo "QASE_API_TOKEN=${{ secrets.QASE_API_TOKEN }}" >>
$GITHUB_ENV
```

```
    echo "QASE_PROJECT=${{ secrets.QASE_PROJECT }}" >>
$GITHUB_ENV
```

1. Ajuste o `qase.config.json` para usar variáveis de ambiente.

---

12) Resumo para alguém totalmente leigo

- O Playwright é o robô.
- Os arquivos em `tests/` são os roteiros.
- O `auth.json` é a chave de login.
- O GitHub guarda o projeto e roda testes automaticamente.
- O Qase guarda os resultados organizados.

Se precisar, posso transformar isso em um manual mais curto para treinamento rápido ou preparar um onboarding para novos membros.