Plano de Testes Completo para https://www.google.com.br/

Este plano de testes visa garantir a qualidade, performance, segurança e acessibilidade do site https://www.google.com.br/. O plano abrange diferentes tipos de testes, incluindo funcional, de performance, de segurança, de acessibilidade, de regressão e de API, com foco em automatizar o máximo possível do processo de teste.

1. Plano de Teste Funcional (PDF)

1.1 Casos de Teste:

* **Navegação:**
    * **Cenário Positivo:** Acessar a página inicial, navegar para diferentes seções (Imagens, Gmail, Drive, etc.), acessar um resultado de busca específico.
    * **Cenário Negativo:** Tentar acessar seções inexistentes, inserir URL incorreta, realizar busca por termo inválido.
* **Busca:**
    * **Cenário Positivo:** Realizar busca por termos simples, termos complexos, frases, imagens, vídeos.
    * **Cenário Negativo:** Realizar busca por termos inválidos, inserir termos com caracteres especiais, pesquisar por conteúdo bloqueado.
* **Login/Logout:**
    * **Cenário Positivo:** Logar com credenciais válidas, realizar logout, logar com conta Google, logar com conta de terceiros.
    * **Cenário Negativo:** Tentar logar com credenciais inválidas, esquecer a senha, tentar logar com conta bloqueada.
* **Submissão de Formulários:**
    * **Cenário Positivo:** Preencher e enviar formulário de pesquisa, formulário de contato, formulário de inscrição.
    * **Cenário Negativo:** Submeter formulário com campos inválidos, submeter formulário sem preencher campos obrigatórios, enviar formulário com dados duplicados.
* **Interação com elementos dinâmicos:**
    * **Cenário Positivo:** Interagir com autocomplete, menus drop-down, modais, carrosséis, filtros de busca.
    * **Cenário Negativo:** Tentar interagir com elementos não funcionais, manipular elementos com comportamento inesperado.

1.2 Critérios de Aceitação:

* Todas as funcionalidades descritas no escopo do teste devem funcionar corretamente.
* O tempo de carregamento das páginas deve ser inferior a 3 segundos.
* Os erros devem ser tratados de forma amigável ao usuário.
* A interface do usuário deve ser clara, intuitiva e fácil de usar.
* O site deve ser compatível com diferentes navegadores e dispositivos móveis.

1.3 Escopo do Teste:

* Páginas principais do site (Página inicial, busca, login, etc.).
* Seções principais do site (Imagens, Gmail, Drive, etc.).
* Fluxos de usuário comuns (pesquisa, login, envio de formulários).

1.4 Ambientes de Teste:

* Navegadores: Chrome, Firefox, Safari, Edge.
* Dispositivos móveis: iOS, Android.
* Resoluções de tela: 1024x768, 1280x800, 1920x1080.

1.5 Riscos e Dependências:

* Mudanças frequentes no site podem afetar a execução dos testes.
* Dependência de APIs externas pode impactar a disponibilidade do site.
* Mudanças nos algoritmos de busca podem influenciar os resultados dos testes.

1.6 Métricas de Sucesso:

* Taxa de defeitos: Percentual de casos de teste que falharam.
* Cobertura de teste: Percentual de funcionalidades do site cobertas pelos testes.
* Tempo de execução dos testes: Tempo total gasto na execução dos testes.

2. Projeto de Automação em Cypress

2.1 Estrutura do Projeto:

```
codigo_automacao/
% % %  cypress/
%    % % %  integration/
%    %    % % %  navegação.spec.js
%    %    % % %  busca.spec.js
%    %    % % %  login.spec.js
%    %    % % %  formulario.spec.js
%    % % %  support/
%    %    % % %  commands.js
%    % % %  plugins/
%    %    % % %  index.js
%    % % %  fixtures/
%       % % %  usuarios.json
% % %  e2e.config.js
% % %  cypress.config.js
% % %  allure-results/
   % % %  [data]
```

```
    % % %  [reports]
```

2.2 Código Cypress:

Navegação:

```javascript
describe('Navegação', () => {
  it('Acessar a página inicial', () => {
    cy.visit('/');
    cy.title().should('eq', 'Google');
  });

  it('Acessar a seção de imagens', () => {
    cy.get('a[href="/imghp?hl=pt-BR"]').click();
    cy.url().should('include', '/imghp');
  });
});
```

Busca:

```javascript
describe('Busca', () => {
  it('Realizar busca por termo simples', () => {
    cy.visit('/');
    cy.get('#APjFqb').type('Cypress{enter}');
    cy.get('.g').should('be.visible');
  });
});
```

2.3 Cobertura de Teste:

* Casos de borda: Testes para validação de limites de entrada, cenários com dados inválidos, comportamentos inesperados.
* Casos de erro: Testes para validação de mensagens de erro, tratamento de exceções, recuperação de erros.
* Cenários com dados dinâmicos: Testes com dados variáveis, simulação de inputs do usuário, integração com APIs.

2.4 Integração com Allure:

```javascript
// cypress.config.js
```

```
module.exports = defineConfig({
  e2e: {
    setupNodeEvents(on, config) {
      // ...
      require('@shepherd.js/cypress-allure-plugin')(on);
    },
  },
});
```

2.5 Configuração de Variáveis de Ambiente:

```javascript
// cypress.config.js
module.exports = defineConfig({
  e2e: {
    baseUrl: Cypress.env('BASE_URL'),
    // ...
  },
});
```

3. Instruções de Execução (README.md)

3.1 Configuração do Ambiente:

* Instalar Node.js: https://nodejs.org/
* Instalar Cypress: `npm install cypress --save-dev`
* Configurar variáveis de ambiente:
    * `BASE_URL=https://www.google.com.br/`
    * `ALLURE_REPORT_DIR=allure-results`
* Executar o comando `npx cypress open` para abrir o Cypress Dashboard.

3.2 Execução dos Testes:

* Executar todos os testes: `npx cypress run`
* Executar testes específicos: `npx cypress run --spec "cypress/integration/busca.spec.js"`
* Executar testes com ambiente de teste específico: `npx cypress run --env environment=staging`

3.3 Interpretação dos Resultados:

* Verificar os resultados no terminal.
* Verificar os relatórios no Cypress Dashboard.
* Visualizar os relatórios Allure em `allure-results/reports/index.html`.

3.4 Configuração de CI/CD:

* Configurar GitHub Actions ou Jenkins para executar os testes automaticamente em cada build.
* Incluir o comando `npx cypress run` no script de CI/CD.

4. Plano de Testes de Performance (PDF)

4.1 Testes de Carga:

* Utilizar ferramentas como JMeter ou k6 para simular tráfego de usuários simultâneos.
* Definir cenários de carga com diferentes níveis de usuários (por exemplo, 100, 500, 1000).
* Monitorar métricas como tempo de resposta, throughput, taxa de erros.

4.2 Testes de Estresse:

* Simular picos de carga para identificar o ponto de falha do site.
* Aumentar gradualmente o número de usuários simultâneos até atingir o limite da capacidade do site.
* Monitorar as mesmas métricas dos testes de carga.

4.3 Testes de Capacidade:

* Definir a capacidade máxima do site sem degradação significativa da performance.
* Executar testes de carga com diferentes níveis de usuários para determinar a capacidade máxima.
* Ajustar a capacidade do site com base nos resultados dos testes.

4.4 Métricas de Performance:

* Tempo de resposta: Tempo médio que o site leva para responder às solicitações.
* Throughput: Número de solicitações atendidas por segundo.
* Taxa de erros: Percentual de solicitações que geram erros.

5. Plano de Testes de Segurança (PDF)

5.1 Testes de Vulnerabilidade:

* Identificar vulnerabilidades comuns como injeção de SQL, XSS, CSRF.
* Utilizar ferramentas como Burp Suite ou ZAP para automatizar a busca por vulnerabilidades.
* Testar as funcionalidades mais importantes do site para garantir sua segurança.

5.2 Teste de Autenticação/Autorização:

* Verificar se as políticas de autenticação e autorização estão implementadas corretamente.
* Testar diferentes cenários de login, logout, acesso a áreas restritas.
* Garantir que apenas usuários autorizados tenham acesso a recursos específicos.

5.3 Teste de Penetração:

* Realizar testes de penetração para avaliar a resistência do site a ataques maliciosos.
* Simular ataques reais para identificar falhas de segurança.
* Contratar especialistas em segurança para realizar testes de penetração mais avançados.

6. Plano de Testes de Acessibilidade (PDF)

6.1 Conformidade com WCAG:

* Testar o site contra as diretrizes WCAG (Web Content Accessibility Guidelines).
* Utilizar ferramentas como Axe, Lighthouse ou NVDA para automatizar os testes.
* Garantir que o site seja acessível a todos os usuários, incluindo pessoas com deficiências.

6.2 Ferramentas de Acessibilidade:

* Axe: Ferramenta de teste de acessibilidade para navegadores web.
* Lighthouse: Ferramenta de análise de performance, acessibilidade e SEO.
* NVDA: Leitor de tela para pessoas com deficiência visual.

6.3 Casos de Teste Acessíveis:

* Navegação com teclado: Testar a navegação pelo site utilizando apenas o teclado.
* Uso de leitores de tela: Testar a usabilidade do site com leitores de tela.
* Acessibilidade para pessoas com deficiências visuais e motoras: Testar a acessibilidade do site para usuários com diferentes tipos de deficiências.

7. Estratégia de Teste de Regressão

7.1 Seleção de Casos de Teste:

* Identificar os casos de teste mais importantes para garantir a qualidade do site.
* Priorizar casos de teste com maior risco de falhas.
* Incluir testes para todas as funcionalidades principais do site.

7.2 Automação de Regressão:

* Criar scripts de teste automatizados para executar os testes de regressão.

* Utilizar ferramentas como Cypress ou Selenium para automatizar os testes.
* Executar os testes de regressão a cada build do site.

7.3 Planejamento de Execução:

* Executar os testes de regressão diariamente ou a cada pull request.
* Integrar os testes de regressão em um pipeline de CI/CD.
* Monitorar os resultados dos testes de regressão para identificar falhas e corrigir problemas.

8. Documentação de Integração de APIs

8.1 Testes de API Automatizados:

* Criar casos de teste para validar as respostas das APIs.
* Utilizar ferramentas como Postman ou Newman para automatizar os testes.
* Simular solicitações e validar as respostas das APIs.

8.2 Testes de Contrato:

* Definir como garantir que as APIs respeitam os contratos esperados.
* Utilizar ferramentas como Pact para verificar a compatibilidade entre as APIs.
* Garantir a consistência das APIs em diferentes ambientes de desenvolvimento.

8.3 Testes de Carga em APIs:

* Detalhe como realizar testes de carga especificamente para as APIs.
* Utilizar ferramentas como k6 para simular tráfego de solicitações às APIs.
* Monitorar as mesmas métricas de performance dos testes de carga para o site.

Organização dos arquivos e pastas:

* A estrutura de arquivos e pastas deve ser clara e organizada.
* O código de automação Cypress deve estar em uma pasta chamada 'codigo_automacao'.
* Os casos de teste devem ser organizados em pastas por funcionalidade.
* Os relatórios de teste devem ser armazenados em um local acessível.

Manutenção dos testes:

* Manter os casos de teste atualizados com as mudanças no site.
* Revisar e atualizar os testes de forma regular.
* Documentar as mudanças e atualizações nos testes.

Conclusão:

Este plano de teste completo fornece uma base sólida para garantir a qualidade, performance, segurança e acessibilidade do site https://www.google.com.br/. A implementação dos testes descritos neste plano, com foco em automação, permitirá que o site seja lançado e mantido com alta qualidade.