Plano de Teste Completo para Google.com.br

Este plano de teste abrangente para o site Google.com.br visa garantir a qualidade, performance, segurança e acessibilidade do site. Ele inclui diferentes tipos de testes, além de instruções para automatizar a execução e gerar relatórios detalhados.

1. Plano de Teste Funcional (PDF)

1.1. Casos de Teste:

* **Funcionalidades Básicas:**
    * Busca:
        * Pesquisar por termos simples e complexos (com operadores booleanos).
        * Validar a exibição dos resultados de pesquisa.
        * Testar diferentes filtros de pesquisa (data, idioma, tipo de arquivo).
    * Navegação:
        * Navegar pelas diferentes seções do site (Imagens, Vídeos, Notícias).
        * Validar a funcionalidade dos links e botões.
        * Testar a navegação entre diferentes páginas.
    * Login/Logout:
        * Efetuar login e logout utilizando diferentes tipos de contas (Google, Facebook).
        * Validar a exibição da página de login e logout.
        * Testar a funcionalidade de recuperação de senha.
    * Submissão de Formulários:
        * Submeter formulários de contato, feedback e pesquisas.
        * Validar a funcionalidade de validação de campos.
        * Testar a exibição de mensagens de erro e sucesso.
    * Interações com Elementos Dinâmicos:
        * Interagir com menus dropdown, modais e carrosséis.
        * Validar a funcionalidade de scroll infinito e auto-complete.
    * Acessibilidade:
        * Validar a acessibilidade de cada página do site, incluindo a navegação com teclado, uso de leitores de tela e compatibilidade com diferentes tecnologias assistivas.
    * Outros:
        * Teste de funcionalidades específicas como Google Maps, Google Translate, Google Drive.
* **Cenários Positivos e Negativos:**
    * Validar a funcionalidade do site em cenários esperados e não esperados.
    * Testar o comportamento do site em situações de erro e falhas.
* **Passos de Execução:**
    * Descrição detalhada dos passos para executar cada caso de teste.
* **Dados de Teste:**
    * Utilização de dados reais e fictícios para testes de diferentes cenários.
* **Critérios de Aceitação:**
    * Especificação clara dos critérios para considerar um teste como aprovado.

1.2. Escopo do Teste:

* Todas as funcionalidades principais do site, incluindo a página inicial, resultados de pesquisa, página de login, páginas de perfil de usuário, Google Maps, Google Drive, e outras funcionalidades específicas.
* Todas as páginas e fluxos de usuário relevantes.

1.3. Ambientes de Teste:

* Navegadores: Chrome, Firefox, Safari, Edge.
* Dispositivos móveis: Smartphones e tablets com diferentes sistemas operacionais (Android, iOS).
* Diferentes resoluções de tela: Alta resolução (Full HD, 4K), baixa resolução (HD, 720p).
* Sistemas Operacionais: Windows, macOS, Linux.

1.4. Riscos e Dependências:

* Dependência de APIs externas.
* Impacto de atualizações do site em funcionalidades existentes.
* Riscos de segurança e privacidade.

1.5. Métricas de Sucesso:

* Taxa de defeitos: Porcentagem de testes falhos.
* Cobertura de teste: Porcentagem de funcionalidades e casos de teste cobertos.
* Tempo de resposta: Tempo médio de carregamento das páginas.
* Taxa de sucesso de login: Porcentagem de logins bem-sucedidos.
* Taxa de conclusão de formulários: Porcentagem de formulários submetidos com sucesso.

2. Projeto de Automação em Cypress

2.1. Estrutura do Projeto:

* **codigo_automacao:**
    * **tests:**
        * **busca:**
            * busca_simples.cy.js
            * busca_complexa.cy.js
            * filtro_data.cy.js
            * filtro_idioma.cy.js
            * filtro_tipo_arquivo.cy.js
        * **navegacao:**
            * navegacao_principal.cy.js
            * navegacao_paginas_internas.cy.js

        * **login_logout:**
            * login_google.cy.js
            * login_facebook.cy.js
            * logout.cy.js
        * **formularios:**
            * formulario_contato.cy.js
            * formulario_feedback.cy.js
        * **elementos_dinamicos:**
            * menus_dropdown.cy.js
            * modais.cy.js
            * carrosseis.cy.js
        * **acessibilidade:**
            * navegacao_teclado.cy.js
            * leitores_tela.cy.js
        * **api:**
            * api_busca.cy.js
            * api_login.cy.js
            * api_cadastro.cy.js
    * **fixtures:**
        * usuarios.json
        * buscas.json
        * dados_formularios.json
    * **commands:**
        * custom_commands.js
    * **plugins:**
        * index.js
    * cypress.config.js

2.2. Código Cypress:

* **Scripts para funcionalidades principais:**
    * **Navegação:**
        * Testar a navegação entre as diferentes seções do site.
        * Validar a funcionalidade dos links e botões.
        * Verificar se as páginas carregam corretamente.
    * **Busca:**
        * Testar diferentes tipos de buscas (simples e complexas).
        * Validar a exibição dos resultados de pesquisa.
        * Testar os filtros de pesquisa.
    * **Login/Logout:**
        * Testar o processo de login e logout.
        * Validar a exibição das mensagens de erro e sucesso.
    * **Submissão de Formulários:**
        * Testar a submissão de diferentes tipos de formulários.
        * Validar a exibição das mensagens de erro e sucesso.
    * **Interações com Elementos Dinâmicos:**

    * Testar menus dropdown, modais, carrosséis, scroll infinito e auto-complete.
  * **Acessibilidade:**
    * Testar a navegação com teclado.
    * Testar a compatibilidade com leitores de tela.

2.3. Cobertura de Teste:

* Casos de borda, casos de erro e cenários com dados dinâmicos.
* Utilização de testes de regressão para garantir a qualidade após alterações no site.

2.4. Integração com Allure:

* Configurar o Cypress para gerar relatórios de teste no formato Allure.
* Criar relatórios detalhados com informações sobre os testes, incluindo capturas de tela, logs e dados de teste.

2.5. Configurações de Variáveis de Ambiente:

* Configurar variáveis de ambiente no Cypress para suportar múltiplos ambientes (desenvolvimento, homologação, produção).

3. Instruções de Execução (README.md)

3.1. Configuração do Ambiente:

* Instalar Node.js e npm.
* Instalar o Cypress globalmente: `npm install cypress -g`.
* Instalar as dependências do projeto: `npm install`.

3.2. Execução dos Testes:

* Rodar os testes de forma local: `npx cypress run`.
* Rodar os testes para um ambiente específico: `npx cypress run --env environment=staging`.
* Executar testes específicos: `npx cypress run --spec cypress/tests/busca/busca_simples.cy.js`.

3.3. Interpretação dos Resultados:

* Verificar os resultados dos testes no terminal.
* Acessar o dashboard do Cypress para obter informações adicionais sobre os testes.
* Visualizar os relatórios gerados pelo Allure.

3.4. Configuração de CI/CD:

* Integrar os testes Cypress em uma pipeline CI/CD.

* Executar os testes automaticamente a cada build.
* Gerar relatórios de teste em ferramentas de CI/CD como GitHub Actions ou Jenkins.

4. Plano de Testes de Performance (PDF)

4.1. Testes de Carga:

* Utilizar ferramentas como JMeter ou k6 para simular uma alta carga de usuários.
* Monitorar o tempo de resposta, throughput, taxa de erros e consumo de recursos.
* Identificar gargalos de performance e áreas de otimização.

4.2. Testes de Estresse:

* Executar testes de carga progressiva para identificar o ponto de falha do site.
* Avaliar o comportamento do site em condições extremas de carga.

4.3. Testes de Capacidade:

* Medir a capacidade máxima do site sem degradação significativa da performance.
* Determinar a capacidade de suporte a um determinado número de usuários simultâneos.

4.4. Métricas de Performance:

* Tempo de resposta: Tempo médio de carregamento das páginas.
* Throughput: Número de requisições processadas por segundo.
* Taxa de erros: Porcentagem de requisições que falharam.
* Consumo de recursos: Memória, CPU e disco.

5. Plano de Testes de Segurança (PDF)

5.1. Testes de Vulnerabilidade:

* Identificar vulnerabilidades comuns como injeção de SQL, XSS, CSRF e outros ataques.
* Utilizar ferramentas de escaneamento de vulnerabilidades como Burp Suite, ZAP ou OWASP.

5.2. Teste de Autenticação/Autorização:

* Verificar se as políticas de autenticação e autorização estão implementadas corretamente.
* Testar a segurança dos mecanismos de login e logout.
* Validar o controle de acesso a recursos específicos.

5.3. Teste de Penetração:

* Simular ataques maliciosos para avaliar a resistência do site.
* Encontrar vulnerabilidades que podem ser exploradas por hackers.
* Avaliar a efetividade das medidas de segurança implementadas.

6. Plano de Testes de Acessibilidade (PDF)

6.1. Conformidade com WCAG:

* Testar o site contra as diretrizes WCAG (Web Content Accessibility Guidelines).
* Garantir a acessibilidade para todos os usuários, incluindo pessoas com deficiência.

6.2. Ferramentas de Acessibilidade:

* Utilizar ferramentas como Axe, Lighthouse e NVDA para realizar testes de acessibilidade.
* Identificar e corrigir problemas de acessibilidade.

6.3. Casos de Teste Acessíveis:

* Testar a navegação com teclado.
* Validar a compatibilidade com leitores de tela.
* Verificar a acessibilidade para pessoas com deficiências visuais e motoras.

7. Estratégia de Teste de Regressão

7.1. Seleção de Casos de Teste:

* Identificar casos de teste críticos que devem ser incluídos no conjunto de regressão.
* Priorizar testes que cobrem as funcionalidades principais e mais utilizadas do site.

7.2. Automação de Regressão:

* Criar scripts de regressão automatizados que serão executados a cada build.
* Utilizar frameworks de teste como Cypress para automatizar os testes de regressão.

7.3. Planejamento de Execução:

* Definir a frequência de execução dos testes de regressão.
* Executar os testes diariamente, em cada pull request ou em intervalos regulares.

8. Documentação de Integração de APIs

8.1. Testes de API Automatizados:

* Criar casos de teste para validar as respostas de APIs.

* Utilizar ferramentas como Postman ou Newman para automatizar os testes de API.

8.2. Testes de Contrato:

* Garantir que as APIs respeitam os contratos esperados.
* Utilizar ferramentas como Pact para implementar testes de contrato.

8.3. Testes de Carga em APIs:

* Realizar testes de carga especificamente para as APIs.
* Utilizar ferramentas como k6 para simular uma alta carga de requisições às APIs.

8.4. Documentação de APIs:

* Criar documentação detalhada sobre as APIs, incluindo os endpoints, os parâmetros, os tipos de resposta e os exemplos de uso.

Organização dos Arquivos e Pastas:

* Todos os arquivos e pastas serão organizados de forma clara e acessível.
* O plano de teste funcional, o projeto Cypress e os planos de testes de performance, segurança e acessibilidade serão salvos em pastas separadas.
* Os relatórios de teste serão armazenados em um local centralizado, facilmente acessível para revisão posterior.

Considerações Adicionais:

* Este plano de teste é um guia inicial que pode ser adaptado e ajustado de acordo com as necessidades específicas do projeto.
* A equipe de teste deve trabalhar em colaboração com a equipe de desenvolvimento para garantir a qualidade do site.
* É importante realizar uma análise de risco para identificar os riscos e dependências que podem impactar os testes.
* As métricas de sucesso devem ser definidas e monitoradas para avaliar o progresso dos testes.

Este plano completo, combinado com as ferramentas e estratégias descritas, permitirá que a equipe de qualidade garanta a qualidade, performance, segurança e acessibilidade do site Google.com.br.