Plano de Teste Completo para o Site de Recursos Humanos - https://opensource-demo.orangehrmlive.com/

Este documento detalha o plano de teste abrangente para o site de Recursos Humanos, incluindo testes funcionais, de automação, performance, segurança, acessibilidade e regressão.

1. Plano de Teste Funcional (PDF)

1.1. Casos de Teste:

* **Login e Logout:**
    * Cenários Positivos: Login com credenciais válidas, logout bem-sucedido.
    * Cenários Negativos: Login com credenciais inválidas, campos obrigatórios em branco, tentativas de login múltiplas, logout com sessão expirada.
* **Gerenciamento de Funcionários:**
    * Cenários Positivos: Adicionar novo funcionário, editar dados de um funcionário existente, remover funcionário.
    * Cenários Negativos: Adicionar funcionário com dados inválidos, editar dados com informações incorretas, remover funcionário com dependências.
* **Folha de Pagamento:**
    * Cenários Positivos: Gerar folha de pagamento, visualizar detalhes da folha de pagamento, exportar relatório de folha de pagamento.
    * Cenários Negativos: Gerar folha de pagamento com dados inválidos, visualizar folha de pagamento com filtro inválido, exportar relatório com formato inválido.
* **Recrutamento:**
    * Cenários Positivos: Criar nova vaga, visualizar candidaturas, aprovar candidato, rejeitar candidato.
    * Cenários Negativos: Criar vaga com dados inválidos, visualizar candidaturas com filtros inválidos, aprovar candidato com dados faltantes, rejeitar candidato sem justificativa.
* **Treinamento:**
    * Cenários Positivos: Criar novo treinamento, matricular funcionários em treinamento, visualizar histórico de treinamentos.
    * Cenários Negativos: Criar treinamento com dados inválidos, matricular funcionários em treinamento com vagas indisponíveis, visualizar histórico com filtros inválidos.
* **Relatórios:**
    * Cenários Positivos: Gerar relatórios padrão, gerar relatórios personalizados, exportar relatórios em diferentes formatos.
    * Cenários Negativos: Gerar relatórios com filtros inválidos, exportar relatórios com formatos inválidos.

1.2. Critérios de Aceitação:

* Todas as funcionalidades devem funcionar conforme documentação.
* Todos os campos obrigatórios devem ser validados.

* O site deve ser responsivo e amigável para usuários de diferentes dispositivos e navegadores.
* Todos os erros devem ser tratados de forma adequada e com mensagens claras para o usuário.

1.3. Escopo do Teste:

* Todas as páginas do site, incluindo login, gerenciamento de funcionários, folha de pagamento, recrutamento, treinamento e relatórios.
* Todos os fluxos de usuário principais, incluindo cadastro, login, gerenciamento de dados, navegação e geração de relatórios.

1.4. Ambientes de Teste:

* **Navegadores:** Chrome, Firefox, Safari, Edge.
* **Dispositivos Móveis:** Android, iOS.
* **Resoluções de Tela:** Diferentes resoluções para desktop, tablet e mobile.

1.5. Riscos e Dependências:

* Dependências em APIs externas podem impactar os testes.
* Mudanças no design do site podem afetar scripts de automação.
* Acesso limitado a dados sensíveis pode limitar a abrangência dos testes.

1.6. Métricas de Sucesso:

* Taxa de defeitos (número de defeitos encontrados em relação ao número total de casos de teste).
* Cobertura de teste (porcentagem de código coberta por testes).
* Tempo médio para corrigir defeitos.

2. Projeto de Automação em Cypress (código_automacao):

2.1. Estrutura do Projeto:

* `codigo_automacao/`
    * `cypress/`
        * `integration/`
            * `login.spec.js`
            * `funcionarios.spec.js`
            * `folha_pagamento.spec.js`
            * `recrutamento.spec.js`
            * `treinamento.spec.js`
            * `relatorios.spec.js`
        * `fixtures/`
            * `usuarios.json`

        * `vaga.json`
        * `treinamento.json`
    * `support/`
        * `commands.js`
        * `index.js`
    * `plugins/`
        * `index.js`

2.2. Código Cypress:

* Scripts Cypress para validar funcionalidades principais:
    * Login/Logout:
        * Login com credenciais válidas e inválidas.
        * Verificação de mensagens de erro.
    * Navegação:
        * Verificação de links e menus.
    * Busca:
        * Busca por funcionários, vagas e treinamentos.
    * Submissão de Formulários:
        * Cadastro de funcionários, vagas e treinamentos.
        * Validação de campos obrigatórios e dados inválidos.
    * Interações com Elementos Dinâmicos:
        * Interação com elementos como tabelas, modais e dropdowns.

2.3. Cobertura de Teste:

* Casos de borda, casos de erro e cenários com dados dinâmicos.
* Utilização de fixtures para dados de teste.

2.4. Integração com Allure:

* Configuração do Allure para gerar relatórios de teste detalhados.
* Gerar screenshots e logs para cada passo do teste.

2.5. Configuração de Variáveis de Ambiente:

* Configuração de variáveis de ambiente para diferentes ambientes (desenvolvimento, homologação, produção).
* Armazenamento de credenciais e URLs em variáveis de ambiente.

3. Instruções de Execução (README.md):

3.1. Configuração do Ambiente:

1. Instalar Node.js e npm.
2. Instalar Cypress globalmente: `npm install cypress -g`.

3. Iniciar o projeto Cypress: `npx cypress open`.

3.2. Execução dos Testes:

* Executar todos os testes: `npx cypress run`.
* Executar testes específicos: `npx cypress run --spec "cypress/integration/login.spec.js"`.
* Executar testes em um ambiente específico: `npx cypress run --env environment=staging`.

3.3. Interpretação dos Resultados:

* Visualizar resultados no terminal e no dashboard do Cypress.
* Acessar relatórios gerados pelo Allure.

3.4. Configuração de CI/CD:

* Integrar os testes Cypress em uma pipeline CI/CD, como GitHub Actions ou Jenkins.
* Executar testes em cada build ou pull request.

4. Plano de Testes de Performance (PDF)

4.1. Testes de Carga:

* Utilizar ferramentas como JMeter ou k6 para simular carga pesada no site.
* Testar diferentes cenários de carga, como login simultâneo de vários usuários.
* Monitorar tempo de resposta, throughput e taxa de erros.

4.2. Testes de Estresse:

* Submeter o site a condições extremas para identificar o ponto de falha.
* Aumentar gradualmente a carga até o site atingir um estado instável.
* Monitorar o desempenho do site e identificar gargalos.

4.3. Testes de Capacidade:

* Determinar a capacidade máxima do site sem degradação significativa da performance.
* Realizar testes com diferentes quantidades de usuários e dados.
* Analisar o impacto na performance e identificar recursos limitantes.

4.4. Métricas de Performance:

* Tempo de resposta: Tempo que o site leva para responder a uma requisição.
* Throughput: Número de requisições que o site pode processar por segundo.
* Taxa de erros: Percentagem de requisições que falham.

5. Plano de Testes de Segurança (PDF)

5.1. Testes de Vulnerabilidade:

* Identificar vulnerabilidades comuns, como injeção de SQL, XSS e CSRF.
* Utilizar ferramentas de teste de vulnerabilidade, como Burp Suite e OWASP ZAP.
* Verificar a implementação de medidas de segurança, como validação de entrada e saída.

5.2. Teste de Autenticação/Autorização:

* Verificar se as políticas de autenticação e autorização estão corretamente implementadas.
* Testar diferentes cenários de acesso, como login com credenciais válidas e inválidas.
* Validar a autorização de acesso a recursos específicos, como páginas restritas.

5.3. Teste de Penetração:

* Simular ataques maliciosos para avaliar a resistência do site.
* Utilizar técnicas de penetração para tentar explorar vulnerabilidades.
* Avaliar a capacidade de resposta do site a ataques e a eficácia das medidas de segurança.

6. Plano de Testes de Acessibilidade (PDF)

6.1. Conformidade com WCAG:

* Testar o site contra as diretrizes WCAG para garantir a acessibilidade para todos os usuários.
* Utilizar ferramentas de acessibilidade, como Axe, Lighthouse e NVDA.

6.2. Ferramentas de Acessibilidade:

* Recomendar ferramentas para realizar testes de acessibilidade.
* Integrar ferramentas de acessibilidade no pipeline CI/CD.

6.3. Casos de Teste Acessíveis:

* Validar a navegação com teclado, uso de leitores de tela, e 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 que devem ser incluídos no conjunto de regressão.
* Priorizar casos de teste críticos e que impactam funcionalidades principais.

7.2. Automação de Regressão:

* Criar scripts de regressão automatizados.
* Executar scripts de regressão a cada build ou pull request.

7.3. Planejamento de Execução:

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

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.

8.2. Testes de Contrato:

* Definir como garantir que as APIs respeitam os contratos esperados.
* Utilizar ferramentas como Pact para realizar testes de contrato.

8.3. Testes de Carga em APIs:

* Realizar testes de carga especificamente para as APIs.
* Utilizar ferramentas como k6 para simular carga pesada em APIs.

Organização:

Todos os arquivos e pastas serão organizados de forma clara e acessível para facilitar a execução e manutenção dos testes. Os relatórios e artefatos serão armazenados em locais facilmente acessíveis para revisão posterior.

Observações:

* Este plano de teste é um exemplo e pode ser adaptado para atender às necessidades específicas do projeto.
* É importante realizar revisões periódicas do plano de teste e atualizá-lo conforme necessário.
* A comunicação e colaboração entre os membros da equipe de teste são essenciais para o sucesso do projeto.