Documentação do Projeto: Página de Inscrição e Login - Programa Trilhas

1. Introdução e Objetivo

Este projeto implementa o frontend de um sistema de inscrição e login para o Programa Trilhas. O objetivo principal é fornecer uma interface clara, segura e fácil de usar para que candidatos interessados possam se registrar e participar do processo seletivo, otimizando o processo de inscrição.

O sistema visa resolver problemas comuns em processos de inscrição, como desorganização, erros de preenchimento e dificuldades de comunicação, através de um formulário web interativo com validação de dados no lado do cliente.

2. Público-Alvo

- **Candidatos:** Interessados em ingressar no mercado de tecnologia da informação através do Programa Trilhas.
- **Administradores:** Responsáveis por gerenciar e analisar as inscrições recebidas.

3. Tecnologias Utilizadas

- **HTML5:** Utilizado para a estruturação semântica do conteúdo da página, incluindo os formulários de login e cadastro com seus respectivos campos (<input>, <select>, <label>, <button>, etc.).
- **CSS3:** Responsável pela estilização visual completa da página, layout dos formulários, responsividade básica (flex-wrap), aparência dos campos, botões, seleção de trilhas e feedback visual para erros (classe .error, .error__message). Inclui:
    - reset.css: Para normalizar estilos padrão entre navegadores.
    - style.css: Contém os estilos específicos do projeto, utilizando Flexbox para layout e importando fontes customizadas (Poppins via Google Fonts). A classe .hidden é usada para controlar a visibilidade dos formulários.
- **JavaScript (ES6+):** Linguagem fundamental para a interatividade e lógica do lado do cliente (js/script.js):
    - **Manipulação do DOM:** Seleção de elementos (getElementById, querySelector), modificação de classes (classList), alteração de conteúdo (textContent) e estilos (style.display).
    - **Controle de Eventos:** Captura de interações do usuário como cliques em links (click em #showRegister, #showLogin) e submissão de formulários (submit nos formulários #login e #register), utilizando addEventListener e event.preventDefault().
    - **Validação de Formulários:** Implementa validações client-side para garantir que os dados inseridos estejam corretos antes de simular o envio. Isso inclui:
        - Verificação de campos obrigatórios.
        - Validação de formato (Email, Idade).

        - Validação de tamanho mínimo (Senha).
        - Verificação de seleção (Sexo, Trilha).
        - Verificação de upload de arquivos.
        - Confirmação de aceite dos Termos e Condições.
    - **Feedback de Erro:** Exibe mensagens de erro específicas (showError) próximas aos campos inválidos (usando span.error__message) e remove-as (removeError) quando corrigido.
    - **Web Storage API (localStorage):** Utilizado para **simular** o armazenamento e recuperação de dados de usuários cadastrados localmente no navegador. **Nota Importante:** Conforme destacado no README.md e comentários no código, o uso de localStorage para dados sensíveis como senhas é **inseguro** e foi adotado **apenas para fins demonstrativos** neste projeto.
- **Assets:** Imagens (assets/img/) são utilizadas para ícones (upload, trilhas, logo, ilustração).

4. Estrutura do Projeto (Arquivos Principais)

```
Projeto desafio3/
│
├── assets/
│   ├── css/
│   │   ├── reset.css       # Normalização de estilos CSS
│   │   └── style.css       # Estilos principais do projeto
│   └── img/
│       ├── *.svg         # Imagens e ícones utilizados (front-end.svg, back-end.svg, etc.)
│
├── js/
│   └── script.js         # Lógica JavaScript (validação, DOM, localStorage)
│
├── index.html            # Arquivo principal HTML com os formulários
├── Documentação da Página de Inscrição.docx  # Documento original com requisitos
└── README.md             # Documentação técnica e instruções
```

5. Funcionalidades Principais Implementadas

- **Formulário de Cadastro (#registerForm):**
    - Coleta informações detalhadas do participante: Nome Completo, Idade, CPF, Sexo, Email, Telefone. [6]
    - Permite o upload (simulado no frontend) de Documento de Identidade e Comprovante de Residência (input type="file"). [6, 7]
    - Coleta dados de Endereço Residencial: CEP, Rua, Número, Cidade, Estado. [6, 7]
    - Permite a seleção de **uma** Trilha de Aprendizagem (Front-end, Back-end, Jogos, UX/UI, Dados) usando input type="radio". [7]
    - Define credenciais de acesso: ID de Usuário e Senha (input type="text", input type="password"). [7]

    - Exige a confirmação de leitura e aceite dos Termos e Condições e Política de Privacidade (input type="checkbox"). [7]
    - **Validação Client-Side:** Verifica todos os campos antes de permitir o "cadastro" (salvamento no localStorage). Mensagens de erro são exibidas para campos inválidos (showError, removeError em script.js). [8]
    - **Botão Cancelar:** Limpa os dados preenchidos no formulário de cadastro (type="reset"). [8, 10]
    - **Botão "Fazer Inscrição":** Após validação bem-sucedida, salva os dados no localStorage (saveUser em script.js) e exibe mensagem de sucesso. Impede cadastro de userId duplicado. [8, 11]
- **Formulário de Login (#loginForm):**
    - Permite que usuários "cadastrados" (no localStorage) façam login usando ID de Usuário e Senha.
    - Valida se os campos foram preenchidos.
    - Verifica as credenciais contra os dados armazenados no localStorage.
    - Exibe mensagem de sucesso ou erro.
- **Alternância entre Formulários:**
    - Links "Cadastre-se" (#showRegister) e "Faça login" (#showLogin) permitem alternar a visualização entre os formulários de login e cadastro sem recarregar a página, utilizando a função toggleForms em script.js que manipula a classe CSS .hidden.

6. Fluxo Lógico de Uso

1. O usuário acessa a página (index.html), visualizando inicialmente o formulário de login (#loginForm).
2. **Para Cadastrar:**
    - Clica no link "Cadastre-se" (#showRegister).
    - O formulário de login é oculto e o formulário de cadastro (#registerForm) é exibido (toggleForms).
    - O usuário preenche todos os dados solicitados no formulário de cadastro. [9]
    - Ao tentar submeter ("Fazer Inscrição"), o JavaScript (script.js) valida todos os campos.
    - Se houver erros, mensagens são exibidas e o envio é bloqueado. [8]
    - Se desejar limpar tudo, clica em "Cancelar". [10]
    - Se todos os dados estiverem válidos, clica em "Fazer Inscrição". Os dados são salvos no localStorage, uma mensagem de sucesso é exibida e o formulário é limpo. [11]
3. **Para Fazer Login:**
    - Na tela inicial (ou clicando em "Faça login" na tela de cadastro), o usuário preenche o ID de Usuário e Senha no formulário #loginForm.
    - Clica em "Entrar".

    - O JavaScript verifica se os campos estão preenchidos e se as credenciais correspondem a algum usuário no localStorage.
    - Uma mensagem de sucesso ou erro ("ID do usuário ou senha incorretos") é exibida.

7. Detalhes da Implementação (Código)

- **HTML (index.html):**
    - Estrutura a página com main, section (formulário) e aside (imagens).
    - Utiliza IDs (#loginForm, #registerForm, #nome, #cpf, etc.) para manipulação via JavaScript.
    - Define os tipos de input adequados (text, email, number, password, file, radio, checkbox, tel).
    - Inclui span.error__message após (ou próximo a) cada campo para exibir mensagens de erro de validação.
    - Linka os arquivos CSS (reset.css, style.css) no <head> e o arquivo JavaScript (script.js) no final do <body>.
- **CSS (style.css):**
    - Define o layout principal usando Flexbox (.container, .container__formulario).
    - Estiliza todos os elementos do formulário (labels, inputs, selects, botões, áreas de upload, cards de trilha).
    - Implementa a classe .hidden { display: none; } para ocultar/mostrar os formulários.
    - Define estilos para o estado de erro (input.error, select.error, .error__message).
- **JavaScript (script.js):**
    - Utiliza DOMContentLoaded para garantir que o script execute após o HTML ser carregado.
    - Organizado em funções auxiliares (toggleForms, validateEmail, validateAge, showError, removeError, saveUser, validateField).
    - Implementa a lógica de validação completa no handler do evento submit do formulário de registro, percorrendo cada campo e aplicando regras específicas.
    - Manipula o localStorage para persistir os dados do usuário (cadastro) e verificar credenciais (login).
    - Previne o envio padrão do formulário (e.preventDefault()) para controlar o fluxo com JavaScript.
    - Gerencia a alternância visual entre os formulários de login e cadastro.

8. Considerações Finais e Melhorias

- **Segurança:** O uso de localStorage para armazenar senhas **não é seguro** e só foi utilizado para fins didáticos. Em uma aplicação real, a

autenticação e o armazenamento de dados devem ser gerenciados por um backend seguro com hashing de senhas.

- **Escalabilidade:** O projeto frontend está bem estruturado. Para uma aplicação completa, seria necessário desenvolver um backend (ex: Node.js, Python/Django, Java/Spring) para:
  - Receber os dados do formulário.
  - Validar dados no servidor (validação dupla é essencial).
  - Armazenar dados de forma segura em um banco de dados.
  - Gerenciar sessões de usuário autenticadas.
  - Lidar com o upload real de arquivos.
- **Validação:** Validações mais robustas podem ser adicionadas (ex: formato de CPF, busca de CEP via API, complexidade de senha).