

Conecta Maranhão: Plataforma de Conexão entre Empresas e Participantes

Sao Luis/Maranhao

Junho de 2025

Sumário

1. Justificativa da Arquitetura Utilizada.....	2
2. Estratégias de Validação de Dados e Tratamento de Erros.....	3
3. Componentização.....	3
4. Estrutura da aplicação.....	5
5. Feedback ao Usuário.....	10
6. Links Importantes.....	11
7. Bibliotecas Externas Utilizadas.....	12
8. Participação da Equipe.....	13

1. Justificativa da Arquitetura Utilizada

Optamos por utilizar a arquitetura baseada em componentes com React e TypeScript, pois permite reutilização de código, melhor organização e manutenção da aplicação. A tipagem estática do TypeScript aumentou a segurança no desenvolvimento, evitando muitos erros em tempo de compilação. Também estruturamos o projeto seguindo a divisão em páginas, componentes reutilizáveis e serviços (API), com diretórios separados para melhor organização.

A biblioteca axios foi escolhida por sua simplicidade e eficiência na realização de requisições HTTP (GET, POST, PUT, DELETE), facilitando a integração com APIs externas. Ela oferece uma sintaxe mais enxuta que a função nativa fetch, além de suporte automático a JSON, tratamento de erros, interceptadores e cabeçalhos personalizados, o que contribui para a organização e segurança do código.

A biblioteca react-router-dom foi utilizada para criar e gerenciar rotas da aplicação, permitindo a navegação entre páginas de forma declarativa e eficiente. Ela possibilita a definição de caminhos personalizados, navegação dinâmica, parâmetros de rota e controle de redirecionamentos, sendo a solução mais amplamente utilizada e mantida pela comunidade React para esse propósito.

Vite foi adotado como ferramenta de desenvolvimento por sua velocidade de inicialização, recarregamento instantâneo (Hot Module Replacement) e integração nativa com TypeScript e React. Ao contrário de ferramentas mais pesadas como Webpack, o Vite utiliza ES Modules no ambiente de desenvolvimento, proporcionando uma experiência mais ágil e produtiva para os desenvolvedores.

2. Estratégias de Validação de Dados e Tratamento de Erros

A validação dos dados dos formulários foi feita com validações manuais em TypeScript e, em alguns casos, com auxílio de bibliotecas. Utilizamos try/catch em requisições assíncronas para capturar e tratar erros de forma apropriada. Mensagens de erro amigáveis são exibidas ao usuário em casos como falha de login, campos inválidos ou erro de rede.

A biblioteca react-imask foi adotada no projeto para facilitar o processo de formatação automática de dados inseridos pelo usuário, como CPF, telefone, datas e outros formatos específicos. Seu uso elimina a necessidade de implementações manuais complexas com expressões regulares, garantindo maior consistência nos dados e melhor experiência ao usuário. Além disso, por ser um wrapper do imaskjs voltado especificamente para aplicações em React, ela se integra de forma nativa aos componentes, mantendo o padrão de desenvolvimento do projeto e simplificando a manutenção do código.

3. Componentização

A aplicação foi construída adotando o paradigma de componentização do React, utilizando a linguagem TypeScript como base para garantir maior segurança na tipagem, melhor manutenção do código e maior previsibilidade durante o desenvolvimento. Toda a estrutura foi planejada com o objetivo de promover a reutilização de elementos, a organização dos arquivos e a escalabilidade do sistema, permitindo que, futuramente, novas funcionalidades sejam adicionadas com menor esforço e menor impacto em partes já implementadas.

Na arquitetura do projeto, foram definidos componentes reutilizáveis, páginas principais e módulos especializados. Entre os componentes reutilizáveis, destaca-se o InputField, responsável por criar campos de entrada de formulário com label, placeholder, tratamento de erros e integração com os esquemas de validação definidos na aplicação. Outro componente essencial é o

Button, um botão genérico utilizado em ações diversas como login, cadastro e envio de formulários. Foi criado também o NavBar, componente que implementa a barra de navegação presente em várias páginas do sistema, garantindo consistência visual e facilidade de acesso entre as rotas. Além disso, desenvolveu-se o UserCard, que apresenta o perfil de um candidato, exibindo seu nome, biografia, links e conquistas de forma organizada e visualmente agradável.

Em relação às páginas principais, a aplicação conta com a LoginPage, onde ocorre o processo de autenticação dos usuários; a CadastroPage, destinada ao registro de novos usuários, contemplando as etapas de criação de conta e preenchimento do perfil completo; a PerfilPage, responsável pela exibição e edição dos dados pessoais do usuário logado; a EmpresaVagasPage, que disponibiliza para as empresas uma visualização dos candidatos cadastrados; e as páginas de Cursos e Seletivos, criadas para exibir as oportunidades disponíveis de maneira clara e segmentada.

Para complementar essa estrutura, foram desenvolvidos hooks e módulos especializados. Um exemplo é o userAuth, hook customizado que gerencia funções de login, autenticação e persistência do usuário, além do api.ts, módulo centralizador das funções de requisições HTTP à API, organizando e padronizando o acesso aos endpoints. Outro recurso importante é o ProtectedRoute, um componente que protege rotas privadas, redirecionando automaticamente os usuários não autenticados, garantindo a segurança das informações restritas.





No que diz respeito à integração com a API, a comunicação foi feita por meio de requisições RESTful utilizando a função fetch e funções assíncronas encapsuladas no arquivo api.ts, que concentra todas as chamadas e abstrações para os endpoints. As principais integrações implementadas incluíram a criação de usuários, por meio do endpoint POST /usuarios, a associação do usuário ao seu tipo, candidato ou empresa, pelo POST /contas, o cadastro completo do perfil de candidato através do POST /pessoas e a consulta geral de todos os candidatos cadastrados, utilizada pelas empresas, via GET /pessoas. Todas as

requisições contaram com tratamento de erros por meio de blocos try/catch e tiveram seus dados tipados com interfaces do TypeScript, garantindo consistência na manipulação e exibição no front-end. Dessa forma, a integração ficou mais segura, compreensível e de fácil manutenção para futuras expansões.

4. Estrutura da Aplicação

- 📁 public
 - └ 📄 vite.svg
- 📁 src
 - └ 📁 assets
 - | └ 📁 About
 - | └ 📁 ImageHome
 - | └ 📁 Medalhas
 - | └ 📄 logo.svg
 - | └ 📄 react.svg
 - └ 📁 components
 - └ 📁 Cadastro
 - └ 📁 Cammon
 - └ 📁 Cursos
 - └ 📁 Empresas
 - └ 📁 Home
 - └ 📁 Layout
 - └ 📁 Login
 - └ 📁 PerfilUsuario
 - └ 📁 Ranking
 - └ 📁 hooks
 - └ 📁 interfaces
 - └ 📁 pages
 - | └ 📁 About

- | | └─ AboutPage.tsx
- | └─ Aplicacoes
 - | └─ AplicacoesUserPage.tsx
 - | └─ VagasEmpresaPage.tsx
- | └─ Conquistas
 - | └─ Conquistas.tsx
- | └─ Cursos
 - | └─ CursoRegistro.tsx
 - | └─ CursosPage.tsx
- | └─ Empresas
 - | └─ EmpresasPage.tsx
- | └─ Login
 - | └─ CadastroPage.tsx
 - | └─ LoginPage.tsx
- | └─ Usuario
 - | └─ PerfilUsuario.tsx
- └─ routes
- └─ services
- └─ utils
 - └─ App.css
 - └─ App.tsx
 - └─ index.css
 - └─ main.tsx
 - └─ vite-env.d.ts
- └─ .gitignore
- └─ .hintrc
- └─ README.md
- └─ eslint.config.js
- └─ index.html
- └─ package-lock.json
- └─ package.json

-  tsconfig.app.json
-  tsconfig.json
-  tsconfig.node.json
-  vite.config.ts

4.1 Estrutura de Diretórios e Arquivos

src/

Diretório principal do código-fonte da aplicação.

assets/

Contém arquivos estáticos como imagens e ícones utilizados na interface.

react.svg: ícone SVG usado como logotipo ou elemento gráfico da interface.

pages/

Estrutura organizada por módulos/páginas da aplicação. Cada subpasta representa uma área funcional.

About/

AboutPage.tsx: Componente da página "Sobre", com informações institucionais ou gerais sobre a plataforma.

Aplicacoes/

AplicacoesUserPage.tsx: Página que exibe as aplicações (candidaturas) feitas por um usuário.

VagasEmpresaPage.tsx: Página que exibe as vagas disponibilizadas por uma empresa.

Conquistas/

Conquistas.tsx: Página com conquistas, selos ou metas atingidas por usuários, vinculadas a ações na plataforma.

Cursos/

CursoRegistro.tsx: Página de registro ou criação de um novo curso.

CursosPage.tsx: Página de listagem ou visualização geral de cursos disponíveis.

Empresas/

EmpresasPage.tsx: Página com funcionalidades relacionadas às empresas cadastradas (perfil, gerenciamento etc.).

Login/

CadastroPage.tsx: Página de cadastro de novos usuários.

LoginPage.tsx: Página de autenticação (login) da plataforma.

Usuario/

PerfilUsuario.tsx: Página de visualização e edição do perfil do usuário logado.

- **App.tsx**

Componente principal da aplicação. Define a estrutura base, como as rotas e a renderização condicional das páginas.

- **App.css e index.css**

Arquivos de estilo global. App.css contém estilos específicos da aplicação, enquanto index.css aplica resets e estilos globais iniciais.

- **main.tsx**

Ponto de entrada da aplicação. Renderiza o componente `<App />` dentro da árvore DOM no index.html.

- **vite-env.d.ts**

Declarações de tipos para que o Vite compreenda arquivos específicos do projeto.

4.2 Arquivos de Configuração na Raiz

- **.gitignore**

Define os arquivos e pastas que devem ser ignorados pelo Git, como `node_modules/` e arquivos temporários.

- **README.md**

Arquivo de documentação inicial do projeto. Contém informações como objetivo, instruções de instalação e uso.

- **eslint.config.js**

Configuração do ESLint, ferramenta de análise estática para manter o código consistente e evitar erros comuns.

- **index.html**

Template HTML da aplicação. Contém a div root onde a aplicação React é injetada.

- **package.json & package-lock.json**

Arquivos de controle de dependências do projeto. package.json define os

pacotes e scripts, e package-lock.json registra suas versões exatas.

- **tsconfig.json, tsconfig.app.json, tsconfig.node.json**

Arquivos de configuração do TypeScript:

- tsconfig.json: configuração base geral.
- tsconfig.app.json: configuração específica para o app.
- tsconfig.node.json: usada para scripts Node.js, como configurações de build ou testes.

- **vite.config.ts**

Arquivo de configuração do Vite, ferramenta de build e desenvolvimento rápido usada neste projeto.

5. Feedback ao Usuário

A aplicação foi projetada para fornecer feedbacks claros e objetivos aos usuários em cada ação realizada, visando garantir maior usabilidade, compreensão dos processos e confiança no sistema. Desde as etapas iniciais de login e cadastro até as funcionalidades de visualização de candidatos e cursos, foram implementados diversos mecanismos de comunicação entre a interface e o usuário.

Um dos principais recursos utilizados são as mensagens de sucesso, que aparecem ao concluir ações como registro de novos usuários, atualização de perfil ou envio de formulários. Essas mensagens têm o objetivo de confirmar ao usuário que a operação foi realizada corretamente, como por exemplo a exibição de mensagens do tipo “Cadastro realizado com sucesso” ou “Login efetuado com sucesso”, geralmente apresentadas em alertas.

Além disso, foram implementadas mensagens de erro, que informam de maneira clara quando alguma operação não pôde ser concluída, seja por falha na comunicação com a API, erro interno ou dados inválidos inseridos pelo usuário. Nestes casos, as mensagens orientam o usuário sobre o problema

encontrado e, quando possível, indicam como corrigi-lo, por exemplo: “Erro ao tentar cadastrar usuário. Verifique seus dados e tente novamente” ou mensagens específicas em campos de formulário, como “O email inserido é inválido” e “Senha deve ter no mínimo 6 caracteres”.

Outro aspecto fundamental do feedback ao usuário está na validação dos formulários, que ocorre tanto antes do envio quanto durante o preenchimento, por meio de validação instantânea em cada campo. Dessa forma, ao inserir dados incorretos ou deixar campos obrigatórios em branco, o usuário é imediatamente informado por meio de mensagens explicativas e pela mudança visual nos campos, como bordas vermelhas ou ícones de alerta, prevenindo erros antes de enviar os dados para a API.

Para ações assíncronas que levam mais tempo de processamento, como login, cadastro e carregamento de listas de candidatos ou cursos, foram implementados indicadores de carregamento, como spinners ou estados de botão com texto “Carregando...”. Esses elementos têm como objetivo sinalizar ao usuário que a requisição está em andamento, evitando cliques repetidos ou confusão quanto ao estado da aplicação.

Em situações em que uma ação bem-sucedida requer mudança de contexto, foram configurados redirecionamentos automáticos, funcionando como feedback implícito. Por exemplo, após realizar login com sucesso, o usuário é encaminhado diretamente para a página de perfil ou dashboard, demonstrando que a autenticação foi concluída corretamente e permitindo continuidade no fluxo de uso.

Também foram implementados feedbacks visuais nos botões e campos, como mudanças de cor ao passar o mouse (hover), alterações de cor ao clicar e estados desativados quando a ação está em processamento. Esses pequenos detalhes garantem maior percepção de interatividade, usabilidade e profissionalismo na interface.

De forma geral, todos esses recursos de feedback ao usuário foram pensados para tornar a experiência mais intuitiva, segura e satisfatória,

reduzindo dúvidas durante o uso da aplicação e prevenindo erros operacionais que comprometem a eficiência do sistema.

6. Links Importantes

-Repositório no GitHub: <https://github.com/ConectaMaranhao/Front-End>

-Aplicação Publicada(GitHub Pages):

<https://conectamaranhao.github.io/Front-End/>

7. Bibliotecas Externas Utilizadas

Durante o desenvolvimento da aplicação, foram utilizadas algumas bibliotecas externas que desempenharam papéis importantes para garantir uma estrutura eficiente, modular e de fácil manutenção. Abaixo, destacam-se as principais:

React: Biblioteca JavaScript principal utilizada para construção da interface do usuário de forma declarativa e baseada em componentes. Foi usada em conjunto com TypeScript para garantir tipagem estática e mais segurança no desenvolvimento.

React-imask: Biblioteca para React, ela é um wrapper (envoltório) para a biblioteca imaskjs, permitindo que você utilize máscaras de entrada (input masks) de forma fácil em componentes React.

React-icons foi usada para adicionar ícones ao projeto de forma prática. Ela reúne vários conjuntos de ícones em um só lugar, permitindo importar apenas o necessário. Foi utilizada principalmente em botões e menus para melhorar a interface, com fácil personalização de tamanho e cor.

Axios: Utilizada para realizar requisições HTTP de forma simples e eficiente. Facilitou a comunicação com a API externa que fornece os dados dos usuários e empresas. Axios foi essencial para o consumo de endpoints e tratamento de respostas assíncronas.

Vite: Ferramenta de build utilizada como ambiente de desenvolvimento moderno e rápido. Proporcionou uma configuração simples, carregamento instantâneo (hot reload) e excelente performance durante o desenvolvimento da aplicação.

CSS Modules: Apesar de não ser uma biblioteca externa instalada via npm, é uma abordagem de escopo local para os estilos CSS, que foi utilizada para organizar os estilos de cada componente separadamente, evitando conflitos e mantendo a manutenção mais simples.

O uso dessas bibliotecas garantiu produtividade no desenvolvimento, reutilização de código e uma base sólida para futuras melhorias no projeto.

8. Participação da Equipe

Durante o desenvolvimento do front-end da plataforma, foram implementadas funcionalidades essenciais que compõem o fluxo principal de uso do sistema, com foco na divisão organizada das responsabilidades para garantir qualidade técnica, integração entre as telas e consistência visual em toda a aplicação.

Foram desenvolvidas as funcionalidades de login e cadastro, abrangendo tanto usuários quanto empresas. Essa implementação contemplou as páginas LoginPage.tsx e CadastroPage.tsx, que contêm formulários com validação de campos obrigatórios, garantindo a integridade dos dados inseridos antes do envio. Também foi realizada a integração com a API, simulando o processo de autenticação e registro para viabilizar o fluxo completo de acesso durante os testes e validações.

Além disso, foi criada a página de perfil do usuário, registrada no arquivo PerfilUsuario.tsx, que apresenta os dados pessoais e profissionais cadastrados, como nome, experiências anteriores, competências, formações e outras informações relevantes para o currículo. Essa tela foi estruturada com layout responsivo, visando boa visualização em diferentes dispositivos, e está

preparada para receber funcionalidades futuras de edição e atualização dos dados inseridos.

Na sequência, foi desenvolvida a tela de cursos, que contempla tanto a visualização dos cursos disponíveis quanto o cadastro de novos cursos na plataforma. As funcionalidades dessa área foram divididas entre os arquivos `CursosPage.tsx`, responsável pela listagem dos cursos, e `CursoRegistro.tsx`, que contém o formulário para criação de novos cursos. A interface dessas páginas inclui feedbacks visuais claros e objetivos, proporcionando uma boa experiência ao usuário durante o processo de cadastro e consulta.

Foram implementadas também as telas relacionadas aos processos seletivos, incluindo a interface onde os usuários podem visualizar vagas disponíveis e se inscrever nelas, bem como a tela para cadastro de novas vagas pelas empresas. Essas funcionalidades foram estruturadas nos arquivos `AplicacoesUserPage.tsx` e `VagasEmpresaPage.tsx`, sendo implementada também a lógica que associa os usuários às vagas às quais se candidataram, garantindo o correto vínculo das informações na base de dados simulada pela API.

Outro recurso implementado foi a visualização de candidatos pelas empresas, permitindo que acessem a lista de inscritos em suas vagas de forma organizada. Essa funcionalidade foi incorporada na página `VagasEmpresaPage.tsx` e apresenta os candidatos em cards, com a implementação inicial de filtros para facilitar a análise por parte das empresas, tornando o processo seletivo mais eficiente e ágil.

Também foi iniciada a criação da tela em que o candidato poderá acompanhar o andamento do processo seletivo. Essa página está em desenvolvimento, buscando apresentar dinamicamente as etapas do processo conforme a vaga progride, e será integrada ao componente de aplicações do usuário assim que finalizada.

Por fim, foi proposta e iniciada a implementação da funcionalidade de gamificação da plataforma, visando o aumento do engajamento por meio de conquistas, medalhas e acúmulo de experiência (XP) pelos usuários. Essa funcionalidade está registrada no arquivo Conquistas.tsx e encontra-se em fase inicial, podendo ser expandida em futuras versões do projeto. Além disso, foi desenvolvida a página institucional “Sobre o Projeto”, localizada no arquivo AboutPage.tsx, que apresenta informações sobre os objetivos, funcionamento e benefícios esperados da plataforma. A estilização dessa página buscou transmitir uma identidade visual clara, acessível e leve, alinhada ao propósito social da iniciativa.