Documentação técnica FrontEnd Música de Fundo

|  |  |  |
| --- | --- | --- |
| Data de alteração do documento. | Alteração específica do documento. | Responsável pela alteração. |
| 03/12/2023 | Elaboração do documento. | Fernando Gonçalves de Amorim |
| 10/12/2023 | Formatação e adição de configuração | Fernando Gonçalves de Amorim |

Este documento fornece uma visão geral técnica do frontend em React, destacando a estrutura de diretórios, configurações principais e implementação das funcionalidades.

Sumário

[1 – Framework React 3](#_Toc153126983)

[1.2 – Instalação e execução: 4](#_Toc153126984)

[2 – Funcionamento e descrição dos arquivos. 4](#_Toc153126985)

[2.1 - Descrição Técnica da Pasta src no React Vite 4](#_Toc153126986)

[2.1.1 - Estrutura Padrão 5](#_Toc153126987)

[2.1.2 - Organização Modular 5](#_Toc153126988)

[2.1.3 - Vantagens do Vite 5](#_Toc153126989)

[2.2 – Conexão com o backend 5](#_Toc153126990)

[2.2.2 – Arquivo request.js (Biblioteca Axios) 6](#_Toc153126991)

[2.2.3 - Arquivo request.js (Funções principais) 6](#_Toc153126992)

[3 - Telas: 9](#_Toc153126993)

Texto

Descrição gerada automaticamente

Figura - Estrutura e organização dos arquivos do frontend

# 1 – Framework React

React, também conhecido como React.js ou ReactJS, é uma biblioteca de JavaScript de código aberto mantida pelo Facebook. Lançado em 2013, o React foi projetado para facilitar o desenvolvimento de interfaces de usuário interativas e reativas para aplicações web. Sua abordagem inovadora e eficiente para construir componentes de interface de usuário dinâmicos tornou-o rapidamente uma escolha popular entre desenvolvedores.

O principal conceito por trás do React é a criação de interfaces de usuário declarativas e componentizadas. Em vez de manipular diretamente o DOM (Modelo de Objeto de Documento), o React introduz um modelo de programação baseado em componentes, onde a interface de usuário é composta por pequenas partes reutilizáveis chamadas de "componentes". Esses componentes podem ser combinados para formar interfaces complexas, proporcionando uma arquitetura modular e fácil de entender.

Uma das características marcantes do React é o uso do conceito de "Virtual DOM". Em vez de atualizar diretamente o DOM cada vez que ocorre uma mudança no estado da aplicação, o React cria uma representação virtual do DOM em memória. Esse "Virtual DOM" é mais eficiente e permite ao React calcular as alterações mínimas necessárias para atualizar a interface de usuário. Esse processo otimizado, conhecido como reconciliação, resulta em um desempenho significativamente melhor em comparação com abordagens tradicionais.

O React é frequentemente utilizado em conjunto com outras bibliotecas e frameworks, como o Redux para gerenciamento de estado, React Router para navegação em aplicações de página única (SPA), e diversas ferramentas para construção e empacotamento de código, como o Webpack.

Em resumo, o React proporciona uma abordagem eficiente e escalável para o desenvolvimento de interfaces de usuário dinâmicas e reativas em aplicações web, sendo amplamente adotado pela comunidade de desenvolvedores devido à sua flexibilidade, desempenho e facilidade de uso.

## 1.2 – Instalação e execução:

As instruções detalhadas para a instalação e execução do projeto podem ser encontradas no arquivo README disponível no repositório oficial do projeto no GitHub. O README serve como um guia abrangente, fornecendo todas as informações necessárias para configurar o ambiente de desenvolvimento e iniciar o aplicativo sem problemas. Este arquivo é essencial para desenvolvedores que desejam contribuir para o projeto ou simplesmente explorar suas funcionalidades.

Além disso, as diretrizes de instalação e execução também estão disponíveis na página inicial do projeto. Ao visitar a página inicial, os usuários podem encontrar um resumo das etapas necessárias para configurar o ambiente e iniciar o aplicativo. Essa abordagem visa fornecer informações acessíveis a qualquer pessoa interessada no projeto, independentemente de sua experiência técnica.

Essa dualidade de fontes de informação, tanto no README do GitHub quanto na página inicial do projeto, oferece flexibilidade aos usuários, permitindo que escolham a plataforma que considerem mais conveniente. Seja navegando no repositório do GitHub para obter detalhes técnicos ou visitando a página inicial para uma visão mais geral, as informações necessárias para a instalação e execução estão prontamente disponíveis, garantindo uma experiência tranquila para todos os interessados no projeto.

# 2 – Funcionamento e descrição dos arquivos.

## 2.1 - Descrição Técnica da Pasta src no React Vite

A pasta src no projeto React Vite desempenha um papel crucial ao abrigar o código-fonte principal da aplicação. Esta estrutura organizacional é vital para a clareza e modularidade do projeto, facilitando o desenvolvimento, manutenção e escalabilidade.

### 2.1.1 - Estrutura Padrão

Dentro da pasta src, é comum encontrar subdiretórios e arquivos essenciais que compõem a base da aplicação. Alguns elementos fundamentais incluem:

- Components (Componentes): O diretório components abriga componentes reutilizáveis da aplicação. Esses componentes podem variar de simples elementos UI a componentes mais complexos que encapsulam lógica específica.

- Pages (Páginas): O diretório pages é destinado às páginas da aplicação, cada uma representando uma rota específica. A estrutura facilita a navegação e gerenciamento de rotas na aplicação.

- App.js / main.jsx: Este arquivo serve como o ponto de entrada da aplicação, onde os componentes principais são normalmente renderizados. Ele atua como o ponto de convergência para estruturar a interface inicial da aplicação.

- data / img: Pastas que armazenam recursos estáticos, como imagens, fontes ou outros arquivos que a aplicação possa necessitar.

### 2.1.2 - Organização Modular

A estrutura da pasta src promove a organização modular do código-fonte. Componentes são isolados em seus próprios diretórios, facilitando a reutilização e manutenção. O mesmo princípio se aplica às páginas, proporcionando uma visão clara e distinta da estrutura de navegação da aplicação.

### 2.1.3 - Vantagens do Vite

O React Vite, sendo uma ferramenta de desenvolvimento rápida, utiliza a pasta src de maneira eficiente. A arquitetura de importação dinâmica do Vite permite um carregamento mais rápido dos módulos, resultando em um ambiente de desenvolvimento extremamente responsivo.

## 2.2 – Conexão com o backend

A conexão com o backend é feita através de requisições GET e POST. Todas as chamadas possuem uma função especifica para criar, buscar, editar ou apagar um registro no endpoint do backend.   
 Para realizar as chamadas, não é necessário autenticação, apenas o envio de um JSON com as informações desejadas para a URL específica. Há também uma coleção do postman com todas as chamadas possíveis para o backend, que pode ser encontrada na pasta Documentações no repositório de backend.  
  
2.2.1 – Arquivo request.js (Introdução)

O arquivo request.js encontrado dentro da pasta src/componentes possui todas as requisições para o backend no formato de funções, aonde cada função recebe parâmetros específicos que interagem de forma única em cada script. Este formato facilita a implantação e manutenção do código, visto que, todos os códigos que possuem ligação com o backend estão em um único arquivo que não se repetem.

### 2.2.2 – Arquivo request.js (Biblioteca Axios)

Axios é uma biblioteca popular de JavaScript utilizada para realizar requisições HTTP em ambientes tanto de navegador quanto de servidor. Criada por Matt Zabriskie, o Axios simplifica o processo de comunicação entre um aplicativo e um servidor, oferecendo uma interface limpa e intuitiva para lidar com operações assíncronas.

O Axios é especialmente reconhecido por sua facilidade de uso e pela consistência de sua API. Ele oferece uma sintaxe simples e elegante para realizar requisições HTTP, suportando métodos como GET, POST, PUT, DELETE, entre outros. Além disso, o Axios é capaz de manipular automaticamente solicitações e respostas JSON, simplificando ainda mais a interação com APIs baseadas nesse formato.

Uma característica importante do Axios é sua capacidade de lidar com promessas, tornando o código assíncrono mais legível e fácil de gerenciar. Ele retorna promessas JavaScript nativas, o que permite o uso de métodos como .then() e .catch() para manipular os resultados das requisições.

Outro ponto forte do Axios é sua compatibilidade com diversos ambientes, incluindo navegadores, Node.js e ambientes de execução JavaScript assíncronos. Essa versatilidade torna o Axios uma escolha popular para desenvolvedores que precisam realizar chamadas de API de maneira eficiente em diferentes contextos.

Em resumo, o Axios é uma biblioteca confiável e eficaz para realizar requisições HTTP em projetos JavaScript, proporcionando uma interface amigável, suporte a promessas e compatibilidade com diversos ambientes, o que contribui para uma experiência de desenvolvimento mais fluida e eficiente.

### 2.2.3 - Arquivo request.js (Funções principais)

Como citado nos tópicos acima, cada requisição para o backend possui uma função especifica, estas funções recebem parâmetros próprios que são utilizados para a estruturação do JSON que será enviado para o endpoint do backend.  
  
2.2.3.1 – Funções de Questions

As funções de “Questions” possuem a finalidade de manipular os dados da tabela Questions no backend. Exitem 4 funções, que são:  
  
getQuestions: A função getQuestions é uma função assíncrona que utiliza a biblioteca Axios para fazer uma requisição HTTP do tipo GET a URL do backend. A função aceita um parâmetro subject, que é utilizado para filtrar as questões retornadas com base no texto da pergunta.  
A função pode ser utilizada da seguinte forma: getQuestions("Geografia"); Aonde getQuestions() é a invocação da função e "Geografia" é o parâmetro de pesquisa das perguntas. Assim a função retornará um objeto com elementos do tema Geografia.

addQuestions: A função cria uma pergunta na API com o texto da pergunta, a opção associada e o tipo de pergunta especificado.  
A função pode ser utilizada da seguinte forma: addQuestions("Onde está localizado a Europa", 1, "Geografia"); Aonde addQuestions() é a invocação da função, "Onde está localizado a Europa" é o primeiro parâmetro, que representa o texto da pergunta, 1 é o segundo parâmetro, que representa o id da opção e por fim "Geografia" é o terceiro parâmetro representando o tipo da pergunta.

updateQuestions: Atualiza uma pergunta existente na API com base no ID da pergunta, alterando seu texto, opção associada e tipo.  
A função pode ser utilizada da seguinte forma: updateQuestions(6, "Onde está localizado a Dinamarca", 1, "Geografia"); Aonde updateQuestions() é a invocação da função, 6 é o primeiro parâmetro que consiste no id da questão que vai ser editada, "Onde está localizado a Dinamarca" é o novo titulo da questão, 1 é o terceiro parâmetro, que representa o id da opção e por fim "Geografia" é o quarto parâmetro representando o tipo da pergunta. OBSERVAÇÃO: O primeiro parâmetro que é o ID da question é obrigatória.

deleteQuestions: Remove uma pergunta da API com base no ID da pergunta.  
A função pode ser utilizada da seguinte forma: deleteQuestions(11); Aonde deleteQuestions () é a invocação da função, e 11 é o id da question que vai ser detelada.

#### 2.2.3.2 – Funções de Introduction

As funções de “Questions” possuem a finalidade de manipular os dados da tabela Questions no backend. Exitem 4 funções, que são:

getIntroduction: Obtém introduções da API com base no tipo especificado.  
A função pode ser utilizada da seguinte forma: getIntroduction("Geografia"); Aonde getIntroduction() é a invocação da função e “Geografia” é o parâmetro de busca da introdução, será retornado um objeto com todas as introduções com o tema Geografia.

addIntroduction: Cria uma nova introdução na API com título, texto e tipo especificados.  
A função pode ser utilizada da seguinte forma: addIntroduction("França e suas Fronteiras", "Texto a ser trabalhado da França", "Geografia"); Aonde addIntroduction() é a invocação da função, "França e suas Fronteiras" é o primeiro parâmetro que consiste no titulo da introdução, o segundo parâmetro é "Texto a ser trabalhado da França", que consiste no texto que vai ser exibido e por fim o terceiro parâmetro "Geografia" que é o tipo do texto. Ao final a função retorna o objeto criado.

updateIntroduction: Atualiza uma introdução existente na API com base no ID da introdução, alterando título, texto e tipo.  
A função pode ser utilizada da seguinte forma: updateIntroduction(3, "Japão", "Texto a ser trabalhado no Japão", "Geografia"); Aonde updateIntroduction () é a invocação da função, 3 é o primeiro parâmetro que consiste no id da introdução que vai ser editada, " Japão " é o novo título da introdução e por fim "Geografia" é o terceiro parâmetro representando o tipo da introdução. OBSERVAÇÃO: O primeiro parâmetro que é o ID da Introduction é obrigatório.

deleteIntroduction: Remove uma introdução da API com base no ID da introdução.  
A função pode ser utilizada da seguinte forma: deleteIntroduction(2); Aonde deleteIntroduction () é a invocação da função, e 2 é o id da Introduction que vai ser detelada.

#### 2.2.3.3 – Funções de Option

As funções de “Option” possuem a finalidade de manipular os dados da tabela Options no backend. Exitem 4 funções, que são:

getOption: Obtém a ID da opção da API com base no texto da opção especificado.  
A função pode ser utilizada da seguinte forma: getOption('Segunda guerra mundial'); Aonde getOption () é a invocação da função e “'Segunda guerra mundial” é o parâmetro de busca da opção, será retornado um objeto com todas as introduções com este título.

addOption: Cria uma nova opção na API com texto e correção especificados.  
A função pode ser utilizada da seguinte forma: addOption("Primeira Guerra munidal", true); Aonde addOption() é a invocação da função, "Primeira Guerra munidal" é o titulo da opção e o segundo parâmetro é uma booleana (True ou False), que consiste em identificar se a questão é a correta ou não, caso a questão seja a correta, ela será cadastrada com o valor True.

updateOption: Atualiza uma opção existente na API com base na ID da opção, alterando texto e correção.  
A função pode ser utilizada da seguinte forma: updateOption(1, "Segunda guerra mundial", true); Aonde updateOption() é a invocação da função, 1 é o primeiro parâmetro que consiste no id da opção que vai ser editada, " Segunda guerra mundial " é o novo título da opção e por fim true é o terceiro parâmetro representando a booleana de uma resposta correta ou não. OBSERVAÇÃO: O primeiro parâmetro que é o ID da option é obrigatório.

deleteOption: Remove uma opção da API com base na ID da opção.  
A função pode ser utilizada da seguinte forma: deleteOption(3) ; Aonde deleteOption () é a invocação da função, e 3 é o id da option que vai ser detelada.

#### 2.2.3.4 – Funções de Reponse

As funções de “Responde” possuem a finalidade de manipular os dados da tabela Responde no backend. Exitem 4 funções, que são:

getResponse: Obtém respostas da API com base no ID do usuário.  
A função pode ser utilizada da seguinte forma: getResponse(1); Aonde getResponse() é a invocação da função e o ID da resposta desejada.

addResponse: Cria uma nova resposta na API com base no ID do usuário, ID da pergunta e opção selecionada.  
A função pode ser utilizada da seguinte forma: addResponse(62, 12, 8); Aonde addResponse() é a invocação da função o primeiro parâmetro 62 é o id do usuário, segundo parâmetro 12 é o id da questão e o terceiro parâmetro 8 é o id da opção que o usuário escolheu.

updateResponse: Atualiza uma resposta existente na API com base no ID da resposta, alterando ID do usuário, ID da pergunta e opção selecionada.  
A função pode ser utilizada da seguinte forma: updateResponse(1, 1, 5, 1); Aonde updateResponse() é é a invocação da função o primeiro parâmetro 1 é o id da Response, o segundo parâmetro 1 é o id do usuário, terceito parâmetro 5 é o id da questão e o quarto parâmetro 1 é o id da opção que o usuário escolheu. OBSERVAÇÃO: O primeiro parâmetro que é o ID da Responde é obrigatório.

deleteResponse: Remove uma resposta da API com base no ID da resposta.  
A função pode ser utilizada da seguinte forma: deleteResponse (3) ; Aonde deleteResponse () é a invocação da função, e 2 é o id da Response que vai ser detelada.

#### 2.2.3.5 – Funções de User

As funções de “User” possuem a finalidade de manipular os dados da tabela User no backend. Exitem 4 funções, que são:

getUser: Obtém informações do usuário da API com base no nome de usuário especificado.  
A função pode ser utilizada da seguinte forma: getUser("Luiza"); Aonde getUser() é a invocação da função e o nome do usuario desejadado.

addUser: Cria um novo usuário na API com nome, data de nascimento, gênero e email especificados. Retorna o usuário recém-criado.  
A função pode ser utilizada da seguinte forma: addUser("Manuel", "2000-11-29", "M", "teste@teste.com"); Aonde addUser() é a é a invocação da função, o primeiro parâmetro "Manuel" é o nome do usuário, segundo parâmetro "2000-11-29" é a data de nascimento, terceito parâmetro “M” é o Gênero do usuário, e por fim o quarto parâmetro "teste@teste.com" é o e-mail do usuário.

updateUser: Atualiza informações de um usuário existente na API com base no ID do usuário, alterando nome, data de nascimento e gênero.  
A função pode ser utilizada da seguinte forma: updateUser(3, "Luiza", "2001-12-25", "F", "update@teste.com"); Aonde updateUser() é invocação da função, o primeiro parâmetro é o ID já existente do User, o segugundo parâmetro " Luiza " é o nome do usuário, terceiro parâmetro "2001-12-25" é a data de nascimento, quarto parâmetro “F” é o Gênero do usuário, e por fim o quinto parâmetro "update@teste.com" é o e-mail do usuário.

deleteUser: Remove um usuário da API com base no ID do usuário.  
A função pode ser utilizada da seguinte forma: deleteUser(2) ; Aonde deleteUser () é a invocação da função, e 2 é o id do User que vai ser detelada.

# 3 - Telas:

A estrutura de telas no diretório src/components/ é uma prática comum no desenvolvimento React, proporcionando uma organização eficiente do código-fonte. Este diretório abriga componentes reutilizáveis, incluindo as diversas telas que compõem a interface do usuário. A abordagem modular facilita a manutenção, escalabilidade e colaboração no desenvolvimento de aplicações React.

A tela AudioResultados é um exemplo genérico de componente localizado em src/components/. Este componente específico lida com a reprodução de áudio baseado em uma lista de músicas fornecida. A estrutura básica do componente é a seguinte:

Texto

Descrição gerada automaticamente

Figura – Exemplo genérico: AudioResultados.jsx

A transição entre telas é geralmente gerenciada pelo sistema de navegação da aplicação, que pode ser implementado usando bibliotecas populares como React Router. Quando um usuário interage com a interface, o sistema de navegação atualiza o componente renderizado, exibindo a tela correspondente. A organização modular das telas em src/components/ facilita a navegação e a manutenção, permitindo que desenvolvedores localizem facilmente os componentes relacionados a uma determinada tela.

Observações:

O exemplo AudioResultados ilustra a modularidade dos componentes de tela e como eles podem ser integrados em uma estrutura React mais ampla.

O controle de reprodução de áudio é um exemplo de lógica específica da tela, demonstrando como cada tela pode encapsular seu próprio comportamento.

# 4 – Postman e Collections

O Postman é uma ferramenta popular de desenvolvimento e teste de API que simplifica e agiliza o processo de construção, teste e documentação de APIs. Ele oferece uma interface gráfica amigável que permite aos desenvolvedores enviar solicitações HTTP para uma API, visualizar as respostas correspondentes e realizar testes automatizados. Uma característica chave do Postman é a capacidade de organizar solicitações relacionadas em coleções, proporcionando uma maneira estruturada de gerenciar e compartilhar conjuntos de solicitações.

Uma "collection" no Postman é um conjunto organizado de solicitações HTTP, juntamente com as configurações e informações necessárias para executar essas solicitações. As coleções permitem agrupar endpoints relacionados, simplificando a execução de testes e fornecendo uma forma eficiente de compartilhar o conjunto de solicitações com outros membros da equipe ou a comunidade em geral.

Importar e usar uma coleção no Postman é um processo direto. Primeiro, você pode importar uma coleção existente no formato JSON diretamente para o Postman. Essa funcionalidade é especialmente útil quando você recebe uma coleção de um colega de equipe ou encontra uma coleção online que deseja testar. Após importar a coleção, você terá acesso a todas as solicitações nela contidas, prontas para serem executadas.

Para usar uma coleção, basta selecionar a solicitação desejada dentro do Postman e fornecer quaisquer parâmetros necessários, como cabeçalhos, parâmetros de consulta ou corpo da solicitação. O Postman oferece uma ampla gama de recursos para visualizar as respostas das solicitações, monitorar o desempenho da API e até mesmo criar testes automatizados para garantir a integridade da API ao longo do tempo.

Em resumo, o Postman simplifica o processo de teste e interação com APIs, e as coleções fornecem uma maneira organizada e eficiente de gerenciar conjuntos de solicitações relacionadas, tornando o desenvolvimento e teste de APIs mais eficazes e colaborativos.