EletroNet

Descrição Técnica

Front-End:

O projeto Front-End está separado por pastas e arquivos, com as tecnologias HTML5, CSS3, JavaScript e com alguns frameworks, dentre eles, o React. Vamos explicar cada uma delas.

Pasta “Public”.

A pasta “public” em uma aplicação web utilizando o framework Express geralmente é utilizada para servir arquivos estáticos. Esses arquivos estáticos podem incluir imagens, estilos CSS, scripts JavaScript, fontes e outros ativos que o navegador do cliente pode requisitar diretamente.

A pasta “public” é uma parte essencial da arquitetura de uma aplicação web com Express, fornecendo uma maneira simples e eficiente de servir arquivos estáticos diretamente aos clientes. Configurada corretamente, ela pode melhorar significativamente a performance e a organização da aplicação.

Dentro da pasta public, temos o arquivo index.html

Index.html: Esse código HTML é um modelo típico usado em aplicações web desenvolvidas com React, especialmente criadas com o create-react-app. Vamos detalhar cada parte:

1. **Estrutura Básica do HTML**:

    - <!DOCTYPE html>: Define o tipo de documento como HTML5.
    - <html lang="en">: Define a linguagem do documento como inglês.
    - <head>: Contém metadados e recursos importantes para o documento.

2. **Metadados e Recursos no <head>**:

    - <meta charset="utf-8" />: Especifica a codificação de caracteres como UTF-8.
    - <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />: Define o ícone (favicon) do site, referenciando %PUBLIC_URL%, que é substituído pelo caminho correto durante o build.
    - <meta name="viewport" content="width=device-width, initial-scale=1" />: Configura a escala inicial e largura da viewport para dispositivos móveis.
    - <meta name="theme-color" content="#000000" />: Define a cor do tema do navegador para a aplicação.
    - <meta name="description" content="Web site created using create-react-app" />: Descreve o site para mecanismos de busca.

3. **Estilos e Fontes**:

    - <link rel="stylesheet" type="text/css" charset="UTF-8" href="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.6.0/slick.min.css" />: Importa um arquivo CSS de um CDN para estilização, neste caso, para o carrossel Slick.
    - <link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/slick-carousel/1.6.0/slick-theme.min.css" />: Importa o tema do carrossel Slick.
    - <link href="https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&display=swap" rel="stylesheet">: Importa a fonte "Poppins" do Google Fonts para uso no site.

4. **Outros Elementos no <head>**:

    - Comentários explicativos sobre manifest.json, %PUBLIC_URL%, e uso correto de caminhos relativos para recursos estáticos.

5. **Corpo do Documento (<body>)**:

    - <noscript>: Uma mensagem exibida se o JavaScript estiver desativado no navegador do usuário.
    - : O ponto de montagem principal para a aplicação React. Aqui é onde o conteúdo da aplicação será renderizado.

6. **Comentários Finais**:

    - Comentários adicionais fornecem instruções sobre como começar o desenvolvimento (npm start ou yarn start) e como criar uma versão de produção (npm run build ou yarn build).

Essencialmente, este arquivo HTML serve como o ponto de entrada para uma aplicação React, definindo metadados, importando estilos e fontes, e configurando elementos básicos para o funcionamento da aplicação no navegador.

Pasta "Src"

Dentro da pasta "Src" (Source) temos muitos componentes, Exemplo: Assets, Componentes, Css e Pages.

1 – Pasta "Assets":

A pasta "Assets" contém recursos estáticos que são utilizados na aplicação. Estes recursos podem incluir imagens, ícones, fontes, arquivos de áudio, vídeos, e outros tipos de mídia que não mudam frequentemente.

Uso Comum:

- **Imagens**: Logotipos, fotos de produtos, banners etc.
- **Ícones**: Arquivos SVG, favicons.
- **Fontes**: Arquivos de fontes personalizadas.

Nesse Projeto, estamos usando apenas imagens.

2 – Pasta "Components":

A pasta "Components" contém componentes reutilizáveis da interface do usuário (UI) que podem ser utilizados em várias partes da aplicação. Estes componentes são geralmente pequenos, modulares e focados em uma funcionalidade específica.

Usado no projeto são os arquivos (em React):

- Ambiente.jsx,
- App.jsx,
- BioContent.jsx,
- EditProdutoModal.jsx,
- Footer.jsx
- Form.jsx,
- Header.jsx,
- ImageText.jsx,
- ProdutoCard.jsx,

- ProdutoForm.jsx,
- ProdutoList.jsx,
- Welcome.jsx

3 – Pasta “Css”:

A pasta “Css” contém arquivos de estilos em CSS (Cascading Style Sheets) que são aplicados globalmente ou a partes específicas da aplicação. Dependendo da organização do projeto, esta pasta pode conter arquivos CSS, SCSS (Sass), ou outros pré-processadores de CSS.

Usado no projeto são os arquivos:

- Ambiente.css,
- Footer.css,
- Form.css,
- Header.css,
- Home.css
- Home.jsx,
- ImageText.css,
- ProductCard.css,
- ProdutoListStyle.jsx
- Welcome.css

4 – Pasta” Pages”:

A pasta Pages contém componentes que representam páginas inteiras da aplicação. Cada arquivo ou subpasta dentro de Pages corresponde a uma rota específica na aplicação.

Usado no projeto são os arquivos (em React):

- Bio.jsx,
- Cadastro.jsx,
- Home.jsx
- Produto.jsx

A organização do projeto com pastas “assets”, “components”, “css”, e “pages” promove uma estrutura modular, fácil de manter e escalável. Cada pasta tem um papel específico, ajudando a separar as preocupações e facilitando o desenvolvimento colaborativo. Esta estrutura é particularmente útil em projetos frontend modernos, onde a modularidade e a reutilização de componentes são cruciais para a eficiência e manutenção a longo prazo.

Bem, já falamos das pastas que estão dentro de “src” e dos arquivos que contêm dentro dessas pastas, agora vamos para aqueles arquivos separados dessas pastas, mas ainda sim em, dentro da pasta “src”, veja abaixo:

App.js

O código apresentado é um exemplo de uma aplicação React que utiliza a biblioteca “react-router-dom” para gerenciar a navegação entre diferentes páginas da aplicação.

Explicação dos Componentes e Roteamento:

BrowserRouter:

- É o roteador principal que usa a API de histórico do HTML5 para manter a interface da URL em sincronia com a aplicação. O BrowserRouter deve envolver o componente raiz da aplicação para que todas as rotas funcionem corretamente.

Routes:

- Contém todas as definições de rotas. Ele é um substituto para o antigo componente Switch do react-router-dom v5, fornecendo uma maneira de renderizar exclusivamente a primeira Route ou Redirect que corresponda à localização atual.

Route:

- Cada Route define uma associação entre um caminho de URL (path) e o componente que deve ser renderizado quando a URL corresponde a esse caminho.
- Parâmetros de rota: A rota /bio/:id demonstra o uso de parâmetros dinâmicos, permitindo que diferentes instâncias do componente Bio sejam renderizadas com base no valor do parâmetro id na URL.

Este código configura o roteamento básico para uma aplicação React utilizando react-router-dom. Ele define como diferentes URLs correspondem a diferentes componentes de página, e também demonstra como lidar com parâmetros dinâmicos nas rotas. Isso permite uma navegação fluida e modular dentro da aplicação, facilitando a manutenção e a expansão futura.

App.test.js

O App.test.js é um teste unitário para uma aplicação React utilizando a biblioteca “@testing-library/react”. Este teste verifica se um elemento específico (um link com o texto "learn react") está presente no documento quando o componente App é renderizado.

Este teste simples, mas poderoso verifica se a aplicação React está renderizando corretamente um link com o texto "learn react". Utilizando @testing-library/react, ele promove boas práticas de teste focadas na perspectiva do usuário final, garantindo que componentes chave estejam presentes e acessíveis no DOM. Este tipo de teste ajuda a manter a qualidade e a funcionalidade da aplicação à medida que ela evolui.

index.js

Inicia uma aplicação React, renderizando o componente principal App dentro de um elemento HTML no DOM. Este código utiliza as bibliotecas react e react-dom/client, que são fundamentais para a criação e renderização de componentes React.

Este código é a base para iniciar uma aplicação React, configurando a renderização do componente principal App dentro de um elemento DOM com o ID root. Ele utiliza as bibliotecas react e react-dom/client, e aplica React.StrictMode para promover boas práticas e detecção de problemas em tempo de desenvolvimento. Esta configuração é típica para iniciar a estrutura de uma aplicação React moderna.

reportWebVitals.js

É uma função chamada reportWebVitals, utilizada para medir e reportar métricas de desempenho importantes de uma aplicação web. Essas métricas são coletadas usando a biblioteca web-vitals, que facilita a medição de Core Web Vitals, como CLS (Cumulative Layout Shift), FID (First Input Delay), FCP (First Contentful Paint), LCP (Largest Contentful Paint) e TTFB (Time to First Byte).

A função reportWebVitals é uma maneira eficiente de coletar e reportar métricas de desempenho cruciais para a experiência do usuário em uma aplicação web. Utilizando importação dinâmica para carregar a biblioteca web-vitals, ela garante que a aplicação inicialize rapidamente e só carregue a biblioteca de métricas quando necessário. Este mecanismo é essencial para monitorar e otimizar a performance da aplicação em ambientes de produção.

Back-End:

No nosso Back-End, utilizamos as tecnologias JavaScript e Node.js para desenvolver toda a estrutura. Com elas, implementamos nossas APIs, gerenciamos o banco de dados e garantimos a escalabilidade e eficiência do sistema.

Dá mesma forma que foi explicado em Front-End, vamos explicar também as pastas e os arquivos do nosso Back-End.

Pasta “Controllers”

A pasta controllers em um projeto Node.js, especialmente em aplicações que seguem o padrão MVC (Model-View-Controller), é utilizada para armazenar os controladores. Os controladores contêm a lógica de controle da aplicação, manipulando as requisições recebidas, interagindo com os modelos e retornando as respostas apropriadas. Dentro da pasta Controller, temos o arquivo produtosController.js.

produtosController.js:

Dentro desse arquivo, o código define um conjunto de manipuladores de requisição (controllers) para a entidade "Produto" em um sistema de gerenciamento de produtos. Ele utiliza uma conexão com o banco de dados (db) e interage com a tabela produtos. Cada manipulador é uma função exportada que lida com operações CRUD (Create, Read, Update, Delete).

Este código define as operações CRUD para a entidade "Produto", manipulando requisições HTTP para criar, recuperar, atualizar e deletar produtos na base de dados. As funções fazem uso de consultas SQL para interagir com a tabela produtos e utilizam async/await para tratar operações assíncronas. As respostas apropriadas são enviadas ao cliente com códigos de status HTTP adequados e mensagens relevantes.

Pasta database

Uma pasta chamada database com um arquivo JSON dentro dela pode ser utilizada para diversas finalidades em um projeto de software. Vamos explorar alguns cenários comuns onde isso é aplicável.

Arquivo JSON: Se o arquivo JSON dentro da pasta database contém dados que a aplicação precisa acessar, modificar ou consultar, ele funciona como um banco de

dados simples. Isso é especialmente útil para aplicações pequenas ou protótipos, onde não há necessidade de um banco de dados completo

Em resumo, a pasta database com um arquivo JSON dentro dela é uma abordagem válida para armazenar dados simples ou configurações em projetos menores ou onde a complexidade de um banco de dados completo não é justificada.

Pasta models

A pasta models é comumente usada em projetos de software para organizar e armazenar definições de modelos de dados ou estruturas de objetos que representam entidades específicas dentro da aplicação.

s arquivos geralmente contêm classes ou definições que representam os objetos principais da aplicação, como usuários, produtos, pedidos etc.

A pasta models serve como um local centralizado para definir e armazenar modelos de dados ou estruturas de objetos na aplicação, promovendo uma organização eficiente do código e facilitando a manutenção e a escalabilidade do projeto de software.

Produto.js: A classe Produto é definida usando a palavra-chave class. Dentro dela, há um construtor (constructor) que é um método especial usado para inicializar objetos criados a partir dessa classe.

O construtor aceita cinco parâmetros:

- id: Identificador único do produto.
- nome: Nome do produto.
- descricao: Descrição detalhada do produto.
- preco: Preço do produto.
- imagemName: Nome do arquivo de imagem associado ao produto.

Dentro do construtor, cada um desses parâmetros é atribuído às propriedades correspondentes da instância da classe usando this.<propriedade> = <valor>.

No final do código, module.exports = Produto; é usado para exportar a classe Produto. Essa sintaxe é típica em ambientes Node.js para tornar a classe disponível para ser importada e usada em outros arquivos JavaScript no mesmo projeto.

o código define uma classe Produto básica em JavaScript, que encapsula as propriedades essenciais de um produto e a exporta para ser utilizada em outros módulos ou arquivos da aplicação.

Pasta Routes:

A pasta routes em um projeto Node.js é usada para organizar e definir as rotas da aplicação. As rotas são responsáveis por determinar como a aplicação responde a várias requisições de endpoints específicos e métodos HTTP (GET, POST, PUT, DELETE etc.). Dentro da pasta Routes, temos o arquivo produtosRoutes.js.

produtosRoutes.js:

Routes é um módulo de roteamento para uma aplicação Node.js utilizando o framework Express. Ele define rotas HTTP para gerenciar recursos de produtos em uma API.

Este código configura um conjunto de rotas para gerenciar operações CRUD (Create, Read, Update, Delete) de produtos em uma aplicação web. Ele utiliza o Express para definir as rotas e o Multer para lidar com uploads de arquivos, delegando a lógica de

negócio para um controlador específico (produtosController). Cada rota mapeia uma operação específica de acordo com o método HTTP e o caminho da URL.

Explicado todos as pastas e arquivos que continham dentro dos mesmos, vamos para arquivos fora dessas pastas, começando por db.js.

app.js

Esse código configura um servidor web utilizando Node.js com o framework Express:

Importação de Módulos

- express: Framework web para Node.js que simplifica a criação de APIs e aplicações web.
- multer: Middleware para o manuseio de uploads de arquivos.
- path: Módulo integrado do Node.js para lidar com caminhos de arquivos e diretórios.

Configuração do Multer

- storage: Utiliza multer.memoryStorage() para armazenar temporariamente os uploads em memória.
- upload: Configura o middleware multer com o objeto storage, definindo como os arquivos serão tratados durante o upload.

Configuração do Express

- app.use(express.json()): Middleware do Express para interpretar o corpo das requisições como JSON.
- Configuração de CORS (Access-Control-Allow-* headers): Permite requisições de diferentes origens (*), especifica métodos permitidos (GET, POST, PUT, DELETE, OPTIONS) e cabeçalhos aceitos (Origin, X-Requested-With, Content-Type, Accept).

Rotas

- app.use('/produtos', produtoRoute): Define que as requisições para /produtos serão tratadas pelo roteador produtoRoute, que geralmente está definido em um arquivo separado (produtoRoute.js).

Servindo Arquivos Estáticos

- app.use('/uploads', express.static(path.join(__dirname, 'uploads'))): Define que os arquivos estáticos na pasta uploads serão servidos na rota /uploads. Isso permite que os arquivos enviados através do multer sejam acessados diretamente pelo navegador ou outras partes da aplicação.

Inicialização do Servidor

- app.listen(5000, () => { ... }): Inicia o servidor Express na porta 5000. Quando o servidor é iniciado com sucesso, exibe uma mensagem indicando que está pronto para receber requisições.

Este código configura um servidor web utilizando Express, define rotas para manipular produtos (/produtos), permite uploads de arquivos utilizando multer, configura CORS para permitir comunicação com diferentes origens e serve arquivos estáticos da pasta uploads. É um exemplo básico de como configurar um servidor web robusto e seguro em Node.js para uma aplicação backend.