Projeto Studio-sol-challenge

Objetivo:

Projeto desenvolvido para a concorrência ao cargo de desenvolvedor front-end na Studio-Sol, com o objetivo de mostrar através deste minhas habilidades na área.

Estrutura:

O projeto está dividido e estruturado da seguinte maneira:

- ➢ pastas:

components: pasta para armazenar todos os componentes do sistema, ou seja, todos os blocos de código de renderização que podem ser reaproveitados onde for preciso.

Nesta pasta existe uma pasta para cada componente em específico contendo na maioria dos casos 3 arquivos, conforme as necessidades.

- index: para a renderização do componente em si;
- styles: para estilos CSS utilizando a biblioteca styled-components;
- types: para as interfaces de Typescript.

Dentro dessa pasta existe um arquivo index utilizado para uma importação mais enxuta no sistema, conseguindo importar todos os componentes através de um único arquivo, evitando assim importar os componentes em várias linhas.

services: pasta que armazena o arquivo contendo o endpoint para a chamada das rotas necessárias ao funcionamento do sistema, utilizando a biblioteca de requisições axios.

styles: configuração de um estilo global para a aplicação, como estilos CSS globais e definição de cores padrões, onde serão e poderão ser utilizados em qualquer componente do sistema.

utils: lugar onde são armazenadas as funções que executam determinada tarefa em específico e que podem ser reutilizadas, como por exemplo, função que executa a lógica de exibição dos leds.

redux: é onde se encontra o ‘cérebro’ da aplicação, o estado global. Contendo informações que são cruciais ao funcionamento do sistema, disponibilizadas em todos os componentes que estão dentro do contexto provido da store do projeto. Possui duas pastas:

fetchActions: responsável por fazer as requisições propriamente ditas às rotas e acionando os dispatchers que alteram os valores das variáveis globais da aplicação;

store: armazena as variáveis globais da aplicação.

Arquivos importantes:

App.tsx: Componente que engloba todos os outros componentes da aplicação e aciona o estilo global da aplicação.

main.tsx: aciona o componente App.tsx e é envolto pelo contexto global da aplicação (o redux), fornecendo a 'loja' de dados (store).

Tecnologias utilizadas:

HTML: elementos visuais para a interação do sistema com o usuário;

CSS: escrita de estilos de formatação dos elementos html;

Javascript: linguagem de programação que permite toda a dinamicidade da interação do usuário com o sistema e toda a lógica para as devidas execuções;

ReactJS: biblioteca para auxílio na escrita de código javascript, onde é feita a junção de javascript com html de forma simples e bastante compreensível;

styled-components: permite escrever códigos dentro do javascript (CSS-in-JS), para agilizar e facilitar a criação de estilos CSS no projeto e deixar o mesmo mais organizado;

typescript: adicionando uma camada de segurança no processo de desenvolvimento do sistema, por possuir uma interface com atributos que são obrigatórios, ou opcionais, assim como também definição de valores padrões como resultado;

axios: utilizada para fazer requisições http com métodos específicos para cada ação, como por exemplo, funções get - buscar dados, post - inserir registros, etc;

react-icons: biblioteca para utilização e exibição de ícones em projetos react, que é um compilado de várias outras bibliotecas já consolidadas para web, para o consumo de ícones;

react-dom: biblioteca para manipulação do DOM, usada neste caso para vincular o componente App ao arquivo inicial em HTML.

redux: para armazenar o estado global da aplicação, variáveis que podem ser consumidas e alteradas em qualquer ponto do projeto, desde que este esteja envolvido dentro do contexto (Provider) na árvore de componentes do projeto;

toolkit: facilitador para redux, contendo uma sintaxe muito mais enxuta e compreensível das ações (actios), redutores (reducers), etc, fornecendo métodos que são úteis à manipulação dessas variáveis;

git: para trabalhar com sincronização do projeto local com um repositório remoto vinculado ao mesmo, hospedado no sistema GitHub, para poder acessá-lo de qualquer lugar do mundo , qualquer dispositivo e a qualquer hora.

Funcionamento do projeto:

O projeto está dividido em 4 áreas na página, sendo cada uma um componente construído e exibido em tela: Header, Message, Display e Footer.

header: cabeçalho contendo o título e um divider;

Message: exibe a mensagem ao usuário referentes ao palpite realizado e o número que foi buscado;

Display: área responsável por exibir os leds que são 7 segmentos posicionados de forma a formar números. Exibe um array de objetos, representando os números digitados pelo usuário ou o status code que é recebido da requisição à rota em casos de falhas; cada um com 7 propriedades string; indicando se o led está ligado ou não; e esses valores são utilizados para classes CSS para suas devidas formatações;

Footer: contém o campo de entrada e o botão de clique que dispara a função de verificar se o número buscado e o digitado são iguais ou não e também alterar os leds;

App.js: ao iniciar o sistema é disparada a primeira requisição que armazena no redux o número buscado através do hook do react, useEffect, passando um array vazio como segundo parâmetro do hook, indicando que a chamada será no início do acionamento do componente, e também consome os valores armazenados no redux para exibições e lógicas.

input: campo de entrada com o limite de até três caracteres numéricos.

Ao clicar no botão de enviar, a função handleGuessNumber() é chamada e faz uma validação, caso o valor do input esteja vazio a função retorna. Caso o input contenha um valor, as validações de comparação entre o valor digitado pelo usuário e o número obtido na requisição são feitas e o valor do input é alterado para seu estado inicial (string vazia 4). Em todos os casos, uma mensagem de informação é exibida e a função changedLeds() é chamada, recebendo um parâmetro: o número digitado pelo usuário e a depender da quantidade de algarismos digitados, a string será mapeada e retornará um array de objetos, representando esse número digitado ou o código de status de erro. Esse array será passado para o redux para modificar os leds na tela e posteriormente consumí-lo nas devidas telas que irão utilizá-los.

No início da função, o número digitado pelo usuário é passado para string e um array vazio é criado para receber os objetos referentes a cada número. Um aço de repetição percorre cada caractere do número digitado e chama a função handleLeds(), que faz a verificação do número. Quando o número é identificado, um push desse objeto ocorre no array criado inicialmente. Essa função retorna o array com as informações de cada caractere do número digitado pelo usuário ou do código de status.

Caso o jogador dê o palpite correto ou haja um erro na requisição, o botão de nova partida aparece e ao ser clicado, a função handleNewGame() é chamada, setando o número obtido pela requisição para vazio, setando a informação ao usuário para vazio, chamando a action do redux para modificar as variáveis para seus valores iniciais.

Coisas que gostaria de adicionar se tivesse mais tempo, como melhorias:

testes unitários;

hospedagem do sistema em um servidor em produção.

Para executar o projeto:

Para executar o projeto rode o script yarn ou npm dev, para executá-lo em localhost.