Fase 1 – Criação dos componentes

1 – Criar o servidor no terminal (npx create-react-app [diretório] e dar start no server (npm start)

2 – Baixar o zip dos arquivos do projeto e separar as imagens pra colocar na pasta public.

3 – Explorar o projeto dentro do VsCode, deletando tudo que não precisa dentro do App.js e App.css

4 – Analisar mais ou menos quantos componentes o projeto terá e criar uma pasta para coloca-los.

5 – O que for de input, criar uma pasta dentro da pasta componentes para colocar os campos de texto.

6 – Receita de bolo ao criar os componentes = index.js e arquivo.css.

7 – Dentro do index. js criar uma constante em arrow function(ex. const CampoTexto = () =>{ return }, importar o arquivo.css e exportar a function (export default CampoTexto).

8 – Trabalhar os inputs dentro de alguma tag semântica (div,section, form, etc.) e atribuir um className para as tags.

9 – Utilizar o parâmetro props dentro das functions para modificar labels e parametros, posteriormente, quando for importa-las.

10 – Ao fazer uso de listas dentro de algum select, criar um componente separado dos inputs de texto e utilizar o props.itens.map com um item, entre parênteses, em arrow function, retornando o option ( {props.itens.map(item => <option>{item} <option>)} entre as tags <select> ).

11 – Atribuir uma Key na tag <option key={item}> {item} <option>

12 – Criar um componente de Formulário para agrupar todos os inputs e listas. Fazer o import dos CampoTextos e Listas, atribuir as devidas labels e itens da lista(atribuir uma constante com a lista dos times no começo da function Formulario() para ser atribuído os itens chamados no .map)

13 – Criar o botão para submeter o formulário <button>. Caso queira carregar uma imagem no lugar do texto do botão, usar a propriedade props.children. Ao importar o botão no formulário, poderá escrever entre as tags <button> como se fosse HTML normal.

Fase 2 – Organizando e scutando eventos do Formulário.

- Ajustando o submit do botão

14 – Dentro da própria tag <form> no componente Formulario, utilizar algum evento JS on.. como por exemplo onSubmit (<form onSubmit=()>)

15 – Criar uma função para ouvir o evento onSubmit e chama-lo, posteriormente, dentro da tag form( exemplo: a função criada const aoSalvar = () =>{}, será ouvida dentro da tag <form onSubmit=(aoSalvar) ).

16- Caso queira impedir que a página seja recarregada ao submeter o formulário pelo botão, atribuir o método evento a função declarada e, dentro da função, utilizar o parâmetro seguido de preventDefault(). Exemplo:

const aoSalvar = (evento) =>{

evento.preventDefault()

}.

17 – Para utilizar a validação do Html Required que torna o preenchimento de um campo input obrigatório, no componente Formulário, localize os inputs que você importou e deseja a validação, dentro da tag dele coloque obrigatorio={true}, exemplo <CampoTexto obrigatorio = {true}>. Em seguida, vá no componente CampoTexto, dentro da tag input coloque required = {props.obrigatorio}, exemplo <input required = {props.obrigatorio}.

- Capturando o valor digitado nos inputs

19 – No componente CampoTexto, onde ficam os inputs de texto, criar uma função para capturar cada letra(value) digitado.

20 – Dentro da função, utilizar o “evento” target do JS como parâmetro e chamar esse evento dentro da função com o “value” (evento.target.value) para salvar as informações digitadas.

21 – Ainda dentro desta função, elevar o estado chamando um parâmetro novo, via props, a ser definida no componente Formulário exemplo:

props.aoAlterado(evento.target.value)

explicação = aoAlterado será responsável por guardar os values digitados. No componente Formulario, existirá uma call-back do aoAlterado, definindo-o.

22 – O valor digitado no campo dos inputs será guardado e elevado, também, via props ao componente de Formulário, para ser definido por meio de um estado(useState):

<input value = {props.valor}>

23 – Colocar dentro da tag input do componente do CampoTexto um novo evento do JS chamado onChange, para ouvir a função de captura de valores digitado anteriormente criada: <input onChange = {nomedafunção}>

- Usando hooks de estados (useState) nas funções

24 – No componente Formulário, criar quantas constante forem necessárias, dependendo da quantidade de inputs existentes no formulário com uma variável e um seter para guardar os useState(‘ ‘) a serem setados, posteriormente.

25 – Depois de criar os estates e guardar em uma variável e seter, é hora de realizar as call-backs do ComponenteDeTexto, atribuindo o valor, declarado via props dentro da tag input ao state criado no Formulário e a função aoAlterado, também declarada via props dentro da função aoDigitado, onde sua única função, aqui, será setar a variável do state. Tudo isso no componente Formulário, dentro da tag <CampoTexto>, anteriormente importada.

Resumindo a função aoAlterado irá retornar um seter de estado: aoAlterado = (valor => setNome(valor)). Em seguida, atribuir um novo parâmetro com o state criado, para atribuir o valor capturado no aoAlterado. exemplo:

valor={nome}

Fim do formulário: A função do formulário não é armazenar listas de colaboradores, apenas de conter e passar adiante os dados.

Fase 3 – Trabalhando e guardando valores adicionados

 - Submeter os dados no submit do botão

26 –Dentro do App.js, criar um state de um array vazio, uma lista que salvará os dados passados adiante no formulário. Ex:

const [colaboradores, setColaboradores] = useState ([ ])

27 – Criar uma função (vazia, por enquanto) para adicionar novos itens ao state colaboradores:

const aoColaboradorAdicionado = () =>{

}

28 – Ainda no app.js, inserir, dentro da tag <Formulario>, uma propriedade para chamar função criada no passo anterior e passar adiante via props para o componente Formulário.

<Formulario aoColaboradorCadastrado = {aoColaboradorAdicionado}>

29 – Construir a função do passo 27, colocando o colaborador como parâmetro a ser recebido como argumento pelo seter, no escopo da função, setar os colaboradores antigos (... colaboradores) e adicionar o novo no final:

const aoColaboradorAdicionado = (colaborador) =>{

setColaboradores(... colaboradores, colaborador)

}

30 – No componente Formulário, colocar dentro da function de submit do botão, a call-back via props dos colaboradores adicionados e registra-los em forma de objeto:

props.aoColaboradorAdicionado = {nome, email, imagem, cargo}

- Adicionando, dinamicamente, card estilizado dos elementos da lista

31 – No caso do projeto, serão adicionados cards dos times. Criar um novo componente chamado Times e seguir a mesma receita de bolo com o index.js e time.css, tudo dentro de uma section.

31.1 – Criar uma tag h3 para receber o nome dos times via props.

<h3> {props.nome} <h3>

Explicação: esse titulo h3, será utilizado dentro da tag <Times

32 - Organizar o css do componente Time de acordo com o figma.

33 – Dentro do APP.js, criar uma constante em formato de objeto, contendo nome, cor de fundo da section e cor de fundo dos times, descritos no figma. Ex:

const times = [

{nome: ‘ ’, corPrimaria: ’ ’, corSecundaria: ‘ ‘},

{ nome: ‘ ’, corPrimaria: ’ ’, corSecundaria: ‘ ‘},

{ nome: ‘ ’, corPrimaria: ’ ’, corSecundaria: ‘ ‘},

{ nome: ‘ ’, corPrimaria: ’ ’, corSecundaria: ‘ ‘}

]

34 – Utilizar o .map para renderizar cada time de forma dinâmica via props dentro da tag <Times> no App.js. Ex:

{times.map(time => <Time nome={time.nome} />}

35 – Definir dentro da tag <Time> a key e a propriedade das cores de fundo estabelecidas no objeto.

{times.map(time => <Time nome={time.nome} key={time.nome}

corDeFundo={time.corPrimaria} corDeFundoCard={time.corSecundaria} />}

36 – No componente Time utilizar o style inline para estilizar o background do card dentro da tag semântica. Ex:

<section style = {{backgroudColor: props.corsecundaria}}>

Explicação = apesar de parecer CSS tradicional, a semântica é diferente, no caso, o background-color virou backgroundColor, ele é tido como objeto, por isso é escrito entre chaves, possibilitando receber via props, o objeto corSecundaria dos times. Outra forma de escrever seria declarando uma constante para receber o background e declarando dentro da tag style, ex:

const styles = {backgroundColor: props.corSecundaria}

<section style = {styles}>

36.1 – Fazer o mesmo com a borda dentro da tag h3, utilizando o borderColor.

- Refatorando códigos duplicados

37 – Excluir a lista de times do componente Formulário, pois o array de objetos criado no passo anterior já satisfaz, podendo ter seus elementos destacados via props e passado adiante, visto q a lista do Formulário é composta apenas por strings, nós poderemos extrair apenas o nome:” “do objeto.

37.1 – Através do. map, extrair do objeto apenas o parametro desejado, no caso, o nome, atribuindo uma prop dentro do import do componente Formulario no App.js. Ex:

<Formulario times=times.map(time=>time.nome)>

- Adicionando os cards visuais do projeto

38 – Criar um novo componente visual para receber os colaboradores estilizados.

39 – Estilizar o componente dos colaboradores de acordo com o figma.

3 – Adicionar uma nova propriedade para cada campo input no .map para que possa passar os colaboradores cadastrador via props