

Análise e Desenvolvimento de Sistemas

Frameworks Web I

Aula 04 - React JS: parte 3

Prof. João Paulo F. C. César
joao cesar@unilavras.edu.br

Me: Makes a small CSS change

My Site:



Estilização

Estilização inline



[Acesse o projeto](#)

Estilização

Quando se trata de estilizar componentes no React, uma das opções é usar estilos inline. Os estilos embutidos no React são escritos como objetos JavaScript, em vez de strings CSS.

- Para usar estilos inline no React, você define um objeto JavaScript onde as propriedades estão no formato camelCase. Em seguida, você atribui esse objeto ao atributo style do elemento JSX.

```
const style = {  
  color: 'blue',  
  fontSize: '20px',  
  textAlign: 'center'  
};  
  
const MyComponent = () => {  
  return <h1 style={style}>Hello World!</h1>;  
}  
  
export default MyComponent;
```



Estilização

No exemplo anterior, o objeto de estilo contém três propriedades: color, fontSize e textAlign, cada uma representando uma propriedade CSS.

- Essas propriedades são atribuídas a um h1 por meio do atributo style.
- Essa abordagem fornece uma maneira de alterar dinamicamente o estilo de um componente com base em seu estado ou props.



[Acesse o projeto](#)

Estilização

Você pode alterar a cor de um botão com base no fato de ele ter sido clicado:

```
import { useState } from "react";

const MyButton = () => {
  const [isClicked, setIsClicked] = useState(false);

  const handleClick = () => {
    setIsClicked(!isClicked);
  }

  const style = {
    backgroundColor: isClicked ? 'green' : 'blue',
    color: 'white',
    padding: '10px',
    border: 'none'
  };

  return (
    <button style={style} onClick={handleClick}>Clique aqui</button>
  )
}

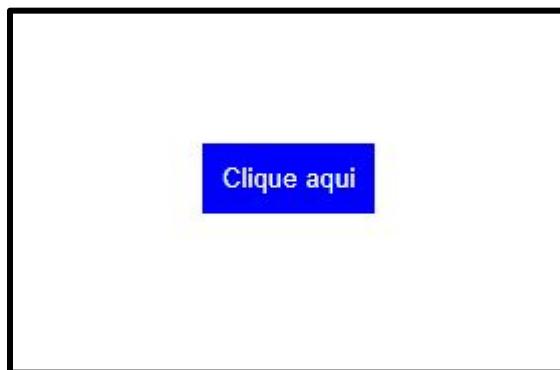
export default MyButton;
```

Clique aqui

Estilização

No exemplo anterior, no componente MyButton, o objeto de estilo inclui uma propriedade backgroundColor que muda com base no estado isClicked.

- Quando o botão é clicado, a função handleClick é acionada, atualizando o estado isClicked e fazendo com que o componente seja renderizado novamente com o novo estilo.



Estilização

No entanto, embora os estilos inline possam ser úteis para pequenas mudanças de estilo e estilo dinâmico, eles têm algumas limitações em comparação ao CSS.

- Por exemplo, eles não suportam pseudo-classes como :hover ou :active, ou queries media para design responsivo.
- Além disso, escrever grandes quantidades de CSS em JavaScript pode ser difícil de gerenciar.

Módulos CSS

Estilização

Módulos CSS são um método popular para estilizar aplicativos React. Eles ajudam a enfrentar alguns dos desafios inerentes ao CSS, como escopo global e colisões de nomes de classes, gerando automaticamente nomes de classes exclusivos.

- Com os Módulos CSS, cada classe CSS tem escopo local por padrão.
- O que significa que você pode ter a mesma classe CSS em arquivos diferentes sem se preocupar em nomear conflitos. Cada nome de classe CSS é transformado em uma variável local exclusiva quando importado para um arquivo JavaScript.

Estilização

Vamos ver como isso funciona com um exemplo. Primeiro, criaremos um arquivo de Módulo CSS com a extensão .module.css. Vamos chamá-lo de Button.module.css:

```
.myButton {  
    background-color: blue;  
    color: white;  
    padding: 10px;  
    border: none;  
    cursor: pointer;  
}
```



[Acesse o projeto](#)

Estilização

Agora podemos importar este módulo CSS para nosso componente React assim:

```
import styles from './Button.module.css'

const MyButton = () => {

    return (
        <button className={styles.myButton}>
            Clique aqui
        </button>
    )
}

export default MyButton;
```

Clique aqui

Estilização

Neste exemplo, a classe primária compõe a classe baseButton, herdando seus estilos.

Os Módulos CSS também suportam composição, um recurso que permite reutilizar classes CSS em diferentes módulos.

- Por exemplo, se você tiver um estilo de botão base e quiser criar uma variante para um botão principal, você pode fazer isso com composição.

```
.baseButton {  
    padding: 10px;  
    border: none;  
    cursor: pointer;  
}  
  
.primary {  
    composes: baseButton;  
    background-color: blue;  
    color:white;  
}
```

```
import styles from './Button.module.css'  
  
const MyButton = () => {  
  
    return (  
        <button className={styles.primary}>  
            Clique aqui  
        </button>  
    )  
}  
  
export default MyButton;
```

Clique aqui

Styled Components

Estilização

Styled Components é uma biblioteca para estilização em aplicativos React que aproveita o poder das Tagged Template Literals em JS e a flexibilidade do CSS.

- Usando componentes estilizados, você pode escrever código CSS dentro do JavaScript, mantendo os estilos vinculados aos componentes onde são usados.
- Isso elimina o risco de colisões de nomes de classes e garante que os estilos sejam isolados em seus respectivos componentes.

Para instalar a biblioteca, execute no terminal

```
npm install styled-components
```



[Acesse o projeto](#)

Estilização

Em **Styled Components**, cada elemento estilizado é um componente React. Aqui está um exemplo de como criar um botão estilizado usando esta biblioteca:

```
1 import styled from "styled-components";
2
3 const StyledButton = styled.button`
4   background-color: blue;
5   color: white;
6   padding: 10px;
7   border: none;
8   cursor: pointer;
9
10  &:hover {
11    background-color: red;
12  }
13`;
14
15 const MyComponent = () => {
16   return <StyledButton>Clique aqui</StyledButton>;
17 };
18
19 export default MyComponent;
```

Clique aqui

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import MyComponent from './MyComponent';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <MyComponent />
);
```

Estilização

No exemplo anterior, `StyledButton` é um novo componente que definimos usando a função `styled.button`.

- Esta função usa um modelo literal contendo nossas regras CSS.
- Em seguida, usamos esse componente `StyledButton` como faríamos com qualquer outro componente React.

Estilização

O Style Components oferece suporte a temas, que é um recurso importante para muitos aplicativos. A criação de temas é feita por meio do componente ThemeProvider.

Aqui está um exemplo:



[Acesse o projeto](#)

Clique aqui

```
import styled, { ThemeProvider } from 'styled-components';

const theme = {
  primaryColor: 'blue',
  secondaryColor: 'darkblue'
};

const StyledButton = styled.button`
  background-color: ${props => props.theme.primaryColor};
  color: white;
  padding: 10px;
  border: none;
  cursor: pointer;

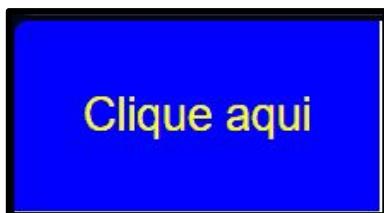
  &:hover {
    background-color: ${props => props.theme.secondaryColor};
  }
`;

const MyComponent = () => {
  return (
    <ThemeProvider theme={theme}>
      <StyledButton>Clique aqui</StyledButton>
    </ThemeProvider>
  )
}

export default MyComponent;
```

Estilização

Você também pode passar props para um componente estilizado:



[Acesse o projeto](#)

```
import styled, { ThemeProvider } from 'styled-components';

const theme = {
  primaryColor: 'blue',
  secondaryColor: 'darkblue'
};

const StyledButton = styled.button`
  background-color: ${props => props.theme.primaryColor};
  color: ${props => props.color || 'white'};
  padding: ${props => props.padding || '10px'};
  border: none;
  cursor: pointer;

  &:hover {
    background-color: ${props => props.theme.secondaryColor};
  }
`;

const MyComponent = () => {
  return (
    <ThemeProvider theme={theme}>
      <StyledButton color="yellow" padding="20px">
        Clique aqui
      </StyledButton>
    </ThemeProvider>
  )
}

export default MyComponent;
```

Estilização

No exemplo anterior, o componente `StyledButton` agora aceita `color` e `padding` como props, e esses props são usados para personalizar o CSS do componente.

- Se os props não forem fornecidos, ele retornará aos valores padrão ('branco' para `color` e '10px' para `padding`).
- O `ThemeProvider` disponibiliza o tema para todos os componentes, e podemos acessá-lo via `props.theme`.
- Ao renderizar `StyledButton` dentro de `MyComponent`, passamos `color` e `padding` como props. O botão renderizado agora terá uma cor de texto amarela e um padding de 20px.

Estilização

Uma desvantagem dos Styled Components é a sobrecarga de ter CSS em JS.

- Isso poderia aumentar potencialmente o tamanho do pacote do seu aplicativo, embora na maioria dos casos o aumento seja insignificante.
- Em resumo, Styled Components é uma biblioteca poderosa e flexível para estilização em aplicativos React. Ele aproveita recursos modernos de JavaScript para permitir que você escreva CSS em seu JavaScript.

Material UI



Estilização

Material UI é um framework React que fornece um conjunto de componentes React pré-construídos e reutilizáveis.

- Com o Material UI, você pode criar interfaces de usuário bonitas, consistentes e funcionais com menos esforço e codificação.
- O Material UI foi desenvolvido com os princípios do Material Design do Google, com o objetivo de oferecer uma experiência de usuário unificada e simplificada em diferentes plataformas e dispositivos.

Para instalar a biblioteca, execute no terminal

```
npm install @mui/material @emotion/react @emotion/styled
```

Estilização



[Acesse o projeto](#)

Depois de instalado, você pode importar os componentes que deseja usar nos componentes do React:

```
import Button from '@mui/material/Button';

const MyButton = () => {
    return (
        <Button color="primary">
            Clique aqui
        </Button>
    )
}

export default MyButton;
```

Estilização



[Acesse o projeto](#)

Os componentes da Material UI são altamente personalizáveis. Você pode ajustar sua aparência para se adequar ao tema do seu aplicativo usando o componente `ThemeProvider` integrado, que permite especificar um tema personalizado:

```
import { ThemeProvider, createTheme } from '@mui/material/styles';
import MyButton from './MyButton';

const theme = createTheme({
  palette: {
    primary: {
      main: '#ff4400',
    },
    secondary: {
      main: '#f44336',
    },
  },
});

const MyComponent = () => {
  return (
    <ThemeProvider theme={theme}>
      <MyButton />
    </ThemeProvider>
  )
}

export default MyComponent;
```

Estilização



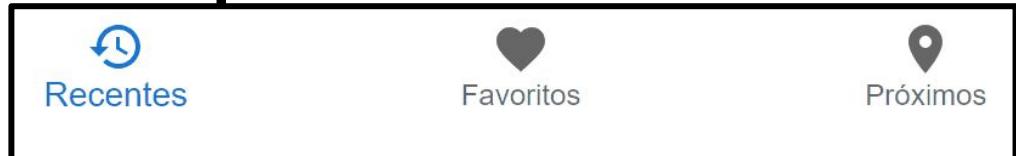
[Acesse o projeto](#)

Mais exemplos

```
1 import * as React from "react";
2 import Box from "@mui/material/Box";
3 import BottomNavigation from "@mui/material/BottomNavigation";
4 import BottomNavigationAction from "@mui/material/BottomNavigationAction";
5 import RestoreIcon from "@mui/icons-material/Restore";
6 import FavoriteIcon from "@mui/icons-material/Favorite";
7 import LocationOnIcon from "@mui/icons-material/LocationOn";
8
9 const MyComponent = () => {
10   const [value, setValue] = React.useState(0);
11
12   return (
13     <Box sx={{ width: 500 }}>
14       <BottomNavigation
15         showLabels
16         value={value}
17         onChange={({event, newValue}) => {
18           setValue(newValue);
19         }}
20       >
21         <BottomNavigationAction label="Recentes" icon={<RestoreIcon />} />
22         <BottomNavigationAction label="Favoritos" icon={<FavoriteIcon />} />
23         <BottomNavigationAction label="Próximos" icon={<LocationOnIcon />} />
24       </BottomNavigation>
25     </Box>
26   );
27 };
28
29 export default MyComponent;
```

Antes, instale:

`npm install @mui/icons-material`



Estilização

Mais exemplos



Tilápis

Tilápis é o nome comum dado a várias espécies de peixes ciclídeos de água doce pertencentes à subfamília Pseudocrocidolita e em particular ao gênero Tilápis.

[COMPARTELHAR](#) [LEIA MAIS](#)



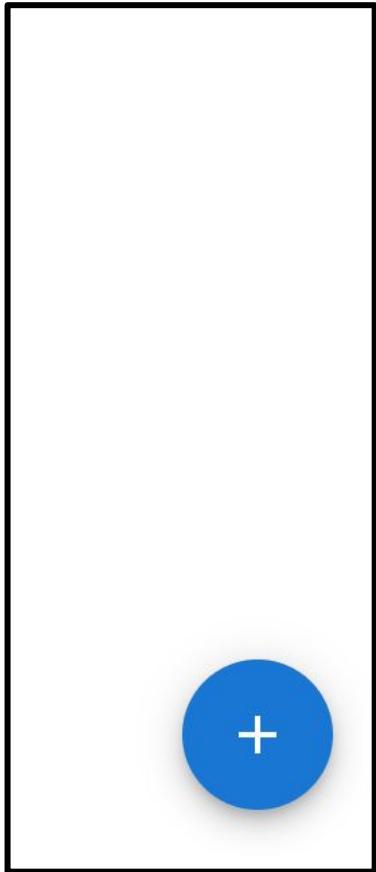
[Acesse o projeto](#)

```
import * as React from 'react';
import Card from '@mui/material/Card';
import CardActions from '@mui/material/CardActions';
importCardContent from '@mui/material/CardContent';
import CardMedia from '@mui/material/CardMedia';
import Button from '@mui/material/Button';
import Typography from '@mui/material/Typography';

export default function MediaCard() {
  return (
    <Card sx={{ maxWidth: 345 }}>
      <CardMedia
        sx={{ height: 140 }}
        image="https://tinyurl.com/4a6zkxuz"
        title="tilapia"
      />
      <CardContent>
        <Typography gutterBottom variant="h5" component="div">
          Tilápis
        </Typography>
        <Typography variant="body2" color="text.secondary">
          Tilápis é o nome comum dado a várias espécies de peixes
          ciclídeos de água doce pertencentes à subfamília
          Pseudocrocidolita e em particular ao gênero Tilápis.
        </Typography>
      </CardContent>
      <CardActions>
        <Button size="small">Compartilhar</Button>
        <Button size="small">Leia Mais</Button>
      </CardActions>
    </Card>
  );
}
```

Estilização

Mais exemplos



[Acesse o projeto](#)

```
import * as React from 'react';
import Box from '@mui/material/Box';
import SpeedDial from '@mui/material/SpeedDial';
import SpeedDialIcon from '@mui/material/SpeedDialIcon';
import SpeedDialAction from '@mui/material/SpeedDialAction';
import FileCopyIcon from '@mui/icons-material/FileCopyOutlined';
import SaveIcon from '@mui/icons-material/Save';
import PrintIcon from '@mui/icons-material/Print';
import ShareIcon from '@mui/icons-material/Share';

const actions = [
  { icon: <FileCopyIcon />, name: 'Copiar' },
  { icon: <SaveIcon />, name: 'Salvar' },
  { icon: <PrintIcon />, name: 'Imprimir' },
  { icon: <ShareIcon />, name: 'Compartilhar' },
];

export default function BasicSpeedDial() {
  return (
    <Box sx={{ height: 320, transform: 'translateZ(0px)', flexGrow: 1 }}>
      <SpeedDial
        ariaLabel="SpeedDial basic example"
        sx={{ position: 'absolute', bottom: 16, right: 16 }}
        icon=<SpeedDialIcon />
      >
        {actions.map((action) => (
          <SpeedDialAction
            key={action.name}
            icon={action.icon}
            tooltipTitle={action.name}
          />
        ))}
      </SpeedDial>
    </Box>
  );
}
```

React Hooks

useState

React Hooks



[Acesse o projeto](#)

O hook useState permite que componentes funcionais tenham estado local. Ele fornece a funcionalidade para ler uma variável de estado e atualizá-la.

- useState é uma função que aceita um argumento, que é o estado inicial, e retorna um array de dois elementos: o estado atual e uma função para atualizar o estado. Aqui está um exemplo básico de uso:

```
import React, { useState } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        clique aqui
      </button>
    </div>
  )
}

export default Counter;
```

Você clicou 0 vezes

Clique aqui

React Hooks

Neste exemplo, useState é chamado com o estado inicial 0. Ele retorna um array, e usamos a desestruturação do array para atribuir o estado atual e a função atualizadora de estado às variáveis count e setCount, respectivamente.

- A função setCount é usada para atualizar o estado. Quando você chama esta função, o React renderiza novamente o componente com o novo estado.
- No componente Contador, é chamado setCount com o novo valor de contagem quando o botão for clicado.



[Acesse o projeto](#)

React Hooks

Uma das vantagens de useState é que ele permite ter múltiplas variáveis de estado em um único componente, cada uma com sua própria função de atualização.

```
import React, { useState } from 'react';

const Form = () => {
    const [name, setName] = useState('');
    const [email, setEmail] = useState('');

    const handleNameChange = event => setName(event.target.value);
    const handleEmailChange = event => setEmail(event.target.value);

    return (
        <form>
            <input type='text' value={name} onChange={handleNameChange} />
            <input type='email' value={email} onChange={handleEmailChange} />
            <button type='submit'>Enviar</button>
        </form>
    )
}

export default Form;
```

useEffect

React Hooks

O hook permite realizar efeitos colaterais - operações que não cabem na fase de renderização de um componente React, como busca de dados, assinaturas ou alteração manual do DOM.

React Hooks



[Acesse o projeto](#)



```
import React from 'react';
import ReactDOM from 'react-dom/client';
import FetchPokemon from './FetchPokemon';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <FetchPokemon pokemonId={1}>
);


```

```
import React, { useEffect, useState } from 'react';

function FetchPokemon({ pokemonId }) {
  const [pokemon, setPokemon] = useState([]);

  useEffect(() => {
    async function loadPokemon() {
      const resposta = await fetch(
        `https://pokeapi.co/api/v2/pokemon/${pokemonId}`
      );
      const pokemon = await resposta.json();
      console.log(pokemon);
      setPokemon(pokemon);
    }
    loadPokemon();
  }, [pokemonId]);

  return (
    <>
      <ul>
        {pokemon ? (
          <div>
            <h1>{pokemon.name}</h1>
            </div>
        ) : (
          'Carregando...'
        )}
      </ul>
    </>
  );
}

export default FetchPokemon;
```

React Hooks

No exemplo anterior, usamos `useEffect` para buscar dados de uma API quando o componente é montado e sempre que a propriedade `pokemonId` muda. O efeito será executado após a primeira renderização e após cada atualização (quando o `pokemonId` for alterado).

- O segundo parâmetro para `useEffect` é a matriz de dependência. Essa matriz deve conter quaisquer variáveis das quais seu efeito dependa.
- Quando qualquer uma dessas variáveis mudar, o efeito será executado novamente. Se a matriz de dependência estiver vazia, o efeito será executado apenas uma vez.

React Hooks

Outro aspecto importante do useEffect é a limpeza dos efeitos colaterais.

- Isso é importante quando você lida com operações que continuam após a desmontagem de um componente, como temporizadores ou assinaturas.
- Para limpar um efeito, você pode retornar uma função dentro do useEffect. Esta função de limpeza será executada quando o componente for desmontado e antes que o efeito seja executado novamente.

React Hooks

Neste exemplo, iniciamos um cronômetro quando o componente é montado e paramos o cronômetro quando o componente é desmontado.

```
import React, { useState, useEffect } from 'react';

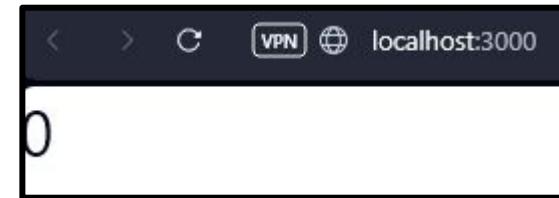
const Timer = () => {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const intervalId = setInterval(() => {
      setSeconds(seconds => seconds + 1);
    }, 1000);

    return () => {
      clearInterval(intervalId);
    }
  }, []);

  return (
    <div>
      {seconds}
    </div>
  )
}

export default Timer;
```



Custom Hooks

React Hooks

Hooks personalizados no React são um recurso poderoso que permite abstrair a lógica do componente em funções reutilizáveis.

- Essencialmente, um Hook personalizado é uma função JavaScript cujo nome começa com “use” e que pode chamar outros Hooks. Eles permitem que você use o estado e outros recursos do React sem escrever uma classe.
- Vejamos um exemplo: digamos que você tenha vários componentes que buscam dados de uma API. Você pode notar muita lógica duplicada para buscar dados, lidar com o estado de carregamento e lidar com erros. Este é um caso de uso perfeito para um Hook personalizado.



[Acesse o projeto](#)

React Hooks

Exemplo:

```
import useFetch from './useFetch';

function FetchPokemon({ pokemonId }) {
  const {
    data,
    loading,
    error,
  } = useFetch(`https://pokeapi.co/api/v2/pokemon/${pokemonId}`);

  if (loading) return 'Carregando...';
  if (error) return 'Erro!';

  return (
    <>
      <ul>
        {pokemon ? (
          <div>
            <h1>{pokemon.name}</h1>
            </div>
        ) : (
          'Carregando...'
        )}
      </ul>
    </>
  );
}

export default FetchPokemon;
```

```
import { useState, useEffect } from 'react';

const useFetch = (url) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch(url);
        const data = await response.json();
        setData(data);
        setLoading(false);
      } catch (error) {
        setError(error);
        setLoading(false);
      }
    };

    fetchData();
  }, [url]);

  return { data, loading, error};
}

export default useFetch;
```

React Hooks

O hook `useFetch` aceita uma URL como parâmetro, busca dados dessa URL e retorna um objeto com os dados, estado de carregamento e qualquer erro que ocorreu. Agora, qualquer componente que precise buscar dados pode usar este hook em vez de duplicar a lógica de busca.

- O Hook `useFetch` simplifica significativamente o componente `User`. O componente não precisa se preocupar em como buscar dados, como lidar com o estado de carregamento ou como lidar com erros. Ele apenas usa o hook `useFetch` e obtém estes dados :)

Prática

Prática

Nesta prática vamos utilizar alguns conceitos aprendidos até aqui para construir uma aplicação de ToDo :)

- Ao final da implementação, este será o resultado

✓ Lista de Tarefas

Nome da Tarefa

Baixa Prioridade

+ Adicionar Tarefa

Tarefa	Prioridade
--------	------------

Criação do projeto

Prática

Criando e abrindo o projeto

- Crie um novo projeto, chamarei ele de “todo-app”, para isso execute no terminal o seguinte comando:

```
npm create vite@latest todo-app -- --template react
```

- Em seguida, execute os comandos abaixo:

```
cd todo-app  
npm install  
npm run dev
```

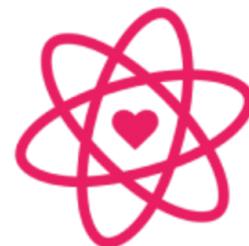
Prática

Criando e abrindo o projeto

- Em seguida, abra esta pasta no VSCode.
- Você pode fazer isso pela interface gráfica ou no mesmo terminal digitando:

```
cd ..  
code -r todo-app
```

Prática



react-icons

Instale a biblioteca react-icons

- A biblioteca react-icons é uma coleção de ícones populares e amplamente usados, projetados especificamente para serem usados em aplicativos React. Essa biblioteca oferece uma maneira fácil e conveniente de incorporar ícones em seus componentes React sem precisar criar imagens personalizadas ou lidar diretamente com SVGs.
- Para instalá-la, no terminal do projeto, execute:

```
npm install react-icons
```

TaskForm.js

Prática

TaskForm.js

- Crie um novo arquivo chamado TaskForm.js dentro da pasta src.
- Este arquivo conterá o componente TaskForm, responsável por renderizar o formulário de adição de novas tarefas na interface do aplicativo.

Prática

Arquivo TaskForm.js



[Clique aqui para ampliar a imagem](#)

```
import React, { useState } from 'react';
import './TaskForm.css';
import { RiAddLine } from 'react-icons/ri';

function TaskForm({ addTask }) {
  const [task, setTask] = useState('');
  const [priority, setPriority] = useState('low');

  const handleSubmit = (e) => {
    e.preventDefault();
    if (task.trim() === '') return;

    addTask({ task, priority });
    setTask('');
    setPriority('low');
  };

  return (
    <form className="task-form" onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Nome da Tarefa"
        value={task}
        onChange={(e) => setTask(e.target.value)}
      />
      <select
        value={priority}
        onChange={(e) => setPriority(e.target.value)}
      >
        <option value="low">Baixa Prioridade</option>
        <option value="medium">Média Prioridade</option>
        <option value="high">Alta Prioridade</option>
      </select>
      <button type="submit">
        <RiAddLine className="button-icon" /> Adicionar Tarefa
      </button>
    </form>
  );
}

export default TaskForm;
```

Prática

Importações

- `import React, { useState } from 'react';`: Importa as funcionalidades do React, incluindo o hook useState.
- `import './TaskForm.css';`: Importa o arquivo de estilos para o componente TaskForm.
- `import { RiAddLine } from 'react-icons/ri';`: Importa o ícone de adição da biblioteca react-icons.

Prática

Componente ‘TaskForm’

- Recebe a função `addTask` como prop. Essa função será chamada quando o formulário for submetido para adicionar uma nova tarefa à lista.
- Utiliza o hook `useState` para gerenciar os estados de `task` (nome da tarefa) e `priority` (prioridade da tarefa).
- `handleSubmit` é chamada quando o formulário é submetido. Ela verifica se o campo de tarefa não está vazio antes de chamar a função `addTask` com a nova tarefa e, em seguida, limpa os estados.

Prática

Componente 'TaskForm'

- Renderiza um formulário com um campo de entrada para o nome da tarefa, um seletor de prioridade e um botão de envio.
- O campo de entrada é vinculado ao estado `task` e atualizado conforme o usuário digita.
- O seletor de prioridade é vinculado ao estado `priority` e atualizado quando o usuário faz uma seleção.
- O botão de envio exibe o ícone de adição `<RiAddLine />` da biblioteca `react-icons` e também a etiqueta "Adicionar Tarefa".

TaskForm.css

Prática

TaskForm.css

- Crie um novo arquivo chamado TaskForm.css dentro da pasta src.
- Esse arquivo conterá os estilos CSS para o componente TaskForm no projeto

Prática

Arquivo TaskForm.css



[Clique aqui para
ampliar a imagem](#)

```
.task-form {  
    background-color: #ffffff;  
    border-radius: 8px;  
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);  
    padding: 20px;  
    margin: 20px;  
    display: flex;  
    flex-direction: column;  
    gap: 10px;  
}  
  
.task-form input,  
.task-form select {  
    padding: 10px;  
    border: 1px solid #ccc;  
    border-radius: 4px;  
    font-size: 16px;  
}  
  
.task-form button {  
    background-color: #007bff;  
    color: #ffffff;  
    border: none;  
    border-radius: 4px;  
    padding: 12px;  
    font-size: 16px;  
    cursor: pointer;  
    transition: background-color 0.3s ease;  
}  
  
.task-form button:hover {  
    background-color: #0056b3;  
}  
  
.button-icon {  
    margin-right: 5px;  
    vertical-align: middle;  
}
```

Prática

.task-form: Define os estilos para o contêiner do formulário de tarefas.

- **background-color:** Define a cor de fundo do contêiner.
- **border-radius:** Define o raio das bordas para arredondá-las.
- **box-shadow:** Adiciona uma sombra suave ao contêiner.
- **padding:** Define o espaçamento interno dentro do contêiner.
- **margin:** Define a margem externa do contêiner.
- **display: flex:** Aplica um layout flexível para os elementos internos.
- **flex-direction: column:** Define a direção da disposição como coluna, para que os elementos se empilhem verticalmente.
- **gap:** Define o espaçamento entre os elementos filhos do contêiner.

Prática

.task-form input, .task-form select: Define os estilos para os campos de entrada de texto e seleção (seletor de prioridade).

- **padding:** Adiciona um espaçamento interno nos campos.
- **border:** Define uma borda ao redor dos campos.
- **border-radius:** Arredonda as bordas dos campos.
- **font-size:** Define o tamanho da fonte nos campos.

Prática

.task-form button: Define os estilos para o botão de envio do formulário.

- **background-color:** Define a cor de fundo do botão.
- **color:** Define a cor do texto no botão.
- **border:** Remove a borda do botão.
- **border-radius:** Arredonda as bordas do botão.
- **padding:** Adiciona espaçamento interno no botão.
- **font-size:** Define o tamanho da fonte no botão.
- **cursor:** Define o cursor como ponteiro ao passar o mouse sobre o botão.
- **transition:** Adiciona uma transição suave para a mudança de cor de fundo.

Prática

.task-form button:hover: Define os estilos para o botão quando o cursor do mouse está sobre ele (estado de hover).

- **background-color:** background-color: Define uma cor de fundo diferente para quando o botão está sendo hoverado.

.button-icon: Define os estilos para o ícone de adição no botão.

- **margin-right:** Adiciona uma margem à direita para separar o ícone do texto.
- **vertical-align:** Alinha verticalmente o ícone no centro do botão.

TaskList.js

Prática

TaskList.js

- Crie um novo arquivo chamado TaskList.js dentro da pasta src.
- Esse arquivo contém o componente TaskList no projeto. Esse componente é responsável por renderizar a lista de tarefas na interface do aplicativo.

Prática

Arquivo TaskList.js



[Clique aqui para ampliar a imagem](#)

```
import React from 'react';
import './TaskList.css';
import { AiOutlineFileText, AiFillExclamationCircle } from 'react-icons/ai';

function TaskList({ tasks }) {
  const priorityLabels = {
    low: 'Baixa',
    medium: 'Média',
    high: 'Alta'
  };

  return (
    <table className="task-table">
      <thead>
        <tr>
          <th className="task-header">
            <AiOutlineFileText className="header-icon" /> Tarefa
          </th>
          <th>
            <AiFillExclamationCircle className="header-icon" /> Prioridade
          </th>
        </tr>
      </thead>
      <tbody>
        {tasks.map((task, index) => (
          <tr key={index}>
            <td className={`task-cell`}>{task.task}</td>
            <td>
              <span className={`priority-dot priority-${task.priority}`}>
                {priorityLabels[task.priority]}
              </span>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  );
}

export default TaskList;
```

Prática

Importações

- `import React from 'react';`: Importa as funcionalidades do React.
- `import './TaskList.css';`: Importa o arquivo de estilos para o componente TaskList.
- `import { AiOutlineFileText, AiFillExclamationCircle } from 'react-icons/ai';`: Importa os ícones de arquivo de texto e de exclamação da biblioteca react-icons.

Prática

Componente TaskList:

- Recebe a prop `tasks`, que é um array contendo as tarefas a serem exibidas.
- `priorityLabels` é um objeto que mapeia os valores de prioridade para seus rótulos correspondentes.

Prática

Renderização do Componente:

- Renderiza uma tabela (`<table>`) com cabeçalho e corpo.
- O cabeçalho (`<thead>`) contém uma linha (`<tr>`) com duas células (`<th>`) para os cabeçalhos das colunas Tarefa e Prioridade.
- Os ícones de arquivo de texto e de exclamação são exibidos ao lado dos títulos das colunas.

Prática

Renderização das Tarefas:

- No corpo (`<tbody>`) da tabela, mapeia o array de tarefas usando `.map()` para criar uma linha (`<tr>`) para cada tarefa.
- Cada tarefa é identificada pelo índice `index` no array, o que não é a melhor prática. Pode ser preferível usar um identificador único.
- Em cada linha, há duas células (`<td>`): uma para o nome da tarefa e outra para a prioridade.
- A célula de nome da tarefa possui a classe `task-cell`, que será usada para estilização.

Prática

Renderização das Tarefas:

- Na célula de prioridade, é exibido um ponto colorido (``) com a classe `priority-dot` para representar a prioridade da tarefa.
- A cor do ponto é definida com base na classe `priority-${task.priority}`, que depende do valor da prioridade da tarefa.
- O rótulo da prioridade é obtido usando o objeto `priorityLabels` com base no valor da prioridade da tarefa.

TaskList.css

Prática

TaskList.css

- Crie um novo arquivo chamado TaskList.css dentro da pasta src.
- Esse arquivo conterá os estilos CSS para o componente TaskList no projeto.

Prática

Arquivo TaskList.css



[Clique aqui para ampliar a imagem](#)

```
/* Estilos para a tabela de tarefas */
.task-table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
    font-size: 16px;
    text-align: left;
}

.task-table th,
.task-table td {
    border: 1px solid #ccc;
    padding: 12px;
    text-align: center;
    font-weight: normal;
    font-family: Arial, sans-serif;
    vertical-align: middle;
}

.task-table th {
    background-color: #f0f0f0;
    font-weight: bold;
}

.task-table tr:hover {
    background-color: #f5f5f5;
}

/* Estilos para os pontos de prioridade */
.priority-dot {
    display: inline-block;
    width: 12px;
    height: 12px;
    border-radius: 50%;
    margin-right: 4px;
}

.priority-dot.priority-low {
    background-color: #6ac259;
}

.priority-dot.priority-medium {
    background-color: #fdbc40;
}

.priority-dot.priority-high {
    background-color: #ff5c5c;
}

/* Estilos para a célula da tarefa */
.task-cell {
    width: 70%;
}

.task-header {
    width: 70%;
}

/* Estilos para ícones nos cabeçalhos */
.header-icon {
    margin-right: 5px;
    vertical-align: middle;
}
```

Prática

.task-table: Define os estilos gerais para a tabela de tarefas.

- **width: 100%;** Define a largura da tabela para ocupar 100% da largura do contêiner.
- **border-collapse: collapse;** Faz com que as bordas das células se unam para criar uma aparência mais limpa.
- **margin-top:** Adiciona margem superior à tabela.
- **font-size:** Define o tamanho da fonte para o texto na tabela.
- **text-align: left;** Alinha o texto à esquerda nas células.

Prática

.task-table th, .task-table td: Define os estilos para as células de cabeçalho (th) e células de dados (td) da tabela.

- **border:** Define uma borda ao redor das células.
- **padding:** Adiciona espaçamento interno nas células.
- **text-align: center:** Centraliza o conteúdo das células.
- **font-weight: normal:** Define o peso da fonte como normal.
- **font-family: Arial, sans-serif:** Define a família de fontes para a tabela.
- **vertical-align: middle:** Alinha verticalmente o conteúdo ao centro das células.

Prática

.task-table th: Define os estilos para as células de cabeçalho da tabela.

- **background-color:** Define a cor de fundo das células de cabeçalho.
- **font-weight: bold:** Define o peso da fonte como negrito para as células de cabeçalho.

.task-table tr:hover: Define os estilos para as linhas da tabela quando o cursor do mouse está sobre elas (estado de hover).

- **background-color:** Define uma cor de fundo diferente para quando a linha está sendo hoverada.

Prática

.priority-dot: Define os estilos para os pontos coloridos que representam a prioridade.

- **border: display: inline-block:** Faz com que os pontos se comportem como elementos de bloco, mas mantenham-se em linha com o texto.
- **width, height:** Define a largura e a altura dos pontos.
- **border-radius:** Arredonda as bordas para criar um círculo.
- **margin-right:** Adiciona margem à direita para separar os pontos do texto.

Prática

.priority-dot.priority-low .priority-dot.priority-medium .priority-dot.priority-high

- Define as cores dos pontos de prioridade com base na classe correspondente.

.task-cell, .task-header: Define os estilos para as células de tarefa e cabeçalho da tarefa.

- `width`: Define a largura das células.

Prática

.header-icon: Define os estilos para os ícones exibidos nos cabeçalhos das colunas.

- **margin-right:** Adiciona margem à direita para separar o ícone do texto.
- **vertical-align:** Alinha verticalmente o ícone no centro do cabeçalho.

App.js

Prática

App.js

- Edite o arquivo App.js que está dentro da pasta src. Caso não tenha o arquivo, crie-o.
- Esse arquivo é o ponto de entrada principal do aplicativo e contém o componente App no projeto. Ele define a estrutura geral da interface e gerencia o estado das tarefas.

Prática

Arquivo App.js



[Clique aqui para ampliar a imagem](#)

```
import React, { useState } from 'react';
import './App.css';
import { AiFillCheckCircle } from 'react-icons/ai';
import TaskForm from './TaskForm';
import TaskList from './TaskList';

function App() {
  const [tasks, setTasks] = useState([]);

  const addTask = (newTask) => {
    setTasks([...tasks, newTask]);
  };

  return (
    <div className="app">
      <h1 className="app-title">
        <AiFillCheckCircle className="app-icon" /> Lista de Tarefas
      </h1>
      <TaskForm addTask={addTask} />
      <div className="task-list-container">
        <TaskList tasks={tasks} />
      </div>
    </div>
  );
}

export default App;
```

Prática

Importações

- `Import React, { useState } from 'react';`: Importa as funcionalidades do React, incluindo o hook useState.
- `import './App.css';`: Importa o arquivo de estilos para o componente App.
- `import { AiFillCheckCircle } from 'react-icons/ai';`: Importa o ícone de marca de verificação da biblioteca react-icons.
- `import TaskForm from './TaskForm';`: Importa o componente TaskForm definido no arquivo TaskForm.js.
- `import TaskList from './TaskList';`: Importa o componente TaskList definido no arquivo TaskList.js.

Prática

Componente App:

- Define o componente App que é o ponto de entrada principal do aplicativo.

Estado e Função addTask:

- Utiliza o hook `useState` para criar o estado `tasks`, que é um array vazio para armazenar as tarefas.
- `addTask` é uma função que recebe uma nova tarefa como argumento e atualiza o estado `tasks` adicionando a nova tarefa ao final do array.

Prática

Renderização do Componente:

- Renderiza um contêiner principal (`<div>`) com a classe `app`, que envolve todo o conteúdo do aplicativo.
- Exibe o título do aplicativo usando o ícone de marca de verificação e a etiqueta "Lista de Tarefas".
- Renderiza o componente `TaskForm` e o componente `TaskList`, passando a função `addTask` como prop para o componente `TaskForm` e o estado `tasks` para o componente `TaskList`.

App.css

Prática

App.js

- Edite o arquivo App.css que está dentro da pasta src. Caso não tenha o arquivo, crie-o.
- Este arquivo contém os estilos CSS aplicados ao componente App no projeto. Esses estilos são responsáveis por definir a aparência geral do aplicativo.

Prática

Arquivo App.css



[Clique aqui para
ampliar a imagem](#)

```
.app {  
    text-align: center;  
}  
  
.app-title {  
    display: flex;  
    align-items: center;  
    justify-content: center;  
    font-size: 24px;  
    margin-top: 20px;  
}  
  
.app-icon {  
    margin-right: 10px;  
    font-size: 28px;  
    vertical-align: middle;  
}
```

Prática

.app: Define os estilos gerais para o contêiner principal do aplicativo.

- **text-align: center:** Centraliza o conteúdo dentro do contêiner.

.app-title: Define os estilos para o título do aplicativo.

- **display: flex:** Aplica um layout flexível aos elementos internos.
- **align-items: center:** Centraliza verticalmente os elementos filhos.
- **justify-content: center:** Centraliza horizontalmente os elementos filhos.
- **font-size:** Define o tamanho da fonte para o título.
- **margin-top:** Adiciona margem superior ao título.

Prática

.app-icon: Define os estilos para o ícone exibido junto ao título.

- **margin-right:** Adiciona margem à direita para separar o ícone do texto.
- **font-size:** Define o tamanho da fonte para o ícone.
- **vertical-align:** Alinha verticalmente o ícone no meio do título.

index.js

Prática

index.js

- Edite o arquivo index.js que está dentro da pasta src. Caso não tenha o arquivo, crie-o.
- Este arquivo é o ponto de entrada para a renderização do aplicativo React no projeto.

Prática

Arquivo index.js



[Clique aqui para
ampliar a imagem](#)

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <App />
)
```

Prática

Importações

- `import React from 'react';`: Importa as funcionalidades principais do React.
- `import ReactDOM from 'react-dom/client';`: Importa o módulo ReactDOM para renderização.
- `import './index.css';`: Importa o arquivo de estilos global para a aplicação.
- `import App from './App';`: Importa o componente App definido no arquivo App.js.

Prática

ReactDOM.createRoot:

- Cria uma raiz de renderização no elemento com o ID '`root`' no documento HTML.
- Isso é uma parte do novo modelo de renderização concorrente introduzido no React 18 e posterior.

Renderização do Componente App:

- Utiliza o método `root.render()` para renderizar o componente `App` dentro da raiz de renderização.
- Isso efetivamente inicia o processo de renderização do aplicativo.

`<App />`:

- Renderiza o componente `App`, que é o componente principal do aplicativo.

index.css

Prática

index.css

- Edite o arquivo `index.css` que está dentro da pasta `src`. Caso não tenha o arquivo, crie-o.
- Este arquivo contém estilos globais que são aplicados a toda a página HTML onde o aplicativo está sendo renderizado no projeto.

Prática

Arquivo index.js



[Clique aqui para ampliar a imagem](#)

```
body {  
  margin: 0;  
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',  
  'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',  
  sans-serif;  
  -webkit-font-smoothing: antialiased;  
  -moz-osx-font-smoothing: grayscale;  
}  
  
code {  
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',  
  monospace;  
}
```

Prática

body: Define estilos globais para o corpo da página HTML.

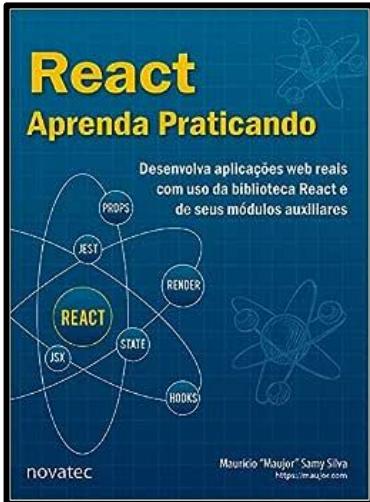
- **margin: 0;** Remove as margens padrão do corpo da página.
- **font-family:** Define uma lista de fontes para serem usadas como fontes padrão para o texto.
- **-webkit-font-smoothing, -moz-osx-font-smoothing:** Melhora a suavização da fonte em diferentes navegadores para uma aparência mais nítida.

code: Define estilos globais para elementos `<code>` (código inline).

- **font-family:** Define uma lista de fontes para serem usadas como fontes padrão para o código inline.

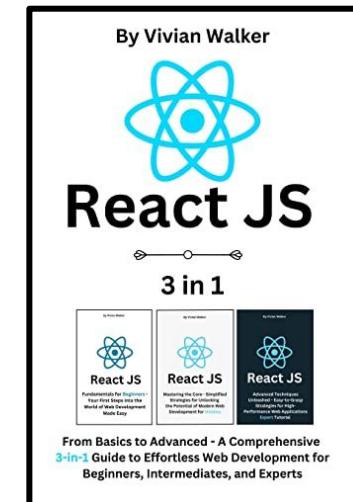
Referências

Referências



SILVA, Maurício Samy. **React - Aprenda praticando.**. Novatec. 2021.

Walker, Vivian. **React JS: From Basics to Advanced - A Comprehensive 3-in-1 Guide to Effortless Web Development for Beginners, Intermediates, and Experts.**



Análise e Desenvolvimento de Sistemas

Frameworks Web I

Aula 04 - React JS: parte 3

Prof. João Paulo F. C. César
joao cesar@unilavras.edu.br