Curso React do Zero

Professor: ***Jader Jackson*** **Atividade Resolvida:** 01
https://github.com/jaderjacksondeveloper/cursoReactAula01

“Se fosse fácil todo mundo era DEV…”

1) Crie uma aplicação de lista de tarefas (To-Do List) usando React. Nesta aplicação simples, você terá um componente principal (App) que renderiza uma lista de tarefas e permite adicionar novas tarefas.

// PREPARAÇÃO DO AMBIENTE
DICAS: Crie uma pasta com o nome atividadeComentada01
Abra essa pastas usando o visual studio code
pressione CRTL + J para abrir o terminal
No terminal crie o projeto usando o vite (npm create vite@latest atividadecomentada01 -- --template react)
Entre na pasta criada usando o seguinte comando: cd atividadecomentada01
Ainda no terminal instale as dependências (npm install)
Agora teste o seu servidor (npm run dev)
Pressione o CTRL e CLIK no endereço do servidor http://localhost:5173/
// FIM PREPARAÇÃO AMBIENTE
2) Crie o componente de Lista de Tarefas (TaskList):
Crie um novo arquivo chamado TaskList.js na pasta src. Este componente será responsável por exibir a lista de tarefas.

// src/TaskList.js

```
import React from 'react';

// Componente funcional TaskList que recebe um array de tarefas como propriedade (props)
const TaskList = ({ tasks }) => {
  return (
    // Renderiza uma lista não ordenada (ul) usando o método map para percorrer as tarefas
    <ul>
      {tasks.map((task, index) => (
        // Para cada tarefa, renderiza um item de lista (li) com a tarefa e uma chave única (index)
        <li key={index}>{task}</li>
      ))}
    </ul>
  );
};
// Exporta o componente TaskList para ser utilizado em outros arquivos
export default TaskList;
```

3) Crie o componente principal (App):
Abra o arquivo src/App.js e modifique-o para incluir o componente TaskList e um formulário para adicionar novas tarefas.

```
// src/App.js
import React, { useState } from 'react';
import TaskList from './TaskList'; // Importa o componente TaskList
// Componente funcional principal chamado App
const App = () => {
  // Define dois estados usando o hook useState: tasks (array de tarefas) e newTask (nova tarefa)
  const [tasks, setTasks] = useState([]);
  const [newTask, setNewTask] = useState('');
  // Função addTask para adicionar uma nova tarefa ao estado tasks
  const addTask = () => {
    // Verifica se a nova tarefa não está vazia antes de adicioná-la
    if (newTask.trim() !== '') {
      // Atualiza o estado tasks adicionando a nova tarefa e reinicia o estado newTask
      setTasks([...tasks, newTask]);
      setNewTask('');
    }
  };
  // Renderiza o conteúdo da aplicação
  return (
    <div>
      <h1>Minha Lista de Tarefas</h1>
      {/* Renderiza o componente TaskList passando o estado tasks como propriedade */}
      <TaskList tasks={tasks} />
      <div>
        {/* Input controlado: o valor é controlado pelo estado newTask */}
        <input
          type="text"
          value={newTask}
          // Atualiza o estado newTask quando o valor do input muda
          onChange={(e) => setNewTask(e.target.value)}
        />
        {/* Botão que chama a função addTask quando clicado */}
        <button onClick={addTask}>Adicionar Tarefa</button>
      </div>
    </div>
  );
};
// Exporta o componente App para ser utilizado em outros arquivos
export default App;
```

4) Rode o aplicativo:
npm run dev

Aprofundando meus conhecimentos:

Em React, os componentes podem receber dados externos por meio de propriedades, comumente chamadas de "props" (abreviação de "properties" ou propriedades em inglês). Quando você cria um componente, pode passar informações para ele definindo atributos nele quando você o utiliza.
No contexto do exemplo fornecido, o componente TaskList recebe um array de tarefas como uma propriedade. Veja o trecho de código em questão:

```
const TaskList = ({ tasks }) => {
  return (
    <ul>
      {tasks.map((task, index) => (
        <li key={index}>{task}</li>
      ))}
    </ul>
  );
};
```

No cabeçalho da função, { tasks } é uma desestruturação de propriedades no argumento da função. Isso significa que dentro do corpo da função, você pode usar tasks diretamente em vez de props.tasks. Essa desestruturação é uma forma concisa de extrair valores específicos de um objeto ou, neste caso, das propriedades do componente.
Quando você usa esse componente em outro lugar (como no componente App), você o faz assim:

```
<TaskList tasks={tasks} />
```

Aqui, tasks é uma variável do estado do componente App, e você está passando essa variável como uma propriedade chamada tasks para o componente TaskList. Portanto, o array de tarefas se torna acessível dentro do componente TaskList como props.tasks ou, devido à desestruturação no cabeçalho da função, simplesmente como tasks.
Isso permite que você reutilize o componente TaskList em diferentes partes do seu aplicativo e forneça diferentes conjuntos de tarefas, tornando o código mais modular e fácil de entender.