1ª Prova – Parte 01

Temas tratados: Conceitos *Javascript;* Conceitos *React JS;* Componentes React JS; Estado componentes.

Orientações

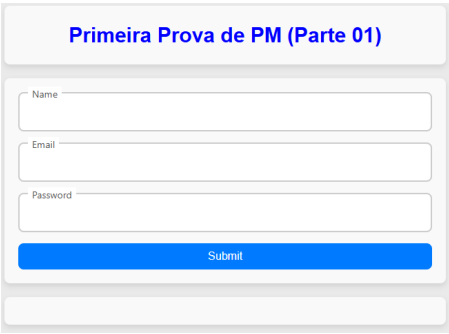
Considere que um projeto React JS foi criado com suporte ao Vite, e que a sequência de inicialização do front-end segue o padrão: o arquivo index.html executa o main.js, e este inclui o componente App.

A seguir, observe o código do componente App:

```
import React, { useState } from "react";
import reactLogo from './assets/react.svg';
import viteLogo from '/vite.svg';
import './container.css';

function App() {
  const [entries, setEntries] = useState([]);

  const handleSubmit = (e) => {
    e.preventDefault(); // impede o reload da página

    const formData = new FormData(e.target);
    const newEntry = {
      name: formData.get("name"),
      email: formData.get("email"),
      password: formData.get("password"),
    };

    setEntries((prev) => [...prev, newEntry]);
    e.target.reset(); // limpa o formulário após o envio
  };

  return (
    <>
      <div className="container">
        <div className="sub-container">
          <div className="header">
            Primeira Prova de PM (Parte 01)
          </div>
        </div>
        <div className="sub-container padding20 margin20">
          <form onSubmit={handleSubmit}>
            <div className="input-wrapper">
              <label htmlFor="name">Name</label>
              <input type="text" name="name" id="name" required />
            </div>
```

```
            <div className="input-wrapper">
              <label htmlFor="email">Email</label>
              <input type="email" name="email" id="email" required />
            </div>
            <div className="input-wrapper">
              <label htmlFor="password">Password</label>
              <input type="password" name="password" id="password" required />
            </div>
            <input type="submit" value="Submit" className="submit-btn" />
          </form>
        </div>
        <div id="content" className="sub-container">
          {entries.length > 0 && (
            <table style={{ width: "100%", borderCollapse: "collapse" }}>
              <thead>
                <tr>
                  <th>Name</th>
                  <th>Email</th>
                  <th>Password</th>
                </tr>
              </thead>
              <tbody>
                {entries.map((entry, index) => (
                  <tr key={index}>
                    <td>{entry.name}</td>
                    <td>{entry.email}</td>
                    <td>{entry.password}</td>
                  </tr>
                ))}
              </tbody>
            </table>
          )}
        </div>
      </div>
    </>
  );
}

export default App;
```

Ao carregar a página, o que se vê é o seguinte:



Questões

1. **Com base no código apresentado, o que a aplicação se propõe a implementar? Explique com suas palavras o objetivo principal do sistema.**

2. **Observe a função *handleSubmit* abaixo. Como seria possível modificar esse código para evitar que entradas com o mesmo e-mail sejam cadastradas mais de uma vez?**

   Função:

```
const handleSubmit = (e) => {
    e.preventDefault(); // impede reload da página

    const formData = new FormData(e.target);
    const newEntry = {
      name: formData.get("name"),
      email: formData.get("email"),
      password: formData.get("password"),
    };

    setEntries((prev) => [...prev, newEntry]);
    e.target.reset(); // limpa o formulário após o envio
 };
```

3. **Explique o que é executado no trecho de código a seguir. Comente sobre a estrutura map e seu papel na renderização da tabela:**

```
<tbody>
  {entries.map((entry, index) => (
    <tr key={index}>
      <td style={{ border: "1px solid #ccc", padding: "8px" }}>{entry.name}</td>
      <td style={{ border: "1px solid #ccc", padding: "8px" }}>{entry.email}</td>
      <td style={{ border: "1px solid #ccc", padding: "8px" }}>{entry.password}</td>
    </tr>
  ))}
</tbody>
```

4. **Crie dois componentes *React*: Container e SubContainer. Eles devem substituir as s com className="container" e className="sub-container", respectivamente. Os novos componentes devem preservar tanto a estrutura funcional quanto o estilo visual da aplicação original.**