Roteiro de Atividade Prática

Nome: Gabriel Turma: .

Nome: Ezequiel Turma: .

**Atividade: Desenvolvendo uma aplicação de lista de tarefas**

Você foi contratado para desenvolver uma aplicação de lista de tarefas simples (To-Do List), na qual os usuários podem adicionar tarefas, marcar como concluídas e removê-las. Além disso, você deve implementar uma funcionalidade que salva automaticamente as tarefas no armazenamento local do navegador (localStorage) sempre que a lista é atualizada.

Para isso, você deve usar os *hooks* useState e useEffect para gerenciar o estado da aplicação e sincronizar as tarefas com o armazenamento local. Também deve manipular eventos para adicionar e remover tarefas.

**Tutorial:**

**1. Configuração do Projeto**

**Crie um Novo Projeto React:**

* + Execute o comando:

|  |
| --- |
| npx create-react-app todo-list  Navegue até a pasta do projeto:  cd todo-list  **Limpeza do Projeto:**   * Abra o projeto em seu editor de código preferido (VS Code, por exemplo). * No arquivo src/App.js, remova todo o conteúdo padrão e substitua por:   import React, { useState, useEffect } from 'react';  function App() {  return (  <div className="App">  <h1>To-Do List</h1>  </div>  );  }  export default App;  **Estilização Básica:**   * Crie um arquivo src/App.css e adicione estilos básicos:   .App {  text-align: center;  font-family: Arial, sans-serif;  }  .todo-item {  display: flex;  justify-content: space-between;  margin: 5px 0;  }  .completed {  text-decoration: line-through;  color: grey;  }  **Importe o Arquivo CSS no App.js:**  import './App.css';  **2. Implementando a Lógica com useState e useEffect**  **Gerenciamento de Tarefas:**   * + No App.js, adicione um estado para armazenar as tarefas:   const [tasks, setTasks] = useState([]);  const [input, setInput] = useState('');  **Adicionando Tarefas:**   * Crie uma função para adicionar tarefas:   const addTask = () => {  const newTask = { text: input, completed: false };  setTasks([...tasks, newTask]);  setInput('');  };  Adicione um campo de entrada e um botão para adicionar tarefas:  <input  type="text"  value={input}  onChange={(e) => setInput(e.target.value)}  placeholder="Nova tarefa"  />  <button onClick={addTask}>Adicionar</button>  **Marcando Tarefas como Concluídas:**   * Crie uma função para alternar o estado de conclusão de uma tarefa:   const toggleComplete = (index) => {  const newTasks = tasks.map((task, i) =>  i === index ? { ...task, completed: !task.completed } : task  );  setTasks(newTasks);  };  **Removendo Tarefas:**   * Crie uma função para remover tarefas:   const removeTask = (index) => {  const newTasks = tasks.filter((\_, i) => i !== index);  setTasks(newTasks);  };  **3. Sincronizando com useEffect e localStorage**  **Salvando Tarefas no Armazenamento Local:**   * + Use useEffect para salvar as tarefas sempre que elas forem atualizadas:   useEffect(() => {  localStorage.setItem('tasks', JSON.stringify(tasks));  }, [tasks]);  **Carregando Tarefas do Armazenamento Local:**   * Use useEffect para carregar as tarefas ao iniciar a aplicação:   useEffect(() => {  const savedTasks = JSON.parse(localStorage.getItem('tasks'));  if (savedTasks) {  setTasks(savedTasks);  }  }, []);  **4. Renderizando a Lista de Tarefas**   1. **Renderize as Tarefas na Tela:**    * No App.js, adicione a seguinte lógica de renderização:   return (  <div className="App">  <h1>To-Do List</h1>  <input  type="text"  value={input}  onChange={(e) => setInput(e.target.value)}  placeholder="Nova tarefa"  />  <button onClick={addTask}>Adicionar</button>  <ul>  {tasks.map((task, index) => (  <li key={index} className="todo-item">  <span  className={task.completed ? 'completed' : ''}  onClick={() => toggleComplete(index)}  >  {task.text}  </span>  <button onClick={() => removeTask(index)}>Remover</button>  </li>  ))}  </ul>  </div>  );  Registre no AVA suas percepções em relação ao código desenvolvido e quais foram os principais desafios encontrados durante o desenvolvimento dessas funcionalidades. |

* **Uso Eficiente de Hooks:** O código demonstra um uso eficaz dos hooks useState e useEffect para gerenciar o estado da aplicação e lidar com efeitos colaterais, como a interação com o localStorage.

* **Componentização:** A estrutura do código é bem organizada, com a lógica da aplicação encapsulada no componente App, facilitando a manutenção e o entendimento.

* **Manipulação de Estado:** O código manipula o estado de forma clara e concisa, utilizando a imutabilidade para atualizar o estado tasks, o que é uma prática recomendada em React.
* **Interação com o DOM:** A interação com o DOM é realizada de forma eficiente através do uso de eventos e manipulação de estilos com classes CSS.
* **Persistência de Dados:** A implementação da persistência de dados utilizando o localStorage é simples e eficaz, permitindo que as tarefas sejam mantidas mesmo após o fechamento do navegador.

**Desafios Encontrados:**

* **Gerenciamento do Estado:** Um dos principais desafios é gerenciar o estado da aplicação de forma eficiente, especialmente ao lidar com a atualização e a remoção de tarefas.
* **Sincronização com o localStorage:** A sincronização do estado com o localStorage pode apresentar desafios, como garantir que os dados sejam salvos e carregados corretamente e lidar com possíveis erros.
* **Manipulação de Eventos:** A manipulação de eventos, como adicionar, concluir e remover tarefas, requer um entendimento claro de como os eventos funcionam em React e como atualizar o estado em resposta a esses eventos.
* **Renderização Condicional:** A renderização condicional, como aplicar estilos diferentes a tarefas concluídas, pode ser um desafio para iniciantes em React.
* **Performance:** Em aplicações maiores, otimizar a performance da renderização da lista de tarefas pode ser um desafio, especialmente ao lidar com um grande número de tarefas.