# Tema 06 – Projeto Agenda de Compromissos – Parte 02

## Introdução

Olá, futuras mentes desenvolvedoras, sejam bem-vindos do curso de Frontend do Instituto da Oportunidade Social. Nessa aula, você continuará a implementar o projeto da agenda de compromissos/tarefas utilizando a abordagem de componentes funcionais.

Atenção: Continue a implementação a partir do mesmo ponto que paramos no tema passado.

## Criar o componente Tasks

Vamos agora criar o componente **Tasks**, que será responsável por exibir a lista de tarefas da aplicação. Siga os seguintes passos para fazer a implementação.

1. No diretório **Components**, crie um arquivo **Tasks.jsx**. A organização do seu diretório deve ficar como mostrado na figura abaixo

Graphical user interface, text, application

Description automatically generated

1. Agora insira o código abaixo no arquivo **Tasks.jsx**:

|  |  |
| --- | --- |
| 1 | const tasks = [      {          id: 1,          text: 'Consulta médica',          day: '5 de Fev as 14:30',          reminder: true,      },      {          id: 2,          text: 'Reunião na Escola',          day: '6 de Fev as 13:30',          reminder: true,      },      {          id: 3,          text: 'Compras no Supermercado',          day: '7 de Fev as 8:30',          reminder: false,      },  ];  const Tasks = () => {      return (          <>              {tasks.map((task) => (                  <h3>{task.text}</h3>              ))}          </>      );  };  export default Tasks; |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |
| 16 |
| 17 |
| 18 |
| 19 |
| 20 |
| 21 |
| 22 |
| 23 |
| 24 |
| 25 |
| 26 |
| 27 |
| 28 |
| 29 |
| 30 |
| 31 |
| 32 |

Observe que esse componente tem uma variável chamada **tasks** com as tarefas que queremos exibir na interface do usuário (linhas 1 a 20). Inserir os dados em um componente torna-os visível apenas para esse componente e seus filhos. Portanto, esses dados ficarão no componente **Tasks** temporariamente. Mais pra frente atualizaremos eles para serem o estado da aplicação React.

Note também que utilizamos o método de alto nível para trabalhar com array e criar um a um cada texto das tarefas para serem exibidos na interface do usuário.

1. Então, agora precisamos exibir o componente Tasks na interface do usuário. Para isso, precisamos fazer com que o componente seja renderizado na página web. No arquivo App.js, devemos atualizar o código da seguinte forma:

|  |  |
| --- | --- |
| 1 | import Header from './Components/Header';  import Tasks from './Components/Tasks';  function App() {      return (          <div className="container">              <Header title="tarefas" />              <Tasks />          </div>      );  }  export default App; |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |

Na linha 2 do código mostrado, temos a instrução para importar o componente filho **Tasks** no componente pai **App** e, desse modo, ele ser reconhecido como um componente válido. Caso você esqueça de colocar essa instrução, a aplicação acusará erro na linha 8.

A linha 8 contém o nosso componente indicando que ele deve ser renderizado e colocado após o componente **Tasks**.

1. Salve o arquivo App.js e você pode ver o resultado no navegador web.

Graphical user interface, text, application, chat or text message

Description automatically generated

Mas se você abrir o console, notará que a aplicação contém um aviso (Warning).

Graphical user interface, text, application, chat or text message, email

Description automatically generated

Esse aviso indica que cada filho criado em uma lista deve conter uma chave (**key**) única. No nosso exemplo, estamos cirando uma lista de elemento <h3> através de método map(), portanto devemos passar uma key única a cada elemento criado. Para remover esse aviso, você deve atualizar o código como mostrado na linha 26. Nessa linha usamos o campo ID de cada uma das tarefas, pois esse campo não possui valor igual e é único para cada tarefa.

|  |  |
| --- | --- |
| 1 | const tasks = [      {          id: 1,          text: 'Consulta médica',          day: '5 de Fev as 14:30',          reminder: true,      },      {          id: 2,          text: 'Reunião na Escola',          day: '6 de Fev as 13:30',          reminder: true,      },      {          id: 3,          text: 'Compras no Supermercado',          day: 'Feb 7th at 1:30pm',          reminder: false,      },  ];  const Tasks = () => {      return (          <>              {tasks.map((task) => (                 <h3 key={task.id}>{task.text}</h3>              ))}          </>      );  };  export default Tasks; |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |
| 16 |
| 17 |
| 18 |
| 19 |
| 20 |
| 21 |
| 22 |
| 23 |
| 24 |
| 25 |
| 26 |
| 27 |
| 28 |
| 29 |
| 30 |
| 31 |
| 32 |

Observe que agora o aviso não parece mais.

Graphical user interface, text, application, chat or text message, email, website

Description automatically generated

### Estado da aplicação

Como citamos anteriormente, vamos fazer com que as tarefas sejam um estado na aplicação e vamos transformá-lo em um estado global, que poderá ser acessado por todos os componentes da aplicação. Para isso, devemos realizar as seguintes atualizações no nosso código:

1. No arquivo App.js, vamos atualizar o código.
2. **import** Header from './Components/Header';
3. **import** Tasks from './Components/Tasks';
4. **import** { useState } from 'react';
6. **function** App() {
7. **const** [tasks, setTasks] = useState([
8. {
9. id: 1,
10. text: 'Consulta médica',
11. day: '5 de Fev as 14:30',
12. reminder: **true**,
13. },
14. {
15. id: 2,
16. text: 'Reunião na Escola',
17. day: '6 de Fev as 13:30',
18. reminder: **true**,
19. },
20. {
21. id: 3,
22. text: 'Compras no Supermercado',
23. day: '7 de Fev as 8:30',
24. reminder: **false**,
25. },
26. ]);
28. **return** (
29. <div className="container">
30. <Header title="tarefas" />
31. <Tasks tasks={tasks} />
32. </div>
33. );
34. }
36. **export** **default** App;

Aqui estamos usando o *hook* useState (linha 6) para definir tasks como um estado da aplicação React. Isso é importante nesse projeto, pois mais a diante iremos precisar que outros componentes acessem as informações contidas nesse estado.

Além disso, na linha 30, estamos passando o conteúdo do estado por meio da propriedade tasks.

1. No arquivo Task,jsx, precisamos agora modificar o código para receber a propriedade e manipulá-la para exibir corretamente na interface do usuário. Desse modo, atualize o código do componente Tasks conforme mostrada abaixo.

|  |  |
| --- | --- |
| 1 | const Tasks = ({ tasks }) => {      return (          <>              {tasks.map((task) => (                  <h3 key={task.id}>{task.text}</h3>              ))}          </>      );  };  export default Tasks; |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |

1. O resultado visual no navegador é o mesmo, mas essa atualização é interessante pois, além de utilizarmos um recurso importante do React, será possível que outros componentes acessem o estado da aplicação. O que muda é que agora o componente **Tasks** possui uma propriedade **tasks** por onde vem o conteúdo do estado global da aplicação.

Graphical user interface, text, application, email, website

Description automatically generated

### Criando o componente TaskItem

Em programação sempre devemos “dividir para conquistar”, isto é, não devemos criar uma função, elemento, componente Chimera (pronuncia-se Quimera), que possui todas as partes do código e realiza todas as ações, etc. Desse modo, vamos criar um elemento filho do componente **Tasks**, que será responsável por criar a exibição de cada agendamento e, assim, dividir a responsabilidade com o componente Pai. Siga os passos para criar o componente TasksItem.

1. No diretório **Components**, crie um arquivo **TaskItem.jsx**. A organização do seu diretório deve ficar como mostrado na figura abaixo

Graphical user interface, application

Description automatically generated

1. Agora insira o código abaixo no arquivo **TaskItem.jsx**:
2. **const** TaskItem = () => {
3. **return** (
4. <div className="task">
5. <h3>Minha Tarefa</h3>
6. </div>
7. );
8. };
10. **export** **default** TaskItem;
11. No arquivo Tasks.js, vamos modificar o código para incluir o componente TaskItem e passar as informações por meio de propriedades para esse componente. Atualize o código como mostrado abaixo.

|  |  |
| --- | --- |
| 1 | import TaskItem from './TaskItem';  const Tasks = ({ tasks }) => {      return (          <>              {tasks.map((task) => (                  <TaskItem key={task.id} task={task} />              ))}          </>      );  };  export default Tasks; |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |

1. Se você visualizar a aplicação no navegador web verá a seguinte interface.

Graphical user interface, application

Description automatically generated

Isso porque o nosso código do componente TaskItem possui um valor estático com o elemento <h3> e o texto minha tarefa.

A informação aparece três vezes porque o método map() percorre os três elementos dentro do array de objeto **tasks**. Desse modo, são criados três elementos TaskItem com Keys diferente como podemos ver na extensão React Developer Tools.

1. Vamos modificar o código do componente **TaskItem** para que ele atualize o conteúdo dinamicamente. Atualize o código como mostrado abaixo:
2. **const** TaskItem = ({ task }) => {
3. **return** (
4. <div className="task">
5. <h3>{task.text}</h3>
6. <p>{task.day}</p>
7. </div>
8. );
9. };
11. **export** **default** TaskItem;
12. Agora podemos ver no navegador web a nossa aplicação funcionando.

Graphical user interface, text, application

Description automatically generated

1. Se você clicar em cada um dos componentes TaskItem na extensão React Developer Tools, poderá ver que cada um tem o conteúdo de um item do array de objetos tasks, que é nosso estado global da aplicação.

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

### Instalando e utilizando um módulo no React

Uma das vantagens do React é que podemos utilizar o gerenciador de pacotes NPM para instalar módulos de terceiros e, assim, podemos utilizar na aplicação. Vamos fazer isso para instalar o pacote react-icons. Esse pacote contém diversas fontes de ícones para serem utilizados em uma página web. No nosso caso, estamos interessados nos ícones do Font Awnsome. Siga os passos para instalar a biblioteca react-icons.

1. No terminal do VS Code, tenha certeza de que você está no diretório do projeto e execute o seguinte comando:

npm i react-icons

1. Com a instalação realizada com sucesso, você pode ver o pacote instalado dentro da pasta node-modules

Text

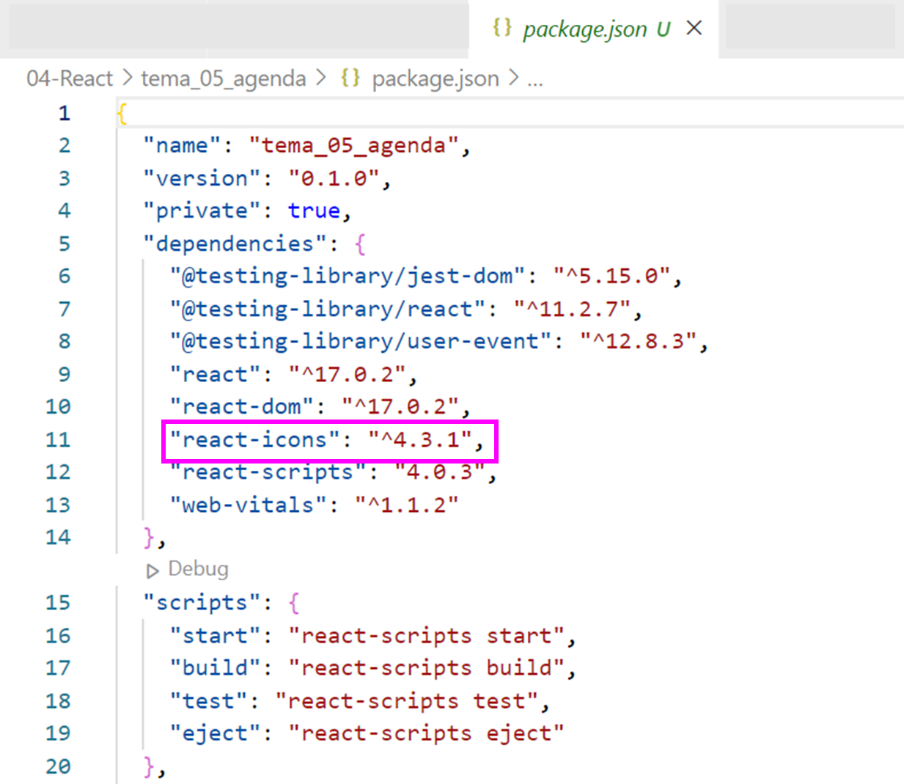
Description automatically generated

1. Dentro desse pacote, temos a pasta fa de Font Awnsome:

Table

Description automatically generated with medium confidence

Você não precisa expandir esse diretório, pois ele é muito grande e você pode se perder na hierarquia dos arquivos do projeto depois, quando voltarmos com a implementação. Você pode conferir de uma forma mais fácil pelo arquivo package.json



Se o pacote não estiver instalado, a aplicação acusará erro nos próximos passos e, então, você deverá fazer a instalação corretamente para funcionar.

1. Então, devemos agora atualizar o código do componente TaskItem como mostrado abaixo.
2. **import** { FaTimes } from 'react-icons/fa';
4. **const** TaskItem = ({ task }) => {
5. **return** (
6. <div className="task">
7. <h3>
8. {task.text}
9. <FaTimes style={{ color: 'red', cursor: 'pointer' }} />
10. </h3>
11. <p>{task.day}</p>
12. </div>
13. );
14. };
16. **export** **default** TaskItem;
17. Agora podemos ver no navegador web o ícone X que será usado para “deletar” o agendamento. No momento, esse ícone não faz nada.

Graphical user interface, application

Description automatically generated