

IOS – Instituto de
Oportunidade Social

React 07 - Projeto Agenda de Compromissos Parte 03



- Passando função por parâmetro;
- Extensão React Developer Tools;
- Renderização condicional;
- Alterando status de atributos;

IOS – Instituto de
Oportunidade Social

Passando função por parâmetro



Até o momento passamos alguns **parâmetros** para função, principalmente com valores de variáveis como: textos, números, booleanos, etc.

Porém o tipo **função** também é válido como passagem de parâmetro para outros **componentes**, sendo possível utilizar o conceito de **arrow function** na criação desse tipo de **variável**.

Passando função por parâmetro

```
import TaskItem from './TaskItem';

const Tasks = ({ tasks, onDelete }) => {
  return (
    <>
      {tasks.map((task) => (
        <TaskItem key={task.id} task={task} onDelete={onDelete} />
      ))}
    </>
  );
};
```

IOS – Instituto de
Oportunidade Social


React Developer Tools




React Developer Tools é uma **extensão** do **Chrome DevTools** para a biblioteca **React** JavaScript de código aberto. Ele permite que você inspecione as hierarquias de componentes do React nas Ferramentas do desenvolvedor do Chrome.

A guia **Componentes** mostra os componentes **raiz do React** que foram **renderizados** na página, bem como os **subcomponentes** que eles acabaram renderizando.

Extensão:

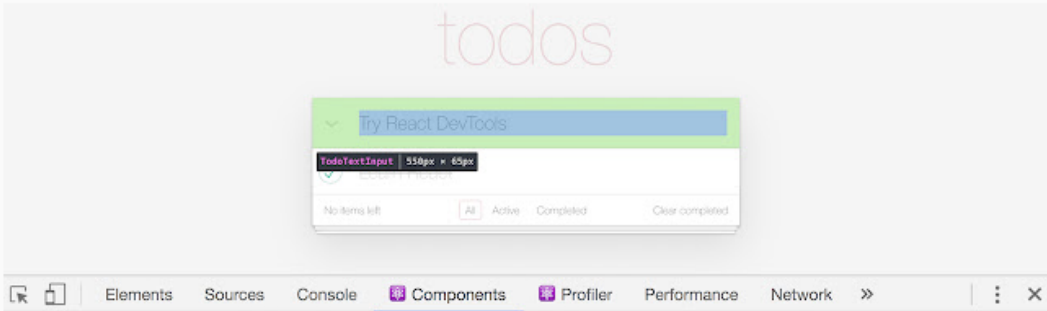


Ferramentas do desenvolvedor do React

 Destaque

★★★★★ 1.390 ⓘ | [Ferramentas de desenvolvimento](#) | Mais de 3.000.000 de usuários

[Visão geral](#) [Práticas de privacidade](#) [Avaliações](#) [Apoiar](#) [Relacionado](#)



IOS – Instituto de
Oportunidade Social

Renderização condicional



Quando a **renderização de Componente** depende de alguns **fatores**, ou seja, nem sempre ele deverá ser renderizado, então podemos utilizar o conceito de **Renderização condicional**.

Lembrando que a **verificação** da renderização do componente é em **tempo de execução**. Dessa forma ele poderá iniciar renderizado e depois sair da página e vice-versa.

Exemplo:

```
return (  
  <div className="container">  
    <Header title="tarefas" />  
    {tasks.length > 0 ? (  
      <Tasks tasks={tasks} onDelete={deletaTarefa} />  
    ) : (  
      'Você não tem tarefas, pode tirar férias!'  
    )}  
  </div>  
);
```

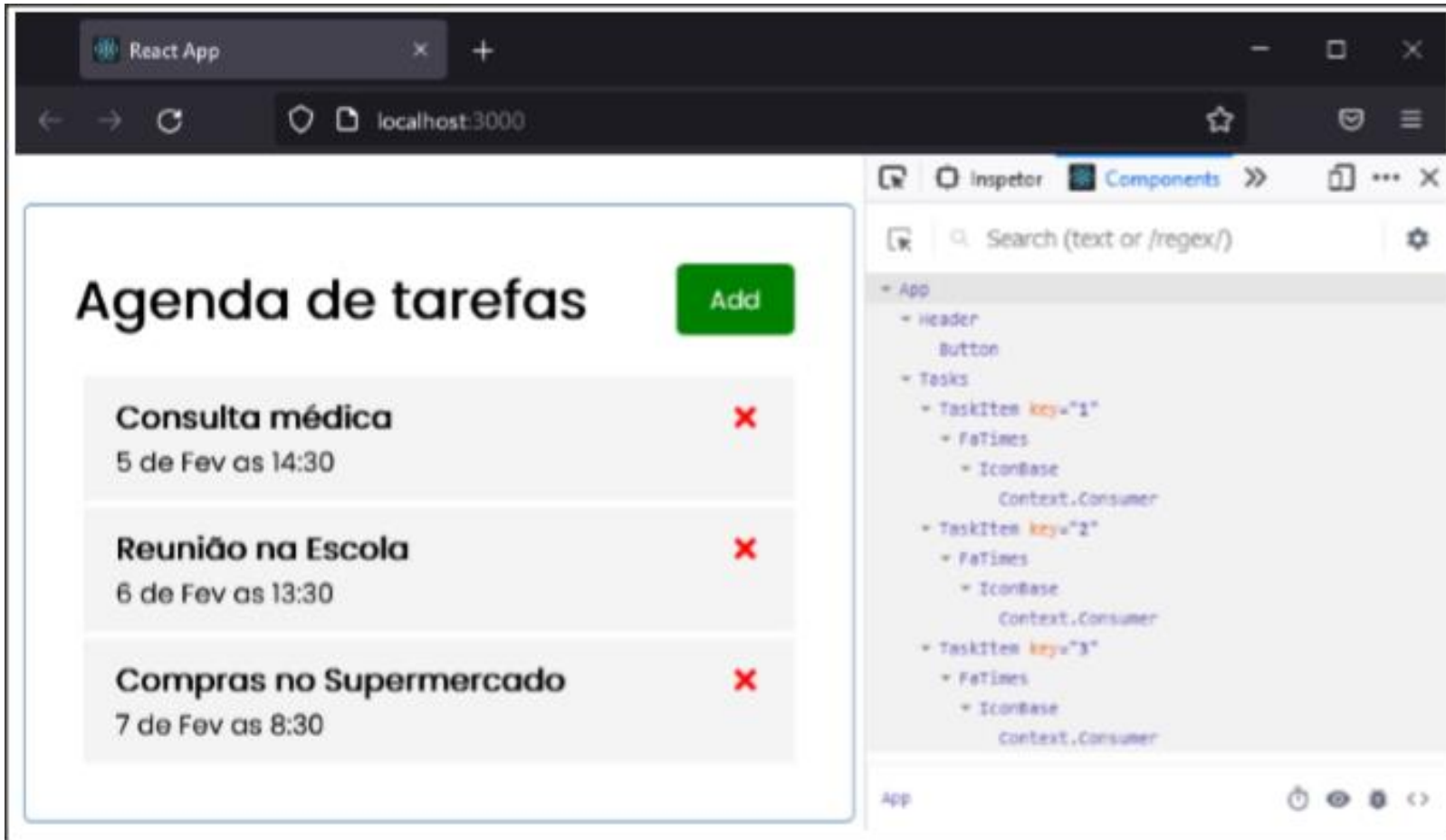
Renderização condicional

tasks:

```
const [tasks, setTasks] = useState([
  {
    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,
  },
])
```

Renderização condicional

Com tasks:



The screenshot shows a web application titled "React App" running on `localhost:3000`. The main content area displays a task agenda titled "Agenda de tarefas" with a green "Add" button. The agenda lists three tasks, each with a red "X" icon:

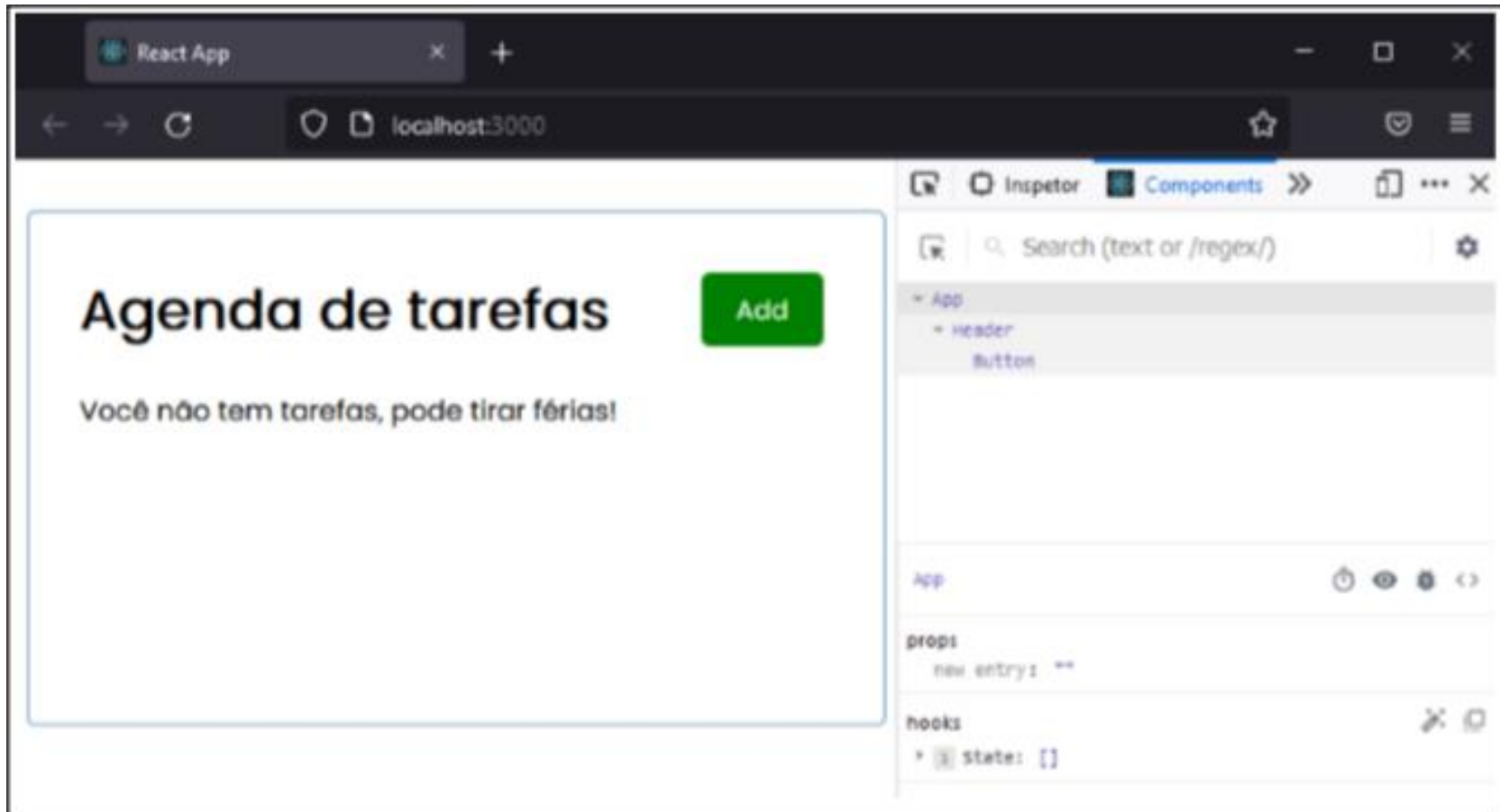
- Consulta médica**
5 de Fev as 14:30
- Reunião na Escola**
6 de Fev as 13:30
- Compras no Supermercado**
7 de Fev as 8:30

The right sidebar shows the React DevTools component inspector. The "Components" tab is active, displaying the component tree for the "App" component. The tree structure is as follows:

- App
 - Header
 - Button
 - Tasks
 - TaskItem `key="1"`
 - FaTimes
 - IconBase
 - Context.Consumer
 - TaskItem `key="2"`
 - FaTimes
 - IconBase
 - Context.Consumer
 - TaskItem `key="3"`
 - FaTimes
 - IconBase
 - Context.Consumer

Renderização condicional

Sem tasks:



IOS – Instituto de
Oportunidade Social

Alterando status de atributo



Alterando status de atributo



Vamos criar o **evento** para mudar o valor booleano (de **true** para **false** ou de false para true) do campo **reminder** de cada item no array de objetos. Esse evento será **disparado** quando **clicarmos duas vezes sobre a tarefa** agendada. A função desse evento vai estar implementada no componente App e será passada para os componentes filhos através de **props**.

Alterando status de atributo

Função para alterar o reminder:

```
const mudarReminder = (id) => {  
  setTasks(  
    tasks.map((task) =>  
      task.id === id ? { ...task, reminder: !task.reminder } : task  
    )  
  );  
  console.log(id);  
};
```

Passando a função mudarReminder por parâmetro:

```
return (  
  <div className="container">  
    <Header title="tarefas" />  
    {tasks.length > 0 ? (  
      <Tasks  
        tasks={tasks}  
        onDelete={deletaTarefa}  
        onToggle={mudarReminder}  
      />  
    ) : (  
      'Você não tem tarefas, pode tirar férias!'  
    )}  
  </div>  
);
```

Alterando status de atributo



Criamos a função **mudarReminder**, que utiliza o método de alto nível **map** para buscar a tarefa que queremos alterar o valor booleano do campo **reminder**. Utilizamos operador **rest (...)** para **atualizar** o **array tasks** com o **objeto** que teve o **reminder** alterado.

Além disso, queremos passar essa **função como propriedade** para o componente **TaskItem**. Então temos que primeiro passar para o componente **Tasks** e depois para o componente **TaskItem**. Pois é necessário respeitar a **hierarquia** da árvore de componentes. Não podemos saltar componentes na árvore.

Desse modo, passamos a função **mudarReminder** através da **propriedade onToggle** para o componente **Tasks**.

Alterando status de atributo

Utilizando className de forma condicional:

```
<div
```

```
  className={`task ${task.reminder ? 'reminder' : ''}`}  
  onClick={() => onToggle(task.id)}
```

>

Com a classe reminder

Consulta médica
5 de Fev as 14:30



Sem a classe reminder

Compras no Supermercado
7 de Fev as 8:30



IOS – Instituto de
Oportunidade Social

Vamos Praticar



Apostila de React

05.React

Páginas 79 a 87

OBS: Acompanhar o passo a passo com o instrutor. Quem não acompanhou a Parte 1 e 2, baixar o material de apoio no Moodle.

IOS – Instituto de
Oportunidade Social

Exercícios



Criar 3 Componentes: 1) **NewTasks.jsx** estilo header com um título e um botão **Add** para adicionar Tasks.

2) **TasksOpen.jsx** contendo as tarefas em aberto, caso não haja tarefas em aberto exibir a mensagem '**Não há tarefas em aberto.**' cada tarefa deverá ter 2 botões: **FaTimes [X]** e **FaToggleOff** o primeiro para excluir a tarefa e o segundo para concluir a tarefa e mover para as finalizadas.

3) **TasksDone.jsx** contendo as tarefas finalizadas, caso não haja tarefas finalizadas exibir a mensagem '**Não há tarefas concluídas.**' cada tarefa deverá ter 2 botões: **FaTimes [X]** e **FaToggleOn** o primeiro para excluir a tarefa e o segundo para reabrir a tarefa e mover para em aberto. Realizar o processo de Build (**npm run build**) do projeto praticado anteriormente e subir no GitHub.