1. Atividades Práticas - React Hooks

Nesta atividade prática, vamos trabalhar um pouco mais com os conceitos de Rect Hooks.

1.1 Criando nosso Projeto

Vamos criar um projeto chamado my-react-hooks, digitando o seguinte comando no terminal:

```
yarn create react-app my-react-hooks
```

O processo leva alguns segundos e, logo após terminar, digite o seguinte comando para entrar no diretório recém criado do nosso projeto:

```
cd my-react-hooks
```

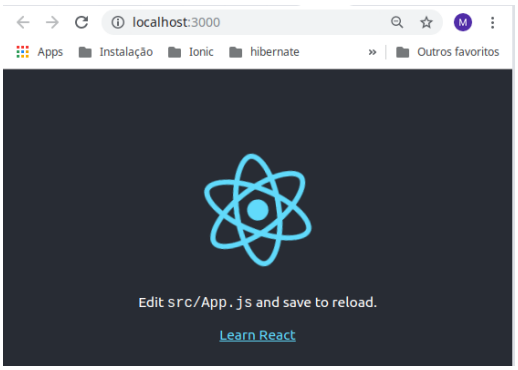
Você pode abrir a aplicação no editor Visual Studio Code. Para isso, dentro do diretório my-react-hooks digite o seguinte comando:

```
code .
```

O Visual Studio Code deverá abrir com a pasta my-react-hooks sendo acessada. Agora, vamos testar nossa aplicação. Digite a seguinte linha de comando:

```
yarn start
```

Ele irá rodar a aplicação em modo desenvolvimento em http://localhost:3000



A página será recarregada automaticamente se você fizer alterações no código.

React Hooks é a nova API do React para agilizar na construção de componentes, pois ela diminuiu a verbosidade na parte de compartilhamentos de componentes, estado e ciclo de vida.

Cabe lembrar que anteriormente, para poder ter estado na aplicação, precisaríamos criar um componente com classes, ou seja, uma classe que estende de React.Component, definir a variável state e informar seus estados. Na versão 16.8 a API Hooks foi criada para poder diminuir a verbosidade no código evitando a necessidade de usar classes para definir componentes com estados e manipulação do ciclo de vida.

1.2 Exercício 1

O primeiro Hook que vamos trabalhar é o useState. Resumidamente, o useState retorna o estado e a função que atualiza o estado. Veja um exemplo abaixo:

```
const [tech, setTech] =  useState([]);
```

Acima, criamos uma const que recebe de forma desestruturada uma variável de estado e uma função que atualiza essa variável de estado, o useState recebe como parâmetro o valor inicial do estado, que no nosso caso é um array vazio.

Bem, crie um arquivo chamado Exe1.js. Crie um estado com valor inicial contendo um array com os seguintes dados: 'ReactJS', 'ReactNative', 'NodeJS'

Após, mostre estes valores na tela através de uma lista não ordenada (use a da tag <ul>). Cabe perceber que você poderá fazer uso da função map() do javascript. Abaixo, iteramos pelo array numbers usando a função map() do JavaScript. Retornamos um elemento <li> para cada item. Finalmente, atribuímos o array de elementos resultante para listItems:

```
const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number) =>

 <li>{number}</li>

);
```

1.2 Exercício 2

Neste segundo exercício vamos continuar com o hook useState. Bem, crie um arquivo chamado Exe2.js. Copie e cole o código do Exe1.js, pois será continuidade dele. Bem, no exercício anterior nós tínhamos criado um estado chamado tech e uma função chamada setTec():

```
const [tech, setTech] =  useState([ .... ]);
```

Agora vamos alterar o estado com a função setTech(). Assim, crie um botão na tela que ao ser clicado (evento onClick()) irá chamar uma função que vai adicionar um item (valor igual a ‘Next.js’) no array do estado.

Assim, quando o usuário clicar no botão Adicionar, uma função será executada, o setTech vai ser invocado e o como o estado é imutável, replico o valor do array (use o spread operator) e adiciono o novo valor. Spread significa espalhar, ou seja, este operador é usado para ‘espalhar’ os elementos de um array quando interpretado em tempo de execução.

Lembre-se que a variável tech armazena todos os dados e o setTech altera o seu estado. Toda vez que a tech é alterada o render é invocado novamente

1.2 Exercício 3

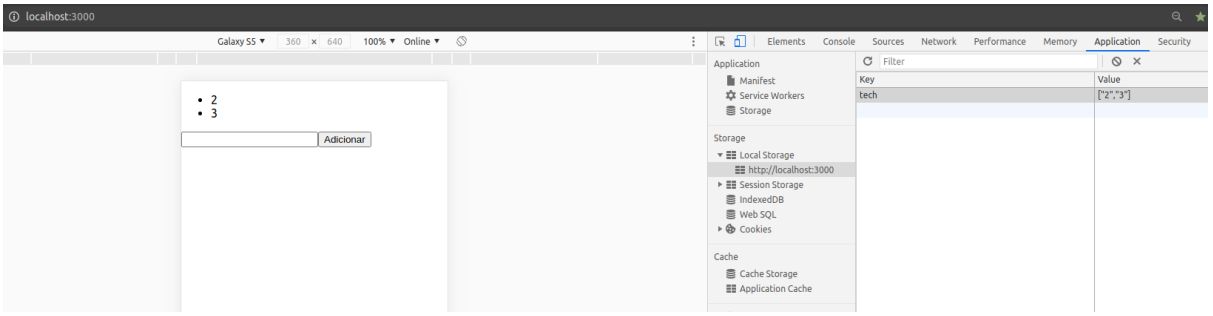
Neste terceiro exercício vamos continuar com o hook useEffect. O useEffect soprõe os ciclos de vida anteriores que usavamos com classes: componentDidMount, componentDidUpdate, componentWillUnmount.

Bem, crie um arquivo chamado Exe3.js. Copie e cole o código do Exe2.js, pois será continuidade dele. Bem, no exercício anterior nós tínhamos criado um estado chamado tech e uma função chamada setTec(). Após, criamos um botão que adicionava um texto fixo (‘Next.js’) ao array do estado.

Agora queremos armazenar cada nova tecnologia digitada pelo usuário no localStorage, sempre que uma variável de estado tiver seu valor alterado (usaremos o hook useEffects). O Local Storage é um espaço reservado pelos browsers para o salvamento de informações localmente. Para isso, teremos que atualizar o estado com o novo valor e chamar o método localStorage.setItem(...) para armazenar esse array com o novo valor.

```
localStorage.setItem('identificador', JSON.stringify(valor));
```

Para verificar se os seus dados estão no Local Storage, clique CTRL + SHIT + I para abrir o modo de desenvolvedor no seu navegador. Após, vá para a aba Application e selecione a opção Local Storage:



Para isso, primeiro será necessário criar um novo hook para armazenar uma nova tecnologia que quero adicionar no Local Storage. Algo assim:

```
const [newTech, setNewTech] = useState("");
```

Em seguida, crie um campo de entrada (tag input do html) que vai armazenar a tecnologia que eu digitar (evento onChange). Algo assim:

```
<input value={newTech} onChange={e =>
setNewTech(e.target.value)} />
```

Quando o usuário clicar no botão de adicionar, chamamos um método que vai repassar para o setTech a nova tecnologia e depois limpar o input passando uma string vazia. Assim, o input recebe um value que é o valor que está sendo digitado, o onChange executa uma função que recebe o evento de digitação no input, que passa para a função de atualização do estado do newTech o valor que está sendo digitado.

Desta forma, nesta atividade prática você criou uma aplicação com React Hooks