Interfaces Web para a Gestão de Dados

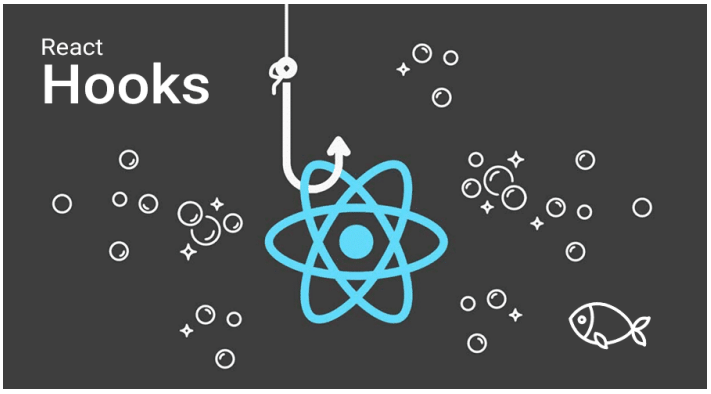
2025/26

REACT – 5ª semana

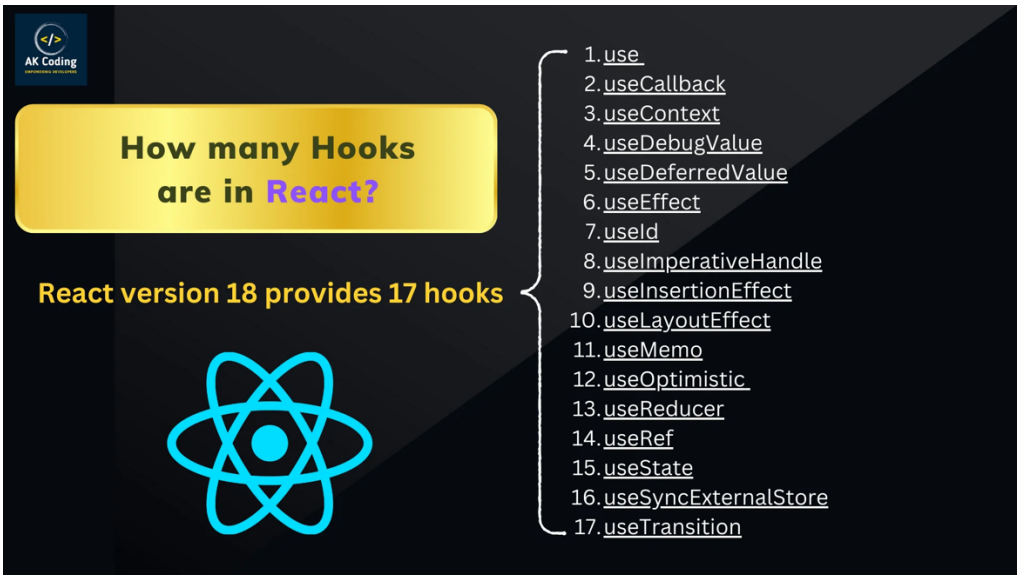
ISCTE Instituto Universitário de Lisboa
Gabinete de Comunicação e Imagem

Desenvolvimento de Frontend em REACT

Índice



Hooks (“ganchos”) permitem que components React tenham acesso a informações para além das variáveis internas de cada componente, como por exemplo State, Effect e Context.



Hooks devem ser invocados dentro de funções React e não podem ser condicionais. A utilização de Hooks é realizada no contexto da programação de funções e não de classes React.

1. useState

useState[1] permite guardar o estado (dados ou propriedades) de um componente.

1.1 Importação de useState

```
import { useState } from "react";
```

1.2 Inicialização de useState

useState é inicializado com o seu estado inicial, que pode ser um valor numérico, uma string, um objeto, um vetor, etc.

useState retorna:

- O seu valor corrente;
- A função que atualiza o seu estado.

[1] https://akcoding.com/react-tutorial/react-hooks/usestate/

```
import { useState } from "react";

function FavoriteColor() {
  const [color, setColor] = useState("");
}
```

Neste exemplo o state `color` é inicializado com `""` e a sua função de atualização de estado é `setColor`.

1.3 Leitura de State

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return <h1>My favorite color is {color}!</h1>
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<FavoriteColor />);
```

1.4 Atualização de State

A atualização de um state é sempre através da função de atualização, nunca diretamente, isto é, `color="red"` não é permitido.

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button
        type="button"
        onClick={() => setColor("blue")}
      >Blue</button>
    </>
  )
}
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<FavoriteColor />);
```

1.5 Utilização de vários State

Com vários useState:

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function Car() {
  const [brand, setBrand] = useState("Ford");
  const [model, setModel] = useState("Mustang");
  const [year, setYear] = useState("1964");
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My {brand}</h1>
      <p>
        is a {color} {model} from {year}.
      </p>
    </>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

Com um objeto:

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function Car() {
  const [car, setCar] = useState({
    brand: "Ford",
    model: "Mustang",
    year: "1964",
    color: "red"
  });

  return (
    <>
      <h1>My {car.brand}</h1>
      <p>
        It is a {car.color} {car.model} from {car.year}.
      </p>
```

```
    </>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

1.6 Utilização com eventos

```
import React, { useState } from 'react';

function MultiStateComponent() {
  const [name, setName] = useState('');
  const [age, setAge] = useState(0);

  return (
    <div>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Name"
      />
      <input
        type="number"
        value={age}
        onChange={(e) => setAge(e.target.value)}
        placeholder="Age"
      />
      <p>
        Name: {name}, Age: {age}
      </p>
    </div>
  );
}
```

1.7 Atualização de objetos e de vetores em State

Quando um state contém um objeto e se pretenda atualizar apenas um atributo do objeto, deve ser utilizado o operador *spread* da seguinte forma:

```
import { useState } from "react";
import ReactDOM from "react-dom/client";

function Car() {
  const [car, setCar] = useState({
```

```
    brand: "Ford",
    model: "Mustang",
    year: "1964",
    color: "red"
  });

  const updateColor = () => {
    setCar(previousState => {
      return { ...previousState, color: "blue" }
    });
  }

  return (
    <>
      <h1>My {car.brand}</h1>
      <p>
        It is a {car.color} {car.model} from {car.year}.
      </p>
      <button
        type="button"
        onClick={updateColor}
      >Blue</button>
    </>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

2. Formulários

2.1 Abertura de um formulário

```
function UmForm() {
  return (
    <form>
      <label>Nome:
        <input type="text" />
      </label>
    </form>
  )
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<UmForm />);
```

2.2 Tratamento dos dados de um formulário

Os dados de um formulário são guardados no state de um componente[2].

As alterações aos dados inseridos num formulário são geridas por eventos.

Exemplo:

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';

function UmForm() {
  const [nome, setNome] = useState("");
  return (
    <form>
      <label>Nome:
        <input
          type="text"
          value={nome}
          onChange={(e) => setNome(e.target.value)}
        />
      </label>
    </form>
  )
}
```

[2] O objeto `state` permite guardar propriedades de um componente. A função `setState` altera o conteúdo do `state` de um componente.

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<UmForm />);
```

2.3 Submissão de um formulário

Submissão com recurso ao atributo onSubmit.

Exemplo com um campo de input do tipo text:

```
import { useState } from 'react';
import ReactDOM from 'react-dom/client';

function UmForm() {
  const [nome, setNome] = useState("");

  const handleSubmit = (event) => {
    // impede o browser de atualizar o form
    event.preventDefault();
    //abre um popup com o valor no campo nome que está no state
    alert('Nome introduzido: ${nome}')
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Nome:
        <input
          type="text"
          value={nome}
          // quando o campo é atualizado guarda o valor
         // no campo nome que está no state
          onChange={(e) => setNome(e.target.value)}
        />
      </label>
      <input type="submit" />
    </form>
  )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<UmForm />);
```

2.4 Tratamento de múltiplos dados e registo no state

Tratamento dos dados do formulário através de um único handler e registo no state.

```
function FormularioEstudante() {

    const [inputs, setInputs] = useState({nome: "", curso: ""});

    const changeHandler = (e) => {
        setInputs({...inputs, [e.target.name]: e.target.value});
    };

    const submitHandler = (e) => {
        e.preventDefault();
        alert("Nome: " + inputs.nome + "\nCurso: " + inputs.curso);
    };

    return (
        <>
            <h2>Exemplo de formulário</h2>
            <form onSubmit={submitHandler}>
                Nome: <input name="nome" type="text" value={inputs.nome}
                       onChange={changeHandler}/>
                <br/>
                 Curso: <input name="curso" type="text"
                          value={inputs.curso}
                           onChange={changeHandler}/>
                 <br/>
                 <input type="submit" value="Submeter"/>
               </form>
             </
  );
}
```

Para exemplos com campos de input do tipo text, ou com campos de input do tipo textarea ou select, ou ainda com vários tipos de campos de input em simultâneo num form, veja em https://www.w3schools.com/REACT/react_forms.asp

3. Router

React, sendo destinado para a programação de SPAs, não inclui um router que possibilite a invocação de várias páginas. Para adicionar várias páginas instale a biblioteca React Router ao seu projeto.

Para a instalação, execute na raiz do projeto:

```
npm i -D react-router-dom
```

Depois crie uma nova diretoria pages em src. Esta diretoria incluirá as páginas React, por exemplo App.js e DadosForm.js .

index.js inclui a definição do Router:

```
(…)
import { BrowserRouter, Routes, Route } from "react-router-dom";
import App from './pages/App';
import DadosForm from './pages/DadosForm';


const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<App />} />
        <Route path="/dadosform" element={<DadosForm />} />
      </Routes>
    </BrowserRouter>
  </React.StrictMode>
);
(…)
```

A componente App tem o caminho “/” e a componente DadosForm tem o caminho "/dadosform".

Depois, qualquer componente pode usar o router para invocar outros componentes. Por exemplo, se a App tem um botão para abrir uma nova página com um formulário através do caminho "/dadosform":

```
 (...)
import { useNavigate } from "react-router-dom";

function App() {
  const navigate = useNavigate();
  return (
    <>
          (...)
          <button onClick={() => navigate("/dadosform")}>
             Formulário
          </button>
          (...)
    </>
  );
}
(...)
```

Para exemplos mais completos sobre React Router consulte https://reactrouter.com/en/main e https://www.w3schools.com/REACT/react_router.asp

4. Memo

A invocação de memo num componente permite que um componente seja invocado na primeira vez que uma página é aberta, e a partir desse momento seja invocado apenas se a sua invocação implicar alterações na página, senão mantém o estado anterior. Assim, a utilização de memo aumenta a performance de uma página composta por componentes React.

A programação de memo é realizada no import e no export do ficheiro do componente.

```
import { memo } from "react";

const UmComponente = () => {
   return (
      <>
         (...)
      </>
   );
};

export default memo(UmComponente);
```

5. Bibliografia

"React Top-Level API", https://legacy.reactjs.org/docs/react-api.html . Acedido em Acedido em 16-04-2024.

"React Tutorial", https://www.w3schools.com/REACT/default.asp . Acedido em 16-04-2024.

Purohit, Rakesh: "A Comprehensive Guide to React Practical Exercises and Coding Challenges", https://www.dhiwise.com/post/a-comprehensive-guide-to-react-practical-exercises-and-coding-challenges . Acedido em 18-04-2024.

Souza, "Using React with Django to create an app: Tutorial" https://blog.logrocket.com/using-react-django-create-app-tutorial/ . Acedido em 24-04-2023.

Singhal, "How to create a REST API with Django REST framework" https://blog.logrocket.com/django-rest-framework-create-api/ . Acedido em 24-04-2023.

Parker, "How to NPM Start for React Tutorial Project" https://www.pluralsight.com/guides/npm-start-for-react-tutorial-project . Acedido em 24-04-2023.

Basques and Emelianova, "Simulate mobile devices with Device Mode" https://developer.chrome.com/docs/devtools/device-mode/ . Acedido em 24-04-2023.

Wasson, Mike, "ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET", MSDN Magazine, Volume 28 Numero 11, Novembro 2013.

Penmetsa, Chaitanya, "Representational State Transfer (REST) and Design Principles", Medium, Janeiro 2024, https://medium.com/codenx/representational-state-transfer-rest-and-design-principles-98640faa1ab4 . Acedido em 16-02-2025.

"Support of Cross-Origin Resource Sharing (CORS)", 4DBlog, Agosto 2020, https://blog.4d.com/support-of-cross-origin-resource-sharing-cors/ , acedido em 16-02-2025.