front-end

prof. Lucas Ferreira

engenharia de software

engenharia de computação

Frameworks de Interação:

React.js

React.js

Sendo o terceiro e última biblioteca de JS "avançada" com potencial de mercado que iremos ver este semestre, o React.js é uma das ferramentas mais versáteis que você pode estudar nos dias de hoje.

Criado (**2013**) e mantido pelo Facebook, Instagram e uma comunidade de desenvolvedores com foco em open-source, a lib **React** se autodenomina *"Uma biblioteca JavaScript para criar interfaces de usuário"*.

Define-se como uma biblioteca JavaScript declarativa, eficiente e flexível para criar interfaces com o usuário.

Ele permite compor UIs complexas a partir de pequenos e isolados códigos chamados “componentes”.

React.js

Sendo o terceiro e última biblioteca de JS "avançada" com potencial de mercado que iremos ver este semestre, o React.js é uma das ferramentas mais versáteis que você pode estudar nos dias de hoje.

Por tanto, a lib **React** objetiva facilitar a criação de UIs interativas baseadas em componentes encapsulados que gerenciam seu próprio estado, focando na estruturação de *views* complexas com fácil acesso a controle de estado.

A lógica do componente é escrita em JavaScript e não em templates *(como no Angular ou Vue.js)*.

A lib **React**, sozinha, é destinada principalmente ao desenvolvimento de interfaces de páginas web.

Também pode ser renderizado no servidor, usando Node, e ser usado para criar aplicações mobile, através do **React Native** *(este sim um framework)*.

Recomendo um breve passeio pela documentação do React:
https://legacy.reactjs.org/docs/getting-started.html
https://react.dev/learn

Iniciando em React

Na teoria não é necessário o uso de ***Node.js***, ***Vite.js***, ***Webpack***, ***Babel***, ***Parcel***, ***Rollup*** e etc para criar um projeto em React, é possível iniciar diretamente de um HTML básico carregando os seguintes scripts em seu front-end:

```
<script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js" crossorigin></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

Iniciando em React

Ainda nesse setup "***precário***" apenas para os primeiros testes, deveríamos criar/usar uma tag script "especial" para que a sintaxe do React (JSX) seja aceita pela página em questão:

```
<div id="root"></div>
<script type="text/babel">
  ReactDOM.render(
    <h1>Hello, world!</h1>,
    document.getElementById('root')
  );
</script>
```

Nesse exemplo "para brincar" percebam que o script do seu código React deve possuir o atributo ***type="text/babel"***.

Iniciando em React

Porém, entretanto, todavia para que seu projeto ganhe corpo e tenha performance de produção será necessário utilizar todas as ferramentas de desenvolvimento front-end populares de mercado, ferramentas estas que usam muito Node.js e "seus amiguinhos" como Webpack e outros.

Uma outra questão que iremos explorar mais a frente, é o ***"novo caminho de ferramental"*** que o React vêm buscando, existiu um momento que o React se focava muito e "apenas" em estar atuando de forma performática no navegador do usuário (100% front-end), mas como o avanço dos paradigmas e expansão no campo de uso da ferramenta o "novo caminho" trás ao React também a possibilidade de trabalhar no servidor, a ideia é que o JavaScript rode de forma ampla nos dois lados da equação (back e front) trazendo assim mais velocidade no carregamento de páginas e melhorias de SEO.

Iniciando em React

Enquanto não vemos essas ferramentas mais complexas que operam React até no back-end, iremos iniciar com opções menos complexas, com foco semelhante ao que vimos em Angular e Vue.js até aqui.

Recomendo neste momento o uso de "projetos facilitadores" como este outro projeto do Facebook chamado **Create React App** *(ou CRA para os íntimos)*:
https://create-react-app.dev/

Também não podemos esquecer da minha opção preferida o Vite.js criando templates de React:
https://vitejs.dev/guide/

Iniciando em React

Outra possibilidade, para um *"light test"* de código React é usar o esplêndido editor de projetos on-line **CodeSandbox** que oferece todo um ambiente complexo de desenvolvimento React rodando diretamente de seu navegador:

https://codesandbox.io/

Ou outras opções como:

https://stackblitz.com/

https://replit.com/

Criando um projeto com React.js

Iremos iniciar com algum teste mais simples e on-line já que teremos mais tempo para explorar o React.js em sala de aula. Porém na sequência para iniciarmos um novo projeto iremos por enquanto voltar ao **Vite.js**.

Para criar um novo projeto do tipo "***React***" usando o Vite.js:

```
# npm 7+; a template também pode ser "react-ts" se preferir TypeScript
npm create vite@latest my-react-app -- --template react

# yarn; a template também pode ser "react-ts" se preferir TypeScript
yarn create vite my-react-app --template react
```

Criando um projeto com React.js

A nova pasta terá o nome do projeto e num primeiro momento precisamos entrar dentro da mesma e instalar as dependências:

```
cd nome-do-projeto
npm install
```

Depois disso quando precisarmos rodar o projeto em modo desenvolvimento, é exatamente como vimos na nossa primeira experiência com ***Vite.js***:

```
npm run dev
```

Criando um projeto com React.js

Nosso projeto criado completamente focado em **React** com os comandos anteriores *(para Vite.js)*, já terá uma base instalada e rodando de React.js pronta para ser trabalhada. Vale observar que o conceito aqui é mantermos a ideia de "SPA".

O único arquivo .html principal *(index.html)* serve apenas para carregar o script principal e abrigar toda a nossa aplicação dentro de uma div primária, normalmente assim:

```
<div id="root"></div>
<script type="module" src="/src/main.jsx"></script>
```

Daqui em diante já podemos editar nossos arquivos **JavaScript** *(ou TypeScript)* e ir criando nossos componentes.

Componentes em React.js - *Básico do Básico*

Após *(seja lá qual for a forma)* estarmos com a lib do React "carregada" em nosso projeto, as primeiras ações provavelmente serão:

- Criar algum componente *(não importa a complexidade ou tamanho, tudo é componente)*
- Renderizar o mesmo *(em tela OU para outro formato desejado)*
- Adicionar algo de variáveis e controle de estado

Componentes em React.js

Antes de começarmos a "brincadeira" de verdade vale uma observação sobre como são estruturados os componentes em React.

Em React não temos distinção entre "tipos de arquivo", diferente do Angular não temos um arquivo separado para lógica e outro .html para a template. Diferente do Vue.js não existe nenhum arquivo "especial" com extensão mágica *(tipo um .react)*.

Os componentes de React são todos escritos 100% em JavaScript *(ou TypeScript)* vai tudo nos arquivos .js/.ts *(também podemos usar as extensões .jsx/.tsx)*.

Sendo que o primeiro conceito que vamos explorar é a ideia de que componentes em React nada mais são **funções de JavaScript que "devolvem/retornam" HTML** para a tela do navegador.

Componentes em React.js

Com esse conceito de ***HTML no meio de JavaScript*** *(como se isso fosse normal)*, qualquer um dos códigos ao lado pode ser considerado um "*componente em React*".

```
const H1 = (
    <div className="header">
        <h1>Hello, world!</h1>
    </div>
);

const H2 = () => <h2>Hello, world!</h2>;

function Title({children}) {
    return (
        <div className="panel">
            <h3>{children}</h3>
        </div>
    );
}

class Heading extends React.Component {
    render() {
        return <h4>{this.props.children}</h4>;
    }
}
```

E como funciona a "sintaxe" do React?

É um bom momento para observar que o React em si não possui sintaxe de templates "especiais". Não existe *ngFor/v-for*, não existe *v-if/ng-if*, nem coisas como *v-model* e *ngModel*. O HTML do React na teoria é mesmo HTML que conhecemos, sem "super poderes". Tem uma ou outra *diferencinha* por causa de palavras reservadas na linguagem JS *(como **class** por exemplo)* mas de resto não há nada de novo sem ser o fato de que o HTML é escrito dentro do JS como se fosse algo "normal" *(o que não é)*.

Um elemento curioso para entender o que o React faz por trás dos panos em relação a sua sintaxe que "mistura HTML com JavaScript" ***(chamada tecnicamente de JSX)*** é perceber que no final de tudo o React irá transformar o código que fazemos em algo mais compreensível ao navegador *(já que navegadores por padrão não suportam JSX)*.

Como funciona a "sintaxe" JSX

Cada "componente" criado em React escrito com a sintaxe comum de JSX em seu projeto:

```
<div className="shopping-list">
  <h1>Lista de compras para {props.name}</h1>
</div>
```

Irá ser convertida em tempo de execução *(e também em produção)* para algo nesse sentido:

```
React.createElement('div', { className: 'shopping-list' },
  React.createElement('h1', null, `Lista de compras para ${props.name}`),
);
```

Como funciona a "sintaxe" JSX

Ou seja, quando estamos com o devido *setup* "armado", nosso projeto por meio do Babel (https://babeljs.io/) irá no final das contas transformar tudo que escrevemos de JSX em funções compatíveis de JavaScript que o navegador poderá rodar sem maiores dificuldades.

Por tanto a ideia de implementar o uso de JSX na lib React nada mais foi que uma decisão de DX *(facilitar a vida para o desenvolver)* e não haver com "ganhos de performance".

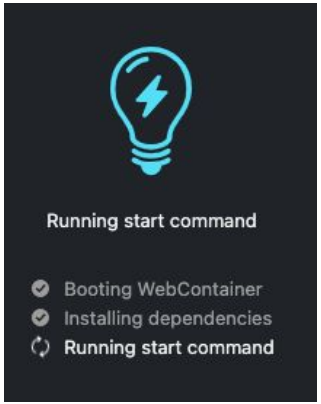
SO...
LET'S CODE!

Vamos testar isso juntos?

Primeiro de tudo quem tiver a oportunidade *(e vontade)* liguem seus computadores e acessem o link abaixo no ***Google Chrome*** de cada máquina:

https://vite.new/react

É um processo relativamente rápido, iremos "levantar" um ambiente on-line para brincarmos de React.



Vamos testar isso juntos?

Após o "boot" podemos dar uma fuçada nesses arquivos criados, serão os mesmos (do modelo) arquivos que encontraremos em nossa máquina caso iniciemos um novo projeto usando ***Vite.js*** com a template React.

Por hora nosso diretório de trabalho está dentro de **`src/`**.

Na sequência podemos ir em "**src/App.jsx**" e dar uma limpada nesse arquivo modelo aí.

src/App.jsx - Versão limpa!

```jsx
import './App.css'

function App() {
  return (
    <div>
      <p>Olá UniSATC!</p>
    </div>
  )
}

export default App
```

Vamos testar isso juntos?

No código sugerido no slide anterior, mantive o ***import*** ao "App.css" apenas para continuar a formatação que já veio ali, mas na teoria nem precisaríamos disso.

Agora vamos explorar um pouco essa relação de criação de funções que "devolvem HTML" para a tela e na sequência partiremos para o tópico de controle de estado / variáveis em React.

Componente com "estado"

Em React desde suas primeiras versões, temos o conceito de ***componentes funcionais puros*** e também ***componentes funcionais com estado próprio***.

Aqui trazemos o conceito de "estado" a toda e qualquer variável de memória que possa interagir com a tela do usuário. Para criar essas "variáveis" mágicas o React estruturou um conceito chamado de "Hooks" muito semelhante ao conceito de "ref" ou "reactive" do Vue.js versão 3.0.

Componente com "estado"

```jsx
import { useState } from 'react';

function Contador() {
  const [count, setCount] = useState(1);
  const incrementarCount = () => setCount(count + 1);

  return (
    <div className="meu-contador">
      <h2>
        Contagem <em>{count}</em>
      </h2>
      <button type="button" onClick={() => incrementarCount()}>
        Incrementar
      </button>
    </div>
  );
}
```

Componente com "estado"

O código da tela anterior inicia um "novo componente" em nossa aplicação, como eu já disse componentes de React nada mais são que funções que "devolvem HTML para a tela".

Essa função chamada aqui de "***Contador***" pode ser invocada para tela como se fosse uma "tag de HTML customizada" algo mais ou menos assim:

```
function App() {
  return (
    <div>
      <Contador />
    </div>
  );
}
```

Componente com "estado"

Para encerrar, a função-componente Contador possui uma variável de estada assinalada pelo método "useState".

Esse método ***useState*** *(assim como o ref do Vue.js)* é uma parte muito importante do React.js moderno, ele faz parte do grupo de "hooks" que vem de fábrica com o React.

Basicamente, se você precisa manter uma variável para armazenar algo em memória e esse "algo armazenado" pode impactar na tela que o usuário vê, você precisará encapsular essa variável dentro de um "useState".

Aí a partir da aula que vem iremos explorar muito mais esse conceito de ***useState*** e seus outros amiguinhos "hooks".

Nossa próxima aula...

Nossa 1ª prova/avaliação

Semana que vem teremos nossa **1ª avaliação**, normal aquela valendo de 0 à 10.

Será uma avaliação executada **100% de forma presencial**, porém em *formato digital* *(vocês usarão os computadores de sala de aula para responder a avaliação)*.

Peço que todos cheguem no horário de costume *(por volta das 19h)* antes de iniciarmos a prova **irei falar uns 10 minutinhos sobre o nosso trabalho de ABP** e depois disso libero vocês para a execução da prova.

O conteúdo da prova será principalmente o início do semestre, o que vimos da **aula 02 até a 06 (revisem os slides todos)** e um ***pouquinho de nada*** sobre Frameworks.

obrigado 🚀