Desenvolvimento para Dispositivos Móveis

Aula 01 - Fundamentos do React (Parte 1)

Material Didático do Instituto Metrópole Digital - IMD

Termo de uso

Os materiais didáticos aqui disponibilizados estão licenciados através de Creative Commons **Atribuição-SemDerivações-SemDerivados CC BY-NC-ND**. Você possui a permissão para realizar o download e compartilhar, desde que atribua os créditos do autor. Não poderá alterá-los e nem utiliza-los para fins comerciais.

Atribuição-SemDerivações-SemDerivados

CC BY-NC-ND

https://creativecommons.org/licenses/by-nc-nd/4.0/

Apresentação

Nesta aula, iniciaremos os estudos sobre o *React*, um importante fundamento para entendermos mais tarde o *React Native*.

Objetivos

- Conhecer conceitos do *react*
- Conhecer como funciona a biblioteca *react*
- Conhecer como criar um projeto *react*
- Conhecer como criar um componente

O que é e porque React?

Link do video da aula: https://youtu.be/ktDIoxF2x68

Antes de mais nada, é preciso entender que o *React* nada mais é do que uma biblioteca *Javascript* feita para facilitar a criação de *interface* com o usuário.

Vantagens do React

O *react* é uma tecnologia moderna, robusta e que permite criar essa *interface* com o usuário utilizando os componentes, facilitando o reuso do código da *interface*. Além de ser muito popular e ser a base tecnológica do *React Native*.

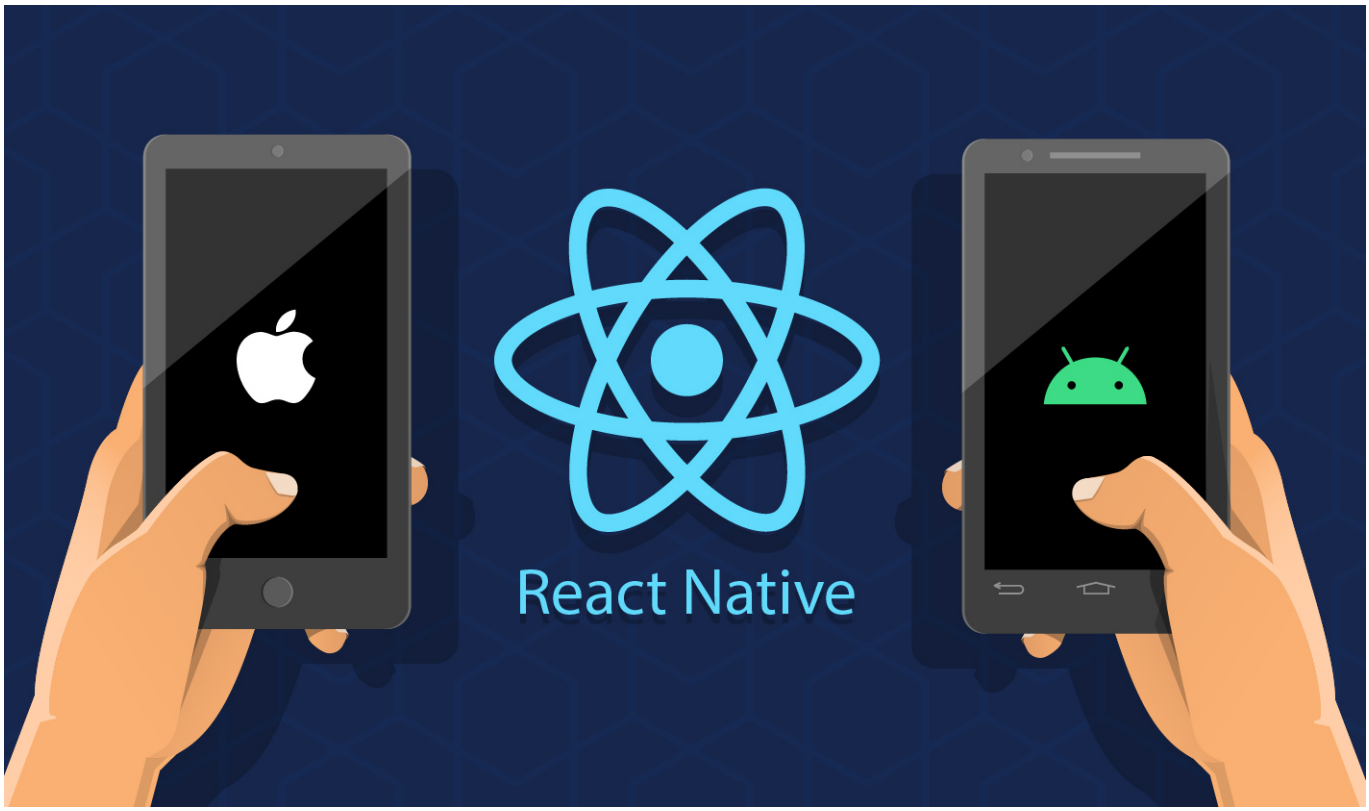

Imagem 1: React Native

Entendendo o *DOM*

Quando criamos *HTML* e *CSS*, estamos criando uma *interface* através da linguagem *HTML*. Porém, por trás, o *browser* mantém o DOM, que é a representação dessa árvore de componentes organizada de forma hierárquica. Então, o *Document Object Model* é uma representação da *interface* do usuário.

E uma coisa interessante é que o *browser* permite que essa parte seja alterada através do *javascript*, refletindo essa alteração na tela do usuário.

Single Page Application

Para trazer essa parte dinâmica, o *React* precisa alterar o *DOM*, com base nos estados e propriedades, armazenados no sistema. Caso essas informações mudem, o *React* irá "reagir", alterando o *DOM* e trazendo a mudança da *interface*.

Como o *React* faz essas alterações via *javascript*, a página permanece sem a sensação de carregamento, fazendo com que o usuário tenha essa mudança na tela, mas sem o *reload* da página. Atualmente, essa característica é comum nos sistemas *web*.

Esse conceito é denominado *Single Page Application (SPA)*.

Virtual *DOM*

Como já foi dito, o *React* altera o *DOM* à medida que os dados mudam. Porém, para fazer isso, é utilizado o conceito do Virtual *DOM*, que nada mais é que uma cópia do *DOM* que o *browser* tem.

A ideia é que atualizar diretamente o *DOM* pode não ser tão performático e o *React* precisa ser cirúrgico nessas alterações. Então, primeiro as mudanças são carregadas nesse Virtual *DOM*, sendo verificada qual a diferença dele para o *DOM* do *browser*. Feita essa operação, o *React* vai no *DOM* do *browser* e atualiza somente o componente necessário.

Create React Application e JSX

Link do video da aula: https://youtu.be/dzdQQtqfd2w

Um projeto *react* tem certa complexidade em sua criação, isso porque existem muitas dependências e configurações que precisam ser feitas e manualmente; isso, além de trabalhoso, levaria bastante tempo.

Criando um projeto *React*

Para contornar esse problema existe uma aplicação chamada *create-react-app*. Esse facilitador cria um projeto com toda essa configuração inicial adiantada.

Então, em seu terminal, execute o comando abaixo:

```
npx create-react-app aula01
```

Limpando o projeto

Após criado, o projeto terá muitos arquivos e para começar, iremos deletaremos alguns deles. Então, na pasta *src* apague todos os arquivos, exceto o '*index.js*'. Além disso, é preciso limpar o código do arquivo que restou, retirando as importações dos arquivos deletados.

JSX

É uma extensão da linguagem *javascript*, que permite criarmos códigos *javascript* com a sensação de estarmos criando código *HTML*. Isso possibilita que no meio da sintaxe do *HTML* seja possível injetar códigos *javascript*.

Renderizando o primeiro componente

Para renderizar um componente utilizaremos os benefícios do JSX. Acompanhando a aula passo a passo, crie o código exibido e ao final ele deve ficar como abaixo:

```
import React from 'react';
import ReactDOM from 'react-dom';

function tick() {
  const element = <h1>Hello World {Math.random()} </h1>

  ReactDOM.render(
    element,
    document.getElementById('root')
  );

}

setInterval(tick, 1000);
```

Criando primeiro componente

Link do video da aula: https://youtu.be/dPCXY3Eu-Q4

Nesta aula veremos como criar componentes *React* de uma maneira mais adequada.

Criando um componente *React*

Um componente no *React* nada mais é que uma função que retorna um JSX. Esses componentes podem receber parâmetros, chamados de propriedades.

Veja abaixo um exemplo:

```
function Welcome(props) {
  return <h1>Hello {props.name} </h1>
}

ReactDOM.render(
  <Welcome name="Paulo" />,
  document.getElementById('root')
);
```

Ao renderizar, será exibido em tela: "Hello Paulo".

Reutilização do componente

O intuito de criar um componente é poder reutilizá-lo. Porém, se tentarmos chamar essa mesma função novamente, mesmo que em formato de componente, nosso programa apresentará um erro.

Isso acontece porque o *render* espera um único componente. Para resolver isso, é criado um componente que reúne todos os outros.

Acompanhando a aula, crie o componente *App* e ao final seu o código deve ficar como abaixo:

```
import React from 'react';
import ReactDOM from 'react-dom';

function Welcome(props) {
  return <h1>Hello {props.name} </h1>
}
```

```
function App() {
  return (
    <div>
      <Welcome name="Paulo" />
      <Welcome name="Maria" />
    </div>
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Resumo

Nesta aula, foi feita uma introdução à biblioteca *react*, mostrando seus conceitos e funcionalidades. Além disso, foi visto como criar um projetos e componentes no *react*.