**SESSAO 2**

Aula 07

1. Iremos utilizar a ferramenta creat-react-app para realizarmos a configuração de nossos projetos.
2. Criar o projeto com : npx create-react-app projeto1 e será criado a pasta projeto1 com os arquivos do projeto. Caso queira instalar os arquivos sem criar a pasta, basta digitar : npx create-react-app . .

Aula 08

1. Vamos olhar o package.json as dependências instaladas, olhando as pastas e arquivos do projeto.
2. Teste o projeto inicializado digite : npm start

Aula 09

1. Execute o npm run build que será criado uma pasta no projeto na versão de deploy.

Aula 11

1. O react não precisa estar em toda a página.

Aula 12

1. Componentes funcionais
2. import logo from './logo.svg';
3. import './App.css';
4. function App() {
5. return (
6. <div *className*="App">
7. <header *className*="App-header">
8. <img *src*={logo} *className*="App-logo" *alt*="logo" />
9. <p>
10. Edit <code>src/App.js</code> and save to reload.
11. </p>
12. <a
13. *className*="App-link"
14. *href*="https://reactjs.org"
15. *target*="\_blank"
16. *rel*="noopener noreferrer"
17. >
18. Learn React
19. </a>
20. </header>
21. </div>
22. );
23. }
24. export default App;
25. e componentes de classe.
26. import logo from './logo.svg';
27. import './App.css';
28. import {Component} from 'react'
29. class App extends *Component*{
30. render(){
31. return (
32. <div *className*="App">
33. <header *className*="App-header">
34. <img *src*={logo} *className*="App-logo" *alt*="logo" />
35. <p>
36. Edit <code>src/App.js</code> and save to reload.
37. </p>
38. <a
39. *className*="App-link"
40. *href*="https://reactjs.org"
41. *target*="\_blank"
42. *rel*="noopener noreferrer"
43. >
44. Learn React
45. </a>
46. </header>
47. </div>
48. );
49. }
50. }
51. export default App;

Aula 13

1. Estado de componentes.
2. São basicamente dados que o componente utiliza, mas toda vez que o estado mudar dizemos para o react para renderizar novamente o componente.
3. Vamos criar um construct, no react o construct sempre recebe props como parâmetro.
4. Devemos chamar o super() pois estamos estendendo da classe mãe Component.
5. Utilizamos o this.state para passar o state do componente variáveis ou funções que iremos utilizar na página. Por exemplo nos criamos uma variável name que recebe um valor. Dentro do render() nós utilizamos um destruct para pegar a variável salva dentro do state e esse valor é passado para uma variável dentro do html.
6. Utilizamos o método onClick e passamos uma arrow function para ele de maneira que quando clicarmos na variável, irá aparecer no console <p> Clicado;
7. constructor(*props*){
8. *super*(props);
9. *this*.state = {
10. name: 'Claudisnei Bello'
11. };
12. }
13. render(){
14. //const name = this.state.name;
15. const {name} = *this*.state;
16. return (
17. <div *className*="App">
18. <header *className*="App-header">
19. <img *src*={logo} *className*="App-logo" *alt*="logo" />
20. <p *onClick*={()=>{console.log('<p> Clicado') }}>
21. {name};
22. </p>
23. Como estamos dentro de uma classe, é como criarmos um método dentro da classe para executar uma ação.
24. Vamos criar então a função HandlePClick e vamos passar o mesmo conteúdo que estava dentro de onClick.
25. Agora basta apagarmos o conteúdo de onClick e chamarmos o método da classe com this.HandlePClick. isso funcionará da mesma maneira.
26. HandlePClick(){
27. console.log('<p> Clicado');
28. }
29. render(){
30. //const name = this.state.name;
31. const {name} = *this*.state;
32. return (
33. <div *className*="App">
34. <header *className*="App-header">
35. <img *src*={logo} *className*="App-logo" *alt*="logo" />
36. <p *onClick*={*this*.HandlePClick}>
37. {name};
38. </p>
39. Para podermos utilizar as variáveis que estão sendo declaradas no construtor, dentro dos métodos, devemos byndar o método no construtor
40. constructor(*props*){
41. *super*(props);
42. *this*.HandlePClick = *this*.HandlePClick.bind(*this*);
43. *this*.state = {
44. name: 'Claudisnei Bello'
45. };
46. }
47. HandlePClick(){
48. const {name} = *this*.state;
49. console.log(`<p> clicado ${name}`);
50. Para alteramos o valor das variáveis no state temos que utilizar o método this.setState{}.
51. HandlePClick(){
52. //const {name} = this.state;
53. // console.log(`<p> clicado ${name}`);
54. *this*.setState({name: 'Débora Sobrinho'});
55. }
56. Vamos criar outro método porém agora como arrow function, pois desta forma não precisamos byndar o método no construtor.
57. Iremos criar um método para que ao clicarmos no link da página, o link seja alterado para outro.
58. Porém devemos passar para função o parâmetro event e dentro da função devemos chamar o event.preventDefault(), desta forma ao clicarmos no link ele não executara mais o que está na página, mas irá executar o que esta dentro da função.
59. Devemos pegar o counter no destruct do this.state e passar essar variável em qualquer lugar dentro da página.
60. HandleAClick=(*event*)=>{
61. event.preventDefault();
62. const {counter} = *this*.state;
63. *this*.setState({counter : counter + 1});
64. }
65. render(){
66. //const name = this.state.name;
67. const {name, counter} = *this*.state;
68. return (
69. <div *className*="App">
70. <header *className*="App-header">
71. <img *src*={logo} *className*="App-logo" *alt*="logo" />
72. <p *onClick*={*this*.HandlePClick}>
73. {name}{counter};
74. </p>
75. <a
76. Para melhorar podemos eliminar a utilização do construct com a class Fields.
77. Ela nos permite criar atributos sem precisar utilizar o construtor.
78. import logo from './logo.svg';
79. import './App.css';
80. import {Component} from 'react'
81. class App extends *Component*{
82. // constructor(props){
83. //  super(props);
84. //   this.HandlePClick = this.HandlePClick.bind(this);
85. //  this.state = {
86. state ={
87. name: 'Claudisnei Bello',
88. counter: 0
89. };
91. HandlePClick=()=>{
92. //const {name} = this.state;
93. // console.log(`<p> clicado ${name}`);
94. *this*.setState({name: 'Débora Sobrinho'});
95. }
96. HandleAClick=(*event*)=>{
97. event.preventDefault();
98. const {counter} = *this*.state;
99. *this*.setState({counter : counter + 1});
100. }
101. render(){
102. //const name = this.state.name;
103. const {name, counter} = *this*.state;
104. return (
105. <div *className*="App">
106. <header *className*="App-header">
107. <img *src*={logo} *className*="App-logo" *alt*="logo" />
108. <p *onClick*={*this*.HandlePClick}>
109. {name} {counter};
110. </p>
111. <a
112. *onClick*={*this*.HandleAClick}
113. *className*="App-link"
114. *href*="https://reactjs.org"
115. *target*="\_blank"
116. *rel*="noopener noreferrer"
117. >
118. Learn React
119. </a>
120. </header>
121. </div>
122. );
123. }
124. }
125. export default App;

Aula 15 – Array dentro do State

1. Vamos fazer algumas alterações, iremos apagar os métodos que foram criados dentro da classe App e todo conteúdo que está dentro da div App.
2. mport './App.css';
3. import {Component} from 'react'
4. class App extends *Component*{
6. state ={
7. name: 'Claudisnei Bello',
8. counter: 0
9. };
11. render(){
12. const {name, counter} = *this*.state;
13. return (
14. <div *className*="App">
16. </div>
17. );
18. }
19. }
20. Vamos criar um array dentro do objeto state do Component.
21. Vamos passar alguns dados para o array e em seguinda no Render() vamos criar uma variável post pegando essa informação do state.
22. import './App.css';
23. import {Component} from 'react'
24. class App extends *Component*{
26. state ={
27. posts:[
28. {
29. id: 1,
30. title: 'Título 1',
31. body: 'Corpo 1'
32. },
33. {
34. id: 2,
35. title: 'Título 2',
36. body: 'Corpo 2'
37. },
38. {
39. id: 3,
40. title: 'Título 3',
41. body: 'Corpo 3'
42. }
44. ]
45. };
47. render(){
48. const {posts} = *this*.state;
49. return (
50. <div *className*="App">
52. </div>
53. );
54. }
55. }
56. export default App;
57. Iremos utilizar o método map do Array.prototype.map() para retornar o que quisermos dentro de um array em Javascript.
58. Então dentro do root App vamos utilizar o método map de arrays na variável post passando como parâmetro post=> <h1>{post.title}</h1>} isso irá renderizar na tela todos os títulos que estão no array posts. Mas irá ocorrer um erro no console sobre Key porque não estamos fornecendo uma chave única para os itens do array, para corrigir este problema, basta adicionarmos no <h1 key={post.id}> e o erro será corrigido.
59. render(){
60. const {posts} = *this*.state;
61. return (
62. <div *className*="App">
63. {posts.map(*post*=> <h1 *key*={post.id}>{post.title}</h1>)}
64. </div>
65. );
66. }
67. Se quisermos mostrar na tela mais de uma linha de informação basta inserir todo <h1> dentro de parênteses.
68. Porem para funcionar temos que adicionar todas as linhas dentro de um único root ou seja tudo terá que estar dentro de uma div e o key que estava no h1 deverá ser colocado dentro da div.

Aula 16 – Life cicle methods part 1

1. Se quisermos que ocorra alguma coisa com o componente assim que ele for montado, podemos utilizar o método de ciclo de vida, componentDidMount(), é um método como o render()
2. Vamos adicionar um this.setState dentro deste método e dentro do this.setState nós iremos adicionar o array posts com algumas alterações nos dados para podermos visualizar as alterações ocorrendo na tela assim que o componente for montado irá ocorrer a alteração dos dados do state.
3. Iremos adicionar o this.setState dentro de um setTimeOut() passando um tempo de 5 segundos, ou seja assim que passa 5 segundos a alteração ocorrerá.
4. Iremos criar dentro do state uma variável chamada counter com valor 0.
5. import './App.css';
6. import {Component} from 'react'
7. class App extends *Component*{
9. state ={
10. counter: 0,
11. posts:[
12. {
13. id: 1,
14. title: 'Título 1',
15. body: 'Corpo 1'
16. },
17. {
18. id: 2,
19. title: 'Título 2',
20. body: 'Corpo 2'
21. },
22. {
23. id: 3,
24. title: 'Título 3',
25. body: 'Corpo 3'
26. }
28. ]
29. };
30. //variavel criada para zerar o timeout
31. timeoutUpdate = null;
32. //esse é um life cicle metods
33. componentDidMount(){
34. *this*.handleTimeout();
36. }
37. //é um life cicle que recebe o estado anterior ou props states
38. componentDidUpdate(){
39. *this*.handleTimeout();
40. }
41. //para apagar o lixo e não dar erro no navegador
42. componentWillUnmount(){
43. //zera o time para dar erro na página quando ocorrer alteraçoes
44. clearTimeout(*this*.timeoutUpdate);
45. }
46. //função criada para atualizar o state do componente
47. handleTimeout=()=>{
48. const {posts, counter} = *this*.state;
49. posts[0].title = 'o título mudou';
50. *this*.timeoutUpdate = setTimeout(() => {
51. *this*.setState({
52. posts,
53. counter: counter + 1
54. })
55. }, 1000);
56. }
58. render(){
59. const {posts, counter} = *this*.state;
60. return (
61. <div *className*="App">
62. <h1>{counter}</h1>
63. {posts.map(*post*=> (
64. <div *key*={post.id}>
65. <h1>{post.title}</h1>
66. <p>{post.body}</p>
67. </div>
68. ))}
69. </div>
70. );
71. }
72. }
73. export default App;