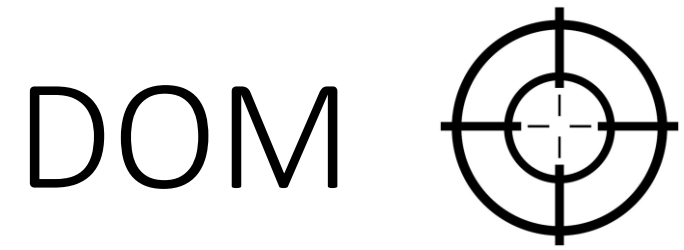


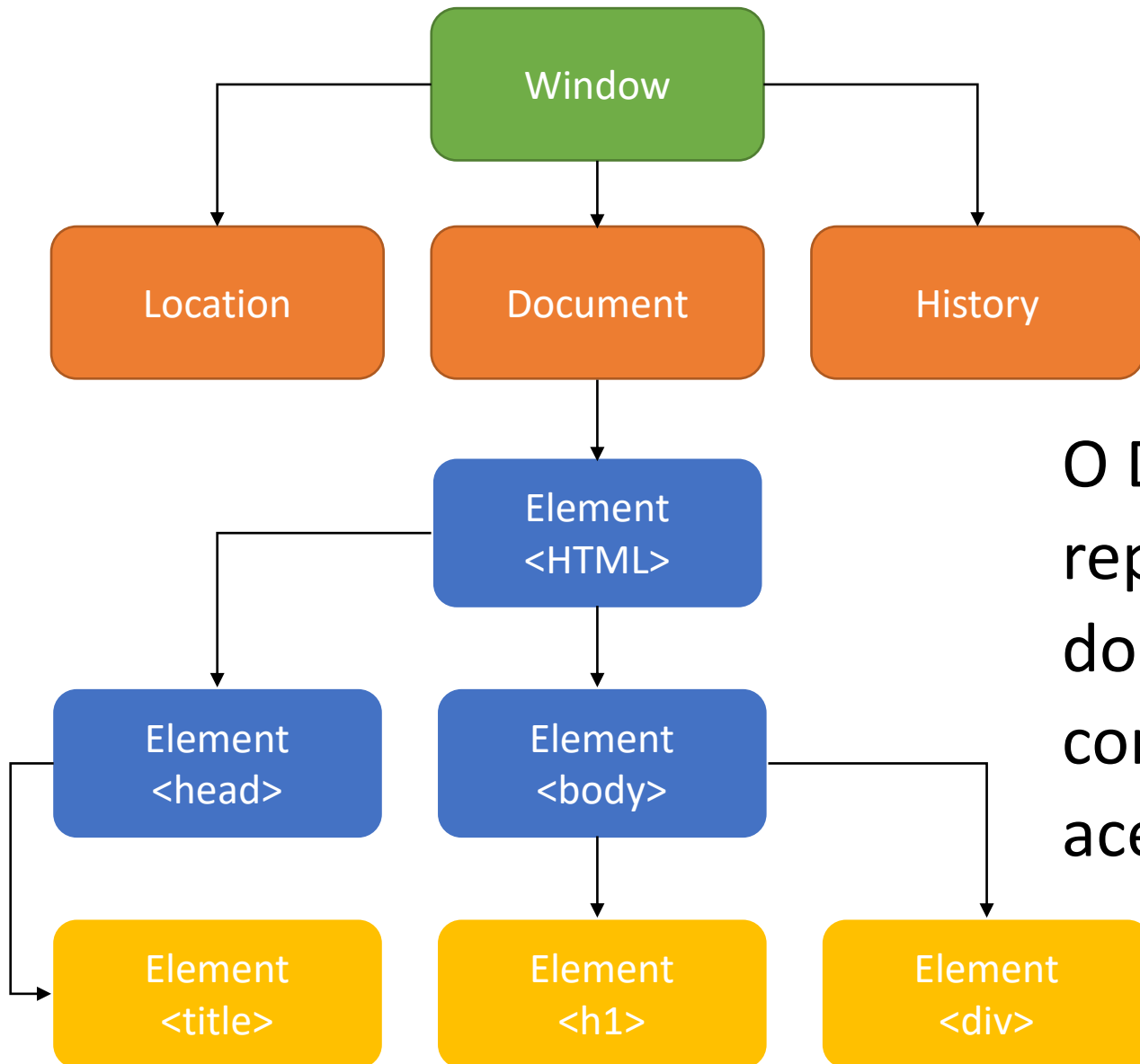
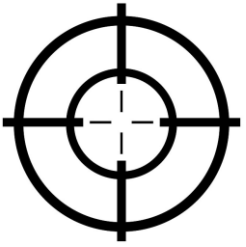
Eventos,  
renderização  
condicional e  
formulários.



Quando uma página da web é carregada pelo navegador, este cria um DOM (Document Object Model) da página.

O DOM é uma convenção multiplataforma e independente de linguagem de programação, criada pela entidade World Wide Web Consortium (W3C), para representação e interação com objetos em documentos HTML, XHTML e XML.

# DOM



O DOM é um objeto, que faz uma representação estruturada do documento HTML e define meios de como essa estrutura pode ser acessada.

# Eventos



Eventos são ações ou ocorrências que acontecem durante a execução de um software. Estas ocorrências podem ser provocadas por atitudes de um usuário e é comum utilizá-las para disparar algum mecanismo de resposta/processamento, que depende do tipo do evento ocorrido.

Um exemplo fácil de entender, ocorre quando um usuário clica em um botão, de uma página web. Este clique pode resultar em uma ação, como um redirecionamento ou submissão de dados.

# Eventos React

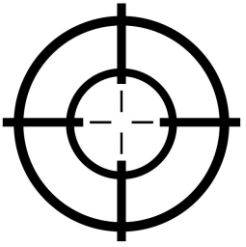


O DOM possui métodos, que são funções JavaScript para manipular elementos no documento HTML e manipular eventos.

Através dos métodos pode-se acessar ou atribuir valores para um elemento no HTML. Também pode-se disparar ou aplicar tratamento a um evento.

Importante! A biblioteca react usa os eventos do DOM, de modo muito semelhante a uma página comum, com poucas mudanças no nome/sintaxe das funções.

# Eventos React



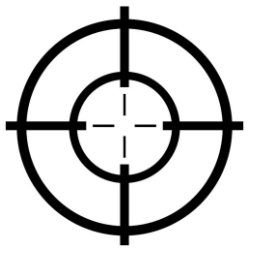
```
ReactDOM.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

```
function App() {  
  function verificaClique(){  
    console.log('Você clicou em: Clique aqui!');  
  }  
  return (  
    <div>  
      <button onClick={verificaClique}>Clique aqui!</button>  
    </div>  
  );  
}  
export default App;
```

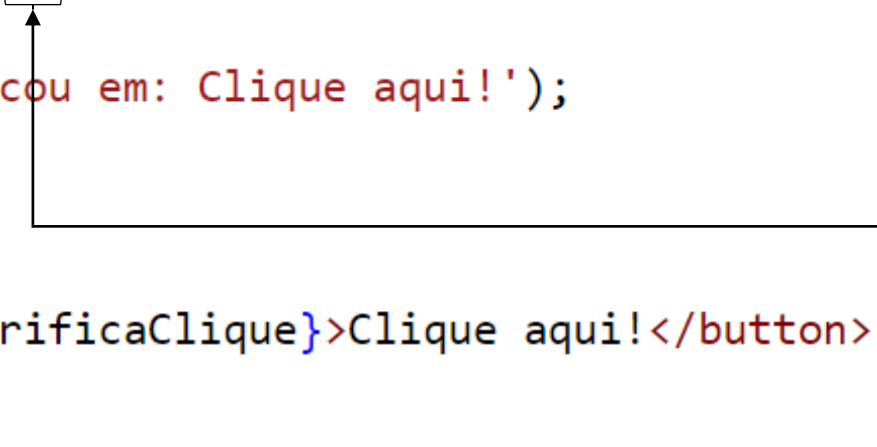
Eventos, em react, são nomeados no estilo camelCase.

Neste exemplo, o botão é um elemento no DOM, que quando clicado dispara um evento. O evento é tratado, manipulado ou atrelado a execução da função.

# Eventos React



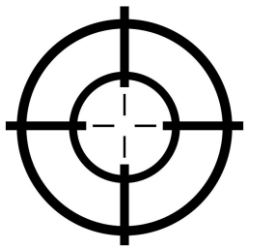
```
function App() {  
  function verificaClique(e){  
    e.preventDefault()  
    console.log('Você clicou em: Clique aqui!');  
  }  
  return (  
    <div>  
      <button onClick={verificaClique}>Clique aqui!</button>  
    </div>  
  );  
}  
export default App;
```

A diagram consisting of a horizontal line that starts from the right side of the code block, extends to the right, and then turns vertically upwards to end with an arrow pointing at the parameter 'e' in the function signature 'verificaClique(e)'.

Evento sintético (genérico).

Neste exemplo, o “e” representa, literalmente, o objeto evento, criado pela ação de clique e capturado pela função atrelada ao evento.

# Eventos React



O atrelamento é feito no método construtor!

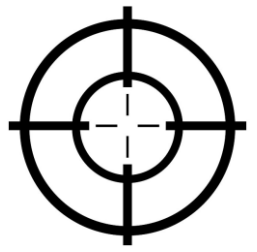
```
class App extends Component<any> {  
  constructor(props) {  
    super(props);  
    this.verificarClique = this.verificarClique.bind(this);  
  }  
  
  verificarClique(e) {  
    console.log(e);  
    console.log('Você clicou em: Clique aqui!');  
  }  
  
  render() {  
    return (  
      <div>  
        <button onClick={this.verificarClique}>Clique aqui!</button>  
      </div>  
    );  
  }  
}
```

```
export default App;
```

Quando se usa componentes de classe é necessário fazer o atrelamento do método ao componente, explicitamente.



# Eventos React



```
class App extends Component {
```

```
  verificarClique = (e) => {  
    console.log(e)  
    console.log('Você clicou em: Clique aqui!')  
  }
```

```
  render() {
```

```
    return (  
      <div>
```

```
        <button
```

```
          onClick={this.verificarClique}>Clique aqui!</button>
```

```
      </div>
```

```
    );
```

```
  }
```

```
}
```

```
export default App;
```

Nestes casos, o construtor só é utilizado se houver props ou state!

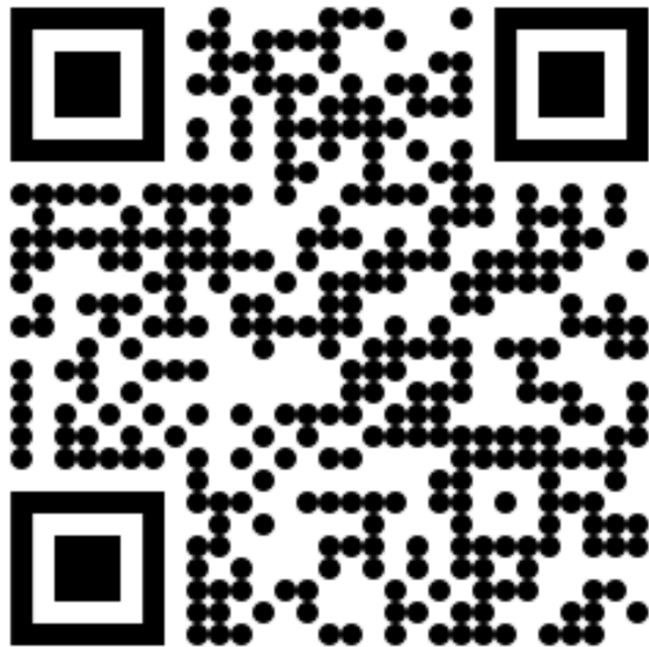
A alternativa para não precisar do atrelamento ao componente é declarar o método como uma função do tipo “arrow”.

# Eventos React

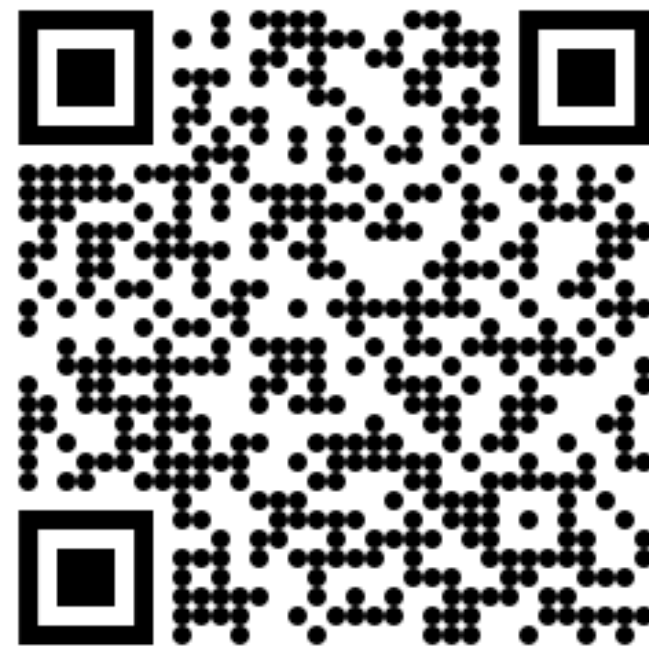


<https://github.com/gerson-pn>

[/react-simple-event](#)



[/react-simple-event-class](#)



```

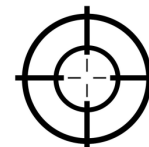
class App extends Component<{ nomes: string[] }, { nomes: string[] }> {
  constructor(props) {
    super(props)
    this.state = {
      nomes: this.props.nomes
    }
  }
  obterListaDeNomes = () => { ...
  }
  removerNomeDaLista(indice: number) { ...
  }
  render() { ...
  }
}
export default App;

```

```

ReactDOM.render(
  <React.StrictMode>
    <App nomes={['Usuario 1', 'Usuario 2', 'Usuario 3']} />
  </React.StrictMode>,
  document.getElementById('root')
);

```



```

    obterListaDeNomes = () => {
      let listaNomes = this.state.nomes.map((nome, indice) =>
        <li key={indice}>
          <div className="alinhamento" >
            <img className="tamanho" src={imagem} alt='' /><br/>
            <button onClick={this.removeNomeDaLista.bind(this, indice)} >{nome}</button>
          </div>
        </li>
      );
      return listaNomes
    }

    render() {
      let nomes = this.obterListaDeNomes()
      return (
        <div>
          <ul>{nomes}</ul>
        </div>
      )
    }
  }

```

O método retorna um código JSX, que será utilizado no método render().



```

class App extends Component<{ nomes: string[] }, { nomes: string[] }> {
  constructor(props) {
    super(props)
    this.state = {
      nomes: this.props.nomes
    }
  }
  obterListaDeNomes = () => { ...
}
  removerNomeDaLista(indice: number) {
    let listaNomes = this.state.nomes
    listaNomes.splice(indice,1)
    this.setState({
      nomes:listaNomes
    })
  }
  render() { ...
}
export default App;

```

```

  obterListaDeNomes = () => {
    let listaNomes = this.state.nomes.map((nome, indice) =>
      <li key={indice}>
        <div className="alinhamento" >
          <img className="tamanho" src={imagem} alt=''/><br/>
          <button onClick={this.removerNomeDaLista.bind(this, indice)} >{nome}</button>
        </div>
      </li>
    );
    return listaNomes
  }

```

A função é ativada pelo clique no elemento botão. Na ativação ela recebe o índice do elemento clicado.



# Renderização condicional

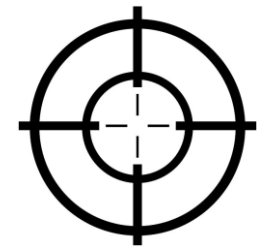


Renderização condicional ocorre quando deseja-se selecionar o que será renderizado (construído) no documento HTML, a partir de uma determinada condição.

Nestes casos pode-se utilizar as mesmas estruturas condicionais if/else, presente tanto na linguagem JavaScript como na linguagem TypeScript.

Importante! Não se deve colocar lógica no código jsx de retorno, a lógica deve ocorrer antes, sempre.

# Renderização condicional



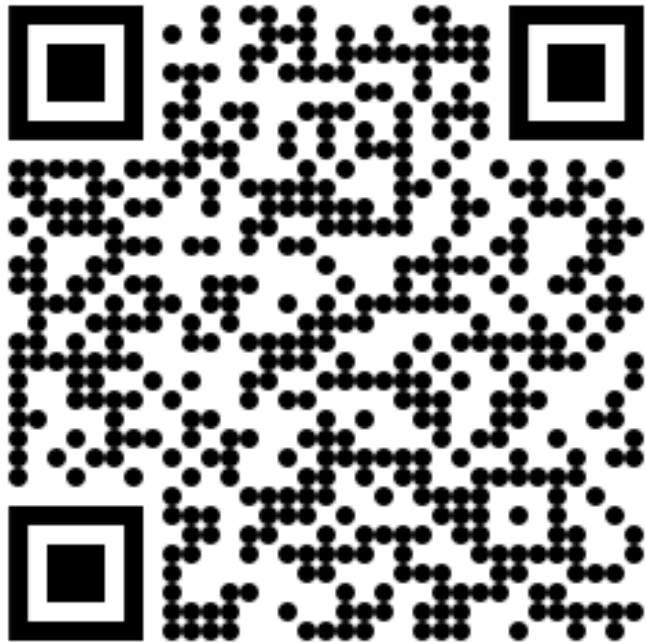
```
class App extends Component<{}, { texto: string, escolha: string }> {  
  constructor(props) { ...  
  }  
  capturarClique(valor: number) { ...  
  }  
  render() {  
    let escolha = this.state.escolha  
    if (escolha === '') {  
      return (  
        <div className="alinhamento"> ...  
        </div>  
      )  
    } else {  
      return (  
        <div className="alinhamento"> ...  
        </div>  
      )  
    }  
  }  
}  
  
export default App;
```

Neste exemplo, a renderização do componente ocorre após uma verificação, uma condição booleana, que leva em consideração o valor de um atributo do state.

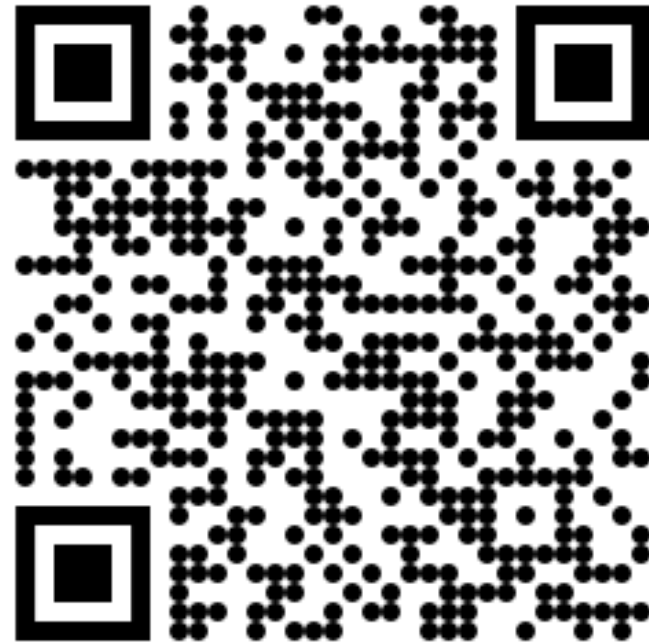
# React

<https://github.com/gerson-pn>

[/react-capturing-click-events](#)



[/react-conditional-event](#)





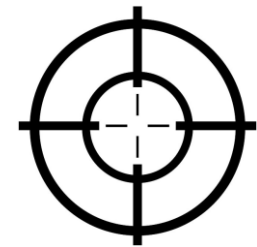
# Formulário



Um formulário HTML é um elemento, no DOM, usado para coletar entradas (dados) fornecidas por um usuário. Geralmente, essas entradas são enviadas e processadas no lado do servidor, no back-end.

Um formulário HTML é uma seção (espaço ou parte) no documento HTML, que contém elementos como campos de texto, campos de senha, caixas de seleção, botões de opção, botão enviar, menus etc.

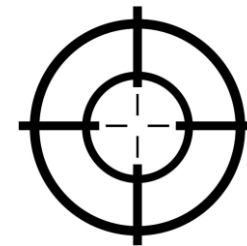
# Formulário



Elementos do tipo formulário funcionam de modo diferente de outros elementos do DOM, porque os formulários mantêm naturalmente algum estado interno nos seus elementos.

Quando se utiliza a biblioteca react, na maioria dos casos, designa-se funções ou métodos para manipular o envio do formulário e ter acesso aos seus dados, que foram fornecidos por um usuário. Isto é feito pelo que a biblioteca denomina de componentes controlados (controlled components).

# Formulário

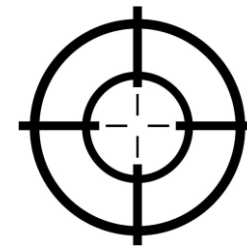


```
class App extends Component<{}, { nome: string }> {  
  constructor(props) {  
    super(props)  
    this.state = {  
      nome: ''  
    }  
    this.manipularEnvio = this.manipularEnvio.bind(this)  
    this.receberValorEntrada = this.receberValorEntrada.bind(this)  
  }  
  
  manipularEnvio(evento) { ... }  
  
  receberValorEntrada(evento) { ... }  
  
  render() { ... }  
}  
export default App;
```

As funções tratam (capturam) os eventos disparados pelo formulário e seus elementos!

Lembre-se, a variável “evento” representa o evento sintético, ou seja, o evento disparado pelo elemento do DOM.

# Formulário



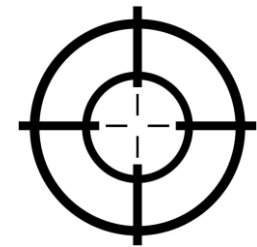
```
manipularEnvio(evento) {  
  evento.preventDefault()  
}
```

```
receberValorEntrada(evento) {  
  let entrada = evento.target.value  
  this.setState({  
    nome: entrada  
  })  
}
```

```
render() {  
  return (  
    <div>  
      <form onSubmit={this.manipularEnvio}>  
        <label>  
          <span>Nome: </span>  
          <input type="text" name="name" value={this.state.nome} onChange={this.receberValorEntrada} />  
        </label><br />  
        <input type="submit" value="Enviar" />  
      </form>  
    </div>  
  )  
}
```

“target” (alvo) representa o elemento que disparou o evento!

# Formulário



Em um formulário é comum a existência de vários campos, onde um usuário pode inserir valor. Para cada campo, pode-se utilizar uma função ou método, para controlar o elemento e seu estado.

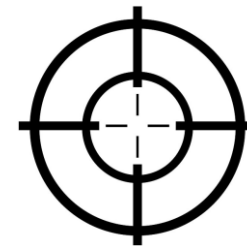
Existem vários elementos que podem ser utilizados para inserção de dados e a biblioteca react consegue manipular a maioria deles, como componentes controlados.

```

type state = {
  pnome: string,
  unome: string;
  password: string;
  email: string
}
class App extends Component<any, state> {
  constructor(props: any) {
    super(props)
    this.state = {
      pnome: '',
      unome: '',
      password: '',
      email: ''
    }
  }
}

```

# Formulário



```

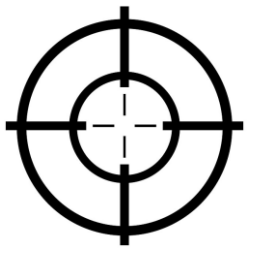
obterPnome = (evento: any) => {
  this.setState({
    pnome: evento.target.value
  })
  console.log(this.state.pnome)
}

obterUnome = (evento: any) => {
  this.setState({
    unome: evento.target.value
  })
  console.log(this.state.unome)
}

obterPassword = (evento: any) => {
  this.setState({
    password: evento.target.value
  })
  console.log(this.state.password)
}

```

# Formulário



```
obterPnome = (evento: any) => {  
  this.setState({  
    pnome: evento.target.value  
  })  
  console.log(this.state.pnome)  
}
```

```
obterUnome = (evento: any) => {  
  this.setState({  
    unome: evento.target.value  
  })  
  console.log(this.state.unome)  
}
```

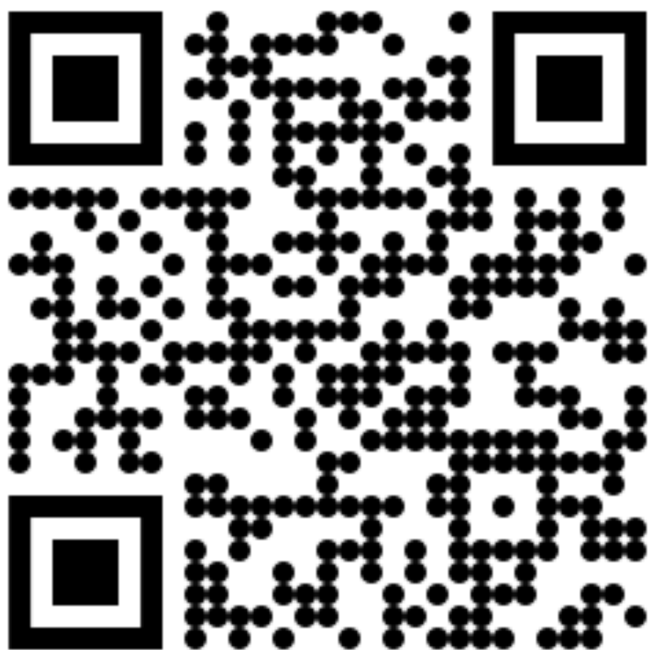
```
<div className="input-field col s6">  
  <input value={this.state.pnome} onChange={this.obterPnome} id="first_name" type="text" className="validate" />  
  <label htmlFor="first_name">Primeiro nome</label>  
</div>  
<div className="input-field col s6">  
  <input value={this.state.unome} onChange={this.obterUnome} id="last_name" type="text" className="validate" />  
  <label htmlFor="last_name">Último nome</label>  
</div>
```

# Formulário



<https://github.com/gerson-pn>

[/react-form-example](#)



[/react-form-example-large](#)

