

# **Análise e Desenvolvimento de Sistemas**

## **Frameworks Web I**

### **Aula 03 - React JS: parte 2**

---

---



# Formulários e Input de dados

# Controlled Components

# Controlled Components

No React, um “Controlled Component” é aquele em que o estado dos elementos de entrada do formulário é controlado pelo estado dentro do componente

- Em vez de permitir que os dados do formulário sejam manipulados diretamente pelo DOM, em um controlled component, os dados do formulário são manipulados por um componente React.
- Este conceito pode parecer um pouco complicado no início, mas traz diversas vantagens. Torna mais fácil modificar ou validar a entrada do usuário, bem como preencher previamente os valores padrão.

# Controlled Components

## Exemplo



[Acesse o projeto](#)

Nome:

```
1 import React from 'react';
2
3 const NameForm = () => {
4   const [name, setName] = React.useState('');
5
6   const handleChange = event => {
7     setName(event.target.value);
8   };
9
10  const handleSubmit = event => {
11    event.preventDefault();
12    alert(`Um nome foi enviado: ${name}`);
13  };
14
15  return (
16    <form onSubmit={handleSubmit}>
17      <label>
18        Nome:
19        <input type='text' value={name} onChange={handleChange} />
20      </label>
21      <input type='submit' value='Enviar' />
22    </form>
23  );
24 };
25
26 export default NameForm;
```

# Controlled Components

— □ ×

```
1 import React from 'react';
2
3 const NameForm = () => {
4   const [name, setName] = React.useState('');
5
6   const handleChange = event => {
7     setName(event.target.value);
8   };
9
10  const handleSubmit = event => {
11    event.preventDefault();
12    alert(`Um nome foi enviado: ${name}`);
13  };
14
15  return (
16    <form onSubmit={handleSubmit}>
17      <label>
18        Nome:
19        <input type='text' value={name} onChange={handleChange} />
20      </label>
21      <input type='submit' value='Enviar' />
22    </form>
23  );
24 };
25
26 export default NameForm;
```

Neste código, no componente NameForm, o valor do elemento de entrada está vinculado a “name”, uma variável de estado.

- Quando o usuário digita no input, o manipulador de eventos handleChange é acionado, atualizando o estado da variável “name”.
- Isso faz com que o componente seja renderizado novamente e o novo valor do nome seja exibido.

# Controlled Components

O mesmo princípio pode ser aplicado a outros elementos de formulário, como `textarea` e `select`.

- O importante a lembrar é que os dados do formulário são armazenados no estado do componente e o estado é atualizado por meio de manipuladores de eventos.
- Ao centralizar o estado no componente, ganhamos mais controle sobre os dados do formulário. Podemos executar facilmente operações como validação de dados, transformação de entrada ou habilitar e desabilitar envios de formulários com base no valor de entrada.

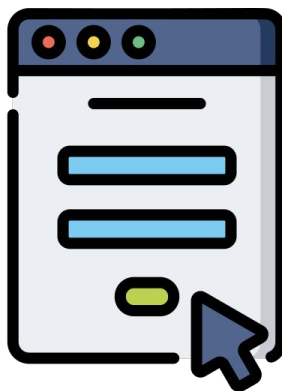


# Trabalhando com múltiplos inputs

# Múltiplos inputs

Quando precisar manipular vários elementos de entrada controlados, você pode adicionar um atributo de nome a cada elemento e permitir que um único manipulador funcione trate todos estes elementos

- Útil quando você tem um formulário com mais de um campo de entrada e deseja evitar escrever uma função de atualização de estado separada para cada um.



# Múltiplos inputs

## Exemplo

### Console ×

```
isGoing: false
isGoing: true
isGoing: false
numberOfGuests: 3
numberOfGuests: 4
numberOfGuests: 5
numberOfGuests: 6
```

Confirmando presença ☐

Número de convidados:



[Acesse o projeto](#)

```
import React from 'react';

const ReservationForm = () => {
  const [reservation, setReservation] = React.useState({
    isGoing: true,
    numberOfGuests: 2
  });

  const handleChange = (event) => {
    const target = event.target;
    const value = target.type === 'checkbox' ? target.checked : target.value;
    const name = target.name;

    setReservation({
      ...reservation,
      [name]: value
    });

    console.log(name + ": " + value);
  };

  return (
    <form>
      <label>
        Confirmando presença
        <input
          name='isGoing'
          type='checkbox'
          checked={reservation.isGoing}
          onChange={handleChange} />
      </label>
      <br />
      <label>
        Número de convidados:
        <input
          name='numberOfGuests'
          type='number'
          value={reservation.numberOfGuests}
          onChange={handleChange} />
      </label>
    </form>
  );
};

export default ReservationForm;
```

# Múltiplos inputs

No componente `ReservationForm`, temos dois campos de entrada: uma caixa de seleção e uma entrada numérica.

- Cada campo possui um atributo `name` que corresponde a uma propriedade do objeto `reservation`.
- A função `handleChange` usa a propriedade `name` do campo de entrada para decidir qual parte do estado atualizar.

Isso permite usar uma única função para lidar com atualizações em vários campos

# Múltiplos inputs

A função `handleChange` primeiro recupera o `event.target` (o campo de entrada que acionou o evento) e determina o novo valor da entrada

- Se a entrada for uma caixa de seleção, ela usará `target.checked`, caso contrário, usará `target.value`. Em seguida, ele usa a propriedade `name` do destino para atualizar a propriedade `state` correspondente.

```
const ReservationForm = () => {
  const [reservation, setReservation] = React.useState({
    isGoing: true,
    numberOfGuests: 2
  });

  const handleChange = (event) => {
    const target = event.target;
    const value = target.type === 'checkbox' ? target.checked : target.value;
    const name = target.name;

    setReservation({
      ...reservation,
      [name]: value
    });

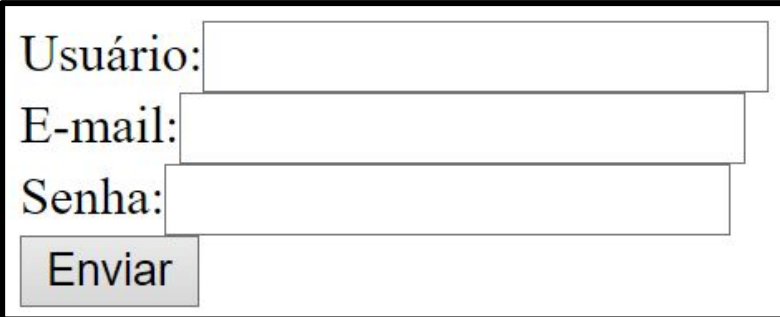
    console.log(name + ": " + value);
  };
};
```

# Validação e Submissão

# Validação e Submissão

A validação e o envio são etapas cruciais no tratamento da entrada do usuário nos formulários. Com o React, podemos aproveitar o estado para realizar a validação e gerenciar os envios de formulários com eficiência

- Como exemplo, vamos considerar um formulário de cadastro onde precisamos validar a entrada do usuário antes de enviar os dados



Usuário:

E-mail:

Senha:

# Validação e Submissão



[Acesse o projeto](#)

```
import React from 'react';

const RegistrationForm = () => {
  const [form, setForm] = React.useState({
    username: '',
    email: '',
    password: ''
  });

  const [errors, setErrors] = React.useState({});

  const handleChange = (event) => {
    setForm({
      ...form,
      [event.target.name]: event.target.value
    });
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    const validationErrors = validate(form);
    setErrors(validationErrors);

    console.log("Erros encontrados");
    console.log(validationErrors);

    if (Object.keys(validationErrors).length === 0) {
      console.log("Dados do formulário");
      console.log(form) // Realizar o envio aqui
    }
  };
};
```

```
return (
  <form onSubmit={handleSubmit}>
    <label>Usuário:
    <input
      type="text"
      name="username"
      value={form.username || ""}
      onChange={handleChange}
    />
    </label>
    <br />
    <label>E-mail:
    <input
      type="text"
      name="email"
      value={form.email || ""}
      onChange={handleChange}
    />
    </label>
    <br />
    <label>Senha:
    <input
      type="password"
      name="password"
      value={form.password || ""}
      onChange={handleChange}
    />
    </label>
    <br />
    <input type="submit" />
  </form>
);

function validate(form) {
  const errors = {};

  if (form.username.trim() === '') {
    errors.username = "Obrigatório informar o usuário."
  }

  if (!/^[S+@\\S+\\.S+/.test(form.email)) {
    errors.email = "E-mail inválido."
  }

  if (form.password.length < 6) {
    errors.password = "A senha deve ter no mínimo 6 caracteres."
  }

  return errors;
}

export default RegistrationForm;
```



# Validação e Submissão

No componente `RegistrationForm`, mantemos um state `“form”` para os dados do formulário e um state `“errors”` para os erros de validação

- A função `handleChange` atualiza o estado do formulário conforme o usuário digita nos campos de entrada.
- Quando o formulário é enviado, evitamos o comportamento padrão de envio do formulário e chamamos a função de validação. Esta função verifica os dados do formulário e retorna um objeto onde as chaves são os nomes dos campos e os valores são as mensagens de erro.

# Validação e Submissão

Se o objeto “errors” estiver vazio, significa que a validação foi aprovada e podemos prosseguir com o envio do formulário

- Neste exemplo, simplesmente registramos os dados do formulário no console, mas em uma aplicação real, seria onde você faria uma solicitação ao seu servidor.
- A função de validação faz uma validação básica de entrada: verifica se o nome de usuário não está vazio, se o e-mail está em um formato válido e se a senha tem pelo menos 6 caracteres.

Mais exemples

# Mais exemplos

## Exemplo 1



[Acesse o projeto](#)

```
import React from 'react';

function MyForm() {
  const [name, setName] = React.useState("");

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`O nome digitado foi: ${name}`)
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Digite seu nome:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <input type="submit" />
    </form>
  )
}

export default MyForm;
```

# Mais exemplos

## Exemplo 2



[Acesse o projeto](#)

```
import React from 'react';

function MyForm() {
  const [inputs, setInputs] = React.useState({});

  const handleChange = (event) => {
    const name = event.target.name;
    const value = event.target.value;
    setInputs(values => ({...values, [name]: value}))
  }

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Nome: ${inputs.username}`);
    alert(`Idade: ${inputs.age}`);
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>Insira o seu nome:
      <input
        type="text"
        name="username"
        value={inputs.username || ""}
        onChange={handleChange}
      />
      </label>
      <br />
      <label>Insira a sua idade:
      <input
        type="number"
        name="age"
        value={inputs.age || ""}
        onChange={handleChange}
      />
      </label>
      <br />
      <input type="submit" />
    </form>
  )
}

export default MyForm;
```

# Mais exemplos

## Exemplo 3



[Acesse o projeto](#)

```
import React from 'react';

function MyForm() {
  const [textarea, setTextarea] = React.useState(
    "O conteúdo do textarea vai no atributo value"
  );

  const handleChange = (event) => {
    setTextarea(event.target.value);
    console.log(event.target.value);
  }

  return (
    <form>
      <textarea value={textarea} onChange={handleChange} />
    </form>
  )
}

export default MyForm;
```

# Mais exemplos

## Exemplo 4



[Acesse o projeto](#)

```
import React from 'react';

function MyForm() {
  const [myCar, setMyCar] = React.useState("Volvo");

  const handleChange = (event) => {
    setMyCar(event.target.value);
    console.log(`Item escolhido: ${event.target.value}`);
  }

  return (
    <form>
      <select value={myCar} onChange={handleChange}>
        <option value="Ford">Ford</option>
        <option value="Volvo">Volvo</option>
        <option value="Fiat">Fiat</option>
      </select>
    </form>
  )
}

export default MyForm;
```

# React + Formik



# React + Formik



[Acesse a documentação](#)

Formik é uma biblioteca popular de gerenciamento de formulários para React que simplifica o processo de construção e validação de formulários

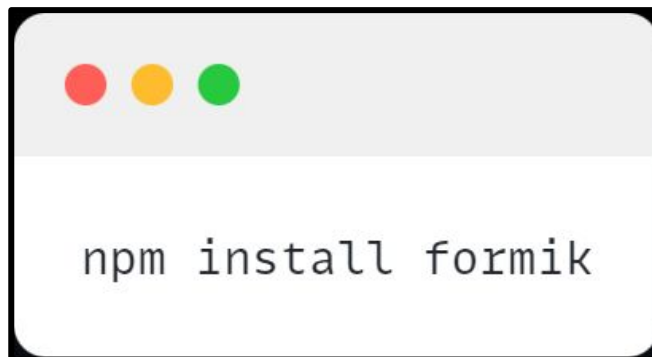
- A biblioteca fornece uma maneira direta e simples de lidar com o estado do formulário, validação, envio de formulário e tratamento de erros.



# FORMIK

# React + Formik

Para usar o Formik em seu projeto React, comece instalando a biblioteca:

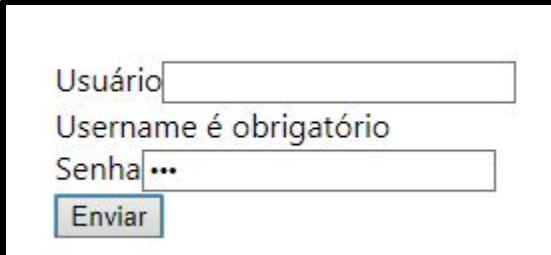


- Depois de instalado, você pode importar os componentes e ganchos necessários do Formik para começar a construir seus formulários

# React + Formik

## Exemplo

- Vamos ver um exemplo simples para demonstrar como o Formik funciona.
- Criaremos um formulário de login básico que coleta nome de usuário e senha, realiza validação e exibe mensagens de erro se os campos estiverem vazios.



Usário

Username é obrigatório

Senha

# React + Formik

## Exemplo



[Acesse o projeto](#)

```
import { Formik, Form, Field, ErrorMessage } from 'formik';

const initialValues = {
  username: '',
  password: '',
};

const validate = (values) => {
  const errors = {};

  if (!values.username) {
    errors.username = "Username é obrigatório"
  }

  if (!values.password) {
    errors.password = "Senha é obrigatória"
  }

  return errors;
}

const LoginForm = () => {
  return (
    <Formik
      initialValues={initialValues}
      validate={validate}
      onSubmit={({values}) => {
        console.log(values);
      }}
    >
      <Form>
        <div>
          <label htmlFor='username'>Usuário</label>
          <Field type="text" id="username" name="username" />
          <ErrorMessage name="username" component="div" />
        </div>

        <div>
          <label htmlFor='password'>Senha</label>
          <Field type="password" id="password" name="password" />
          <ErrorMessage name="password" component="div" />
        </div>

        <button type='submit'>Enviar</button>
      </Form>
    </Formik>
  );
};

export default LoginForm;
```

# React + Formik

## Exemplo

- Primeiro, devemos importar os componentes e ganchos necessários do Formik:



```
import { Formik, Form, Field, ErrorMessage } from 'formik';
```

# React + Formik

## Exemplo

- Em seguida, são definidos os valores iniciais do formulário e as suas regras de validação:

```
const initialValues = {  
  username: '',  
  password: '',  
};  
  
const validate = (values) => {  
  const errors = {};  
  
  if (!values.username) {  
    errors.username = "Username é obrigatório"  
  }  
  
  if (!values.password) {  
    errors.password = "Senha é obrigatória"  
  }  
  
  return errors;  
}
```

# React + Formik

## O próximo passo é criar o formulário, utilizando o componente Formik

- Para isso, definimos a estrutura do formulário usando o componente `<Form>` do Formik. Dentro do formulário, utilizamos o componente `<Field>` para definir campos de entrada para coleta de dados.
- As propriedades `id` e `name` em `<Field>` correspondem ao nome do campo no objeto `“form”`.
- O componente `<ErrorMessage>` é usado para exibir mensagens de erro se houver algum erro de validação associado a um campo específico.
- Por fim, definimos o `“onSubmit”` no componente `<Formik>` para lidar com o envio de formulários.

# React + Formik

## Exemplo

```
const LoginForm = () => {
  return (
    <Formik
      initialValues={initialValues}
      validate={validate}
      onSubmit={(values) => {
        console.log(values);
      }}
    >
      <Form>
        <div>
          <label htmlFor='username'>Usuário</label>
          <Field type="text" id="username" name="username" />
          <ErrorMessage name="username" component="div" />
        </div>

        <div>
          <label htmlFor='password'>Senha</label>
          <Field type="password" id="password" name="password" />
          <ErrorMessage name="password" component="div" />
        </div>

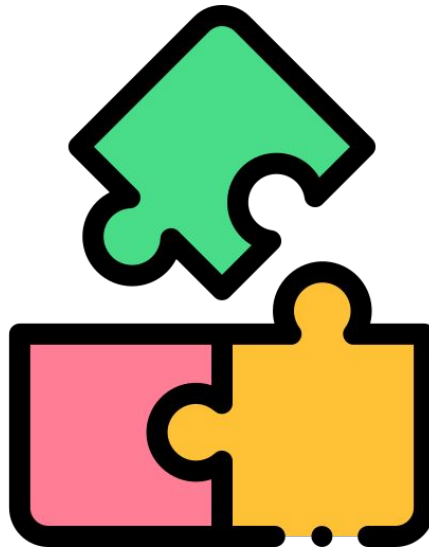
        <button type='submit'>Enviar</button>
      </Form>
    </Formik>
  );
};
```



# Recursos

# Recursos

O Formik oferece uma variedade de recursos que tornam o gerenciamento de formulários em aplicativos React mais conveniente e eficiente



# Validação

# Recursos

## Validação de formulários

- O Formik oferece suporte de validação integrado por meio do suporte de validação. Você pode definir uma função de validação que receba os valores do formulário como argumento e retorne um objeto contendo quaisquer erros de validação.

```
const validate = (values) => {  
  const errors = {};  
  
  if (!values.username) {  
    errors.username = "Username é obrigatório"  
  }  
  
  if (!values.password) {  
    errors.password = "Senha é obrigatória"  
  }  
  
  return errors;  
}
```

# Submissão

# Recursos

## Submissão de formulários

- Simplifica o envio de formulários fornecendo a propriedade “onSubmit”, onde você pode definir a lógica para lidar com o envio de formulários.

```
<Formik
  initialValues={initialValues}
  validate={validate}
  onSubmit={({values, { setSubmitting } }) => {
    chamaApi(values)
      .then((response) => {
        console.log("Envio com sucesso!");
        console.log(response);
      })
      .catch((error) => {
        console.log("Erro ao enviar dados!");
        console.log(error);
      })
      .finally(() => {
        setSubmitting(false);
      });
  }}
>
<Form>
  {/*Conteúdo do formulário */}
</Form>
</Formik>
```

# Recursos

## Submissão de formulários

- A função `onSubmit` recebe os valores do formulário como argumento, permitindo realizar ações como fazer solicitações de API, atualizar o estado ou navegar para uma página diferente.
- Já a função `setSubmit` fornecida pelo Formik permite controlar o estado de envio do formulário, o que pode ser útil para mostrar indicadores de carregamento ou desabilitar o botão de envio enquanto o formulário está sendo enviado.

# Gerenciamento de estado



# Recursos

## Gerenciamento de estado

- Formik cuida do gerenciamento do estado do formulário, incluindo rastreamento de valores do formulário, campos clicados e status de envio.
- Ele fornece esses valores por meio das propriedades “values”, “touched” e “isSubmitting”, permitindo que você os acesse e utilize na lógica do seu formulário.

```
<Formik
  initialValues={initialValues}
  onSubmit={({values, { setSubmitting } }) => {
    console.log(values);

    // Verifica se o campo foi clicado
    if(values.email && touched.email) {
      // Realiza alguma ação
    }

    // Verifica o status de submissão
    if (isSubmitting) {
      // Mostra indicador de loading
    }

    // Gerencia a submissão do formulário
    enviaApi(values)
      .then((response) => {
        console.log(response);
      })
      .catch((error) => {
        console.log(error);
      })
      .finally(() => {
        setSubmitting(false);
      });
  }}
>
<Form>
  /*Conteúdo do formulário */
</Form>
</Formik>
```

Mais exemples

# Mais exemplos



[Acesse o projeto](#)

## Exemplo

### Entre em Contato

**Nome:**

**Email:**

**Mensagem:**

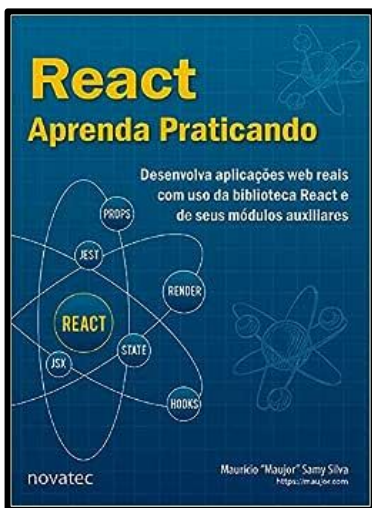
Avalie a aula!

Avalie a aula de hoje :)



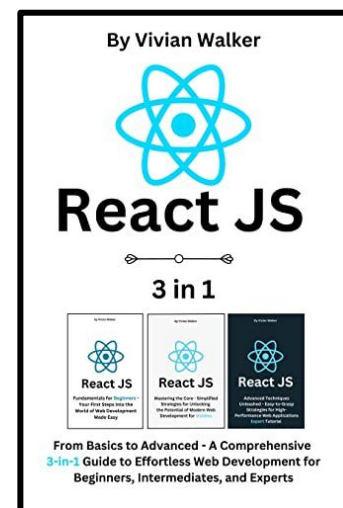
# Referências

# Referências



SILVA, Maurício Samy. **React - Aprenda praticando**. Novatec. 2021.

Walker, Vivian. **React JS: From Basics to Advanced - A Comprehensive 3-in-1 Guide to Effortless Web Development for Beginners, Intermediates, and Experts**.



# Referências



W3Schools. React Tutorial. Disponível em:  
<<https://www.w3schools.com/react/>>. Acesso em 07 ago. 2024.



# Referências



Todos os ícones utilizados são gratuitos e livres para uso pessoal e comercial.

Eles foram retirados do site <https://www.flaticon.com/>.

**Autores:**

Freepik

# **Análise e Desenvolvimento de Sistemas**

## **Frameworks Web I**

### **Aula 03 - React JS: parte 2**

---

---