

# React + Forms

# O que vamos ver hoje?

- Como fazemos formulários e problemas com isso
- Revisão da tag `<form>`
- Validações
- Unificação dos handlers
- Criação do hook `useForm()`

# Como estamos fazendo formulários

# Como estamos fazendo forms?



- Criamos um input no JSX: `<input />`
- Cada input deve ter um **estado** relacionado
  - Esse estado é passado na propriedade **value**
- Criamos uma **função** para guardar o valor atual no estado correspondente
  - Essa função é passada na propriedade **onChange**

# Como estamos fazendo forms?



- Quando precisamos enviar os dados do form como **body de uma requisição**, em geral criamos um novo objeto pegando os valores dos estados:

```
const body = {  
  email: estadoEmail,  
  senha: estadoSenha  
}
```

# Problemas

# Problemas

- Como fazer validações?
- Ter que ficar criando um estado e uma função para cada input
- Necessidade de criar um novo objeto toda hora pras requisições com o formato pedido pela API

**Revisão: tag <form>**

# Revisão: Forms no HTML

- No HTML, temos a tag `<form>`
- Vantagens 😊
  - Faz algumas validações nativamente
  - Nos permite inserir nossas validações personalizadas
- Desvantagens 😓
  - Comportamento padrão atualiza a página quando o formulário é enviado

# Revisão: Como usar a tag form



- Colocamos dentro da tag form todos os nossos **inputs** e um **botão** para enviar os dados
- A função que ocorre ao finalizar o formulário deve ser passada na propriedade **onSubmit** da tag form
- Devemos **bloquear o comportamento padrão** para que a página não seja atualizada!

# Forms + React - Pontos de atenção

- A função que faz a requisição deve (preferencialmente) ser passada no evento **onSubmit** do elemento **form**
  - E não no **onClick** do **button**
- A função deve receber um evento e **obrigatoriamente** chamar a função **event.preventDefault()**
  - Isso impede que a página atualize sozinha

# Validações

# O que é validação?

- Verificar se todos os dados **obrigatórios** foram preenchidos
- Verificar se o dado está no **formato** esperado
- Exemplos
  - O email é realmente um email?
  - O CPF é um CPF?
  - A senha tem o tamanho que deveria?
  - O CEP existe?

# Por que validar formulários?

- Uma regra **pétrea** de desenvolvimento em geral é *“Nunca confie no input dos usuários”*
- Presuma que o usuário inserirá os dados de forma incompleta/incorreta
- Dada essa regra, precisamos **preparar nossos formulários** para conseguirmos capturar erros do usuário diretamente no preenchimento

# Validação - required

```
<label htmlFor="username">Nome de usuário</label>  
<input id="username" type="text" required />
```

Nome de usuário:

Enviar




Please fill out this field.

# Validação - type

```
<label forHtml="email">E-mail</label>  
<input id="email" type="email" required />
```

E-mail:

 A part following '@' should not contain the symbol '}'.

E-mail:

 '!' is used at a wrong position in 'sd!'.

E-mail:

 Please include an '@' in the email address. 'asd' is missing an '@'.

# Validação - pattern

- Forma mais simples e customizável de se fazer validação, só tem um porém...

REGEX

# REGEX - O que é?

- Regular Expressions ou **Expressões Regulares**
- É uma forma de descrever regras que capturam conjuntos de caracteres, ou **encontram padrões em texto**, muito usado para validações
- Está disponível em várias linguagens de programação e tem fama de ser difícil entre devs, mas **não é um monstro!**

# Validação - pattern

- Abaixo temos o RegEx “[A-Z]{3}”. Isso significa que o padrão requer 3 letras maiúsculas
- Ao passarmos para um input como **pattern**, no momento do submit do form, o valor será testado contra o padrão que provemos

```
<input type="text" name="country-code" pattern="[A-Z]{3}" title="Código do País (BRA, USA, FRA)">
```

## Validação - pattern (adendo)

- Detalhe, para usar com o componente `<TextField>` do Material-UI usamos a prop **inputProps**:

```
inputProps={ { pattern: "[a-z]" } }
```

# Validação - Exemplos Regex

- Validar E-mail

`[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$`

- Validar CPF

`[0-9]{3}\.[0-9]{3}\.[0-9]{3}-[0-9]{2}`

- Somente letras minúsculas (sem espaços)

`^[a-z]+$`

- Site para ajudar: <https://regexr.com/>

[Vamos ver na prática!](#) 

# Unificar estados e handlers de onChange

# Unificar handlers de onChange



- O código que criamos para cada um dos inputs é muito parecido! Mudam apenas algumas coisinhas

```
1 const TelaLogin = () => {
2   const [email, setEmail] = useState("")
3   const [senha, setSenha] = useState("")
4
5   const mudaEmail = (event) => {
6     setEmail(event.target.value)
7   }
8
9   const mudaSenha = (event) => {
10    setSenha(event.target.value)
11  }
12 }
```

# Unificar handlers de onChange

- A única coisa que muda são os nomes de cada input!
- Para nos ajudar a unificar isso, podemos utilizar a propriedade **name** do `<input />`, que dá a ele um nome com o qual poderemos nos referir ao campo
- Assim, nosso input passará a precisar de 3 coisinhas: **name**, **value** e **onChange**

# Unificar handlers de onChange

- Podemos criar um **método único** responsável por lidar com **qualquer alteração de campo** no componente:
  - Economizamos muitas linhas repetidas
  - Centralizamos a lógica
- Também podemos criar um **único estado** que seja um objeto com todos os dados do formulário

## onChange

- Responsável por lidar com as mudanças de todos os inputs

Atenção para a sintaxe nova: **{ [name]: value }**

- A chave e o valor serão dinâmicos

```
1 import React, { useState } from "react"
2
3 const Login = () => {
4   const [form, setForm] = useState({ email: "", password: "" })
5
6   const onChange = (event) => {
7     const { name, value } = event.target
8     setForm({ ...form, [name]: value })
9   }
10
11  const handleClick = (event) => {
12    event.preventDefault()
13    //Aqui entraria a sua requisição do axios
14    console.log(form)
15  }
16
17  return (
18    <form onSubmit={handleClick}>
19      <input
20        name="email"
21        value={form.email}
22        onChange={onChange}
23        type="email"
24      />
25      <input
26        name="password"
27        value={form.password}
28        onChange={onChange}
29        type="password"
30      />
31      <button>Fazer Login</button>
32    </form>
33  )
34 }
35
36 export default Login
37
```

# Criando um Custom Hook

# Custom Hook - useForm()

- Essa lógica de formulários irá se **repetir** em várias telas dos nossos sites
- Podemos extrair essa lógica para uma função reutilizável!
- Mas como essa função precisa usar o hook useState, ela será um **custom hook**

# Custom Hook - useForm()

- Criamos um hook chamado **useForm()** que contém os valores e a lógica de atualização do formulário

**Objeto com  
os valores do  
formulário**

**Handler  
unificado**

```
1 const useForm = (initialState) => {  
2   const [form, setForm] = useState(initialState)  
3  
4   const onChange = (event) => {  
5     const { name, value } = event.target  
6     setForm({ ...form, [name]: value })  
7   }  
8  
9   return [form, onChange]  
10 }
```

# Custom Hook - useForm()



- Usamos o hook informando o estado inicial
- Chamamos dentro do input o estado e o onChange

```
1 const Login = () => {
2   const [form, onChange] = useForm({ email: "", password: "" })
3
4   const handleClick = (event) => {
5     event.preventDefault()
6     console.log("BODY:", form)
7   }
8
9   return (
10    <form onSubmit={handleClick}>
11      <input
12        name="email"
13        value={form.email}
14        onChange={onChange}
15        placeholder="E-mail"
16        type="email"
17      />
18      <input
19        name="password"
20        value={form.password}
21        onChange={onChange}
22        placeholder="Senha"
23        type="password"
24      />
25      <button>Fazer Login</button>
26    </form>
27  )
28 }
```

# Exemplo Prático

# Exercício Prático

- Crie um formulário utilizando o custom hook **useForm** que siga as seguintes especificações:
- Todos os campos são obrigatórios
  - **Nome** → No mínimo 3 letras
  - **Idade** → No mínimo 18
  - **Email** → Válido

Vamos ver na prática!

