# React + Forms

Labenu_

# 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

Labenu_

# 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**

**Vamos ver na prática!** 🔬

# 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
}
```

**Vamos ver na prática!** 🔬

# Problemas

Labenu_

# 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>

Labenu_

# 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!

**Vamos ver na prática!** 🔬

# Forms + React - Pontos de atenção ⚠️

- 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

Labenu_

# 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

# 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

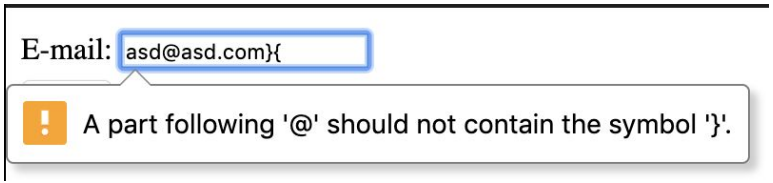
- Dada essa regra, precisamos **preparar nossos formulários** para conseguirmos capturar erros do usuário diretamente no preenchimento

# Validação - required ✅

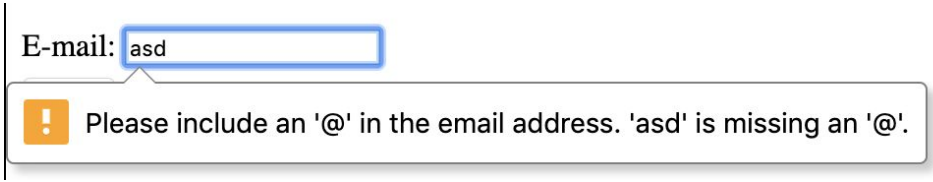
```
<label forHtml="username">Nome de usuário</label>
<input id="username" type="text" required />
```

# Validação - type ✅

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







# 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/

# Exercício 1

- Crie um formulário de login com os campos e-mail e senha. Os campos devem possuir as seguintes validações:
  - Ambos são de preenchimento obrigatório
  - O e-mail deve aceitar apenas e-mails válidos
  - A senha não pode ter menos que 3 caracteres

# Unificar estados e handlers de onChange

Labenu_

# 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**

# 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

**Vamos ver na prática!** 🔬

## 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

```jsx
import React, { useState } from "react"

const Login = () => {
  const [form, setForm] = useState({ email: "", password: "" })

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

  const handleClick = (event) => {
    event.preventDefault()
    //Aqui entraria a sua requisição do axios
    console.log(form)
  }

  return (
    <form onSubmit={handleClick}>
      <input
        name="email"
        value={form.email}
        onChange={onChange}
        type="email"
      />
      <input
        name="password"
        value={form.password}
        onChange={onChange}
        type="password"
      />
      <button>Fazer Login</button>
    </form>
  )
}

export default Login

```

# Criando um Custom Hook

Labenu_

# 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**

**Vamos ver na prática!** 🔬

# 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

Labenu_

# 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

Labenu_

Obrigado(a)!