

# FRONT-END DESIGN ENGINEERING

[React – Hook Form – Consumindo API externa]

[Aula 19]

[Atualizado em: 08/04/2024]



## **REACT – HOOK FORM – CONSUMINDO API EXTERNA**

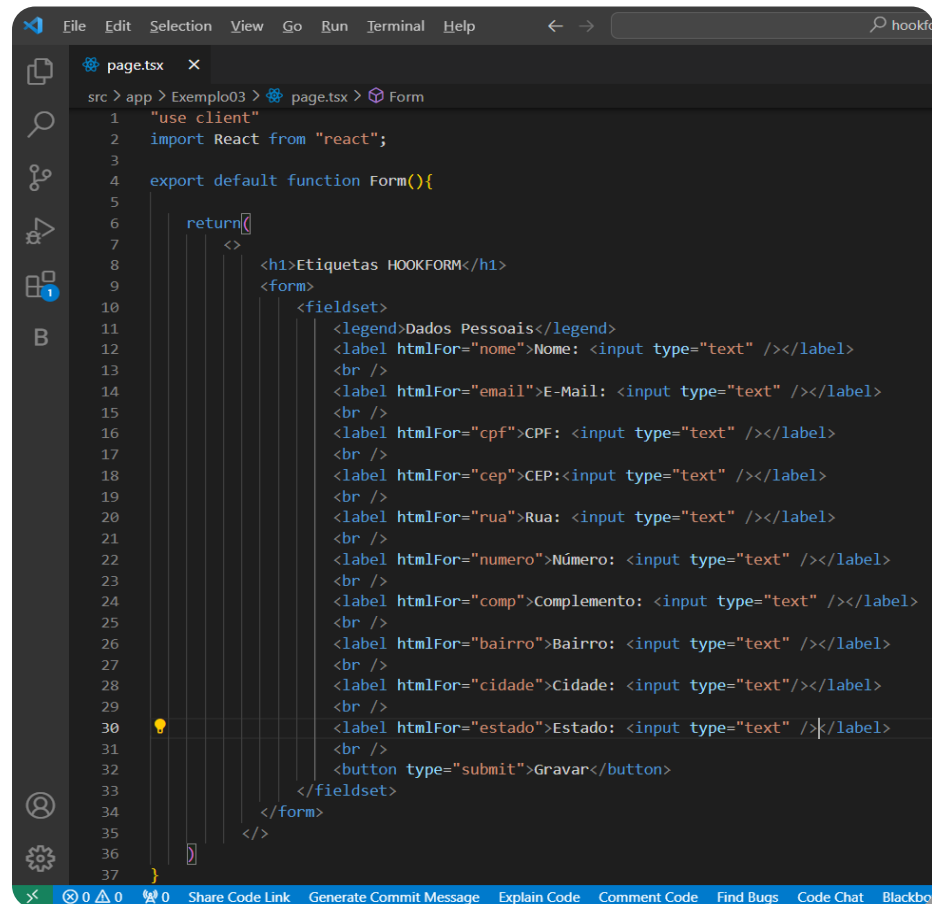
REACT – HOOK FORM

# REACT – HOOK FORM

- ✓ **Introdução ao Hook Form – Consumindo API externa**
- ✓ Vamos modificar o formulário criado na aula passada e inserir mais alguns campos, como por exemplo: CEP, RUA, NUMERO, COMPLEMENTO, BAIRRO, CIDADE, ESTADO e ESTADO.
- ✓ As APIs (Application Programming Interface ou Interfaces de Programação de Aplicações) são fundamentais para a comunicação entre diferentes sistemas e para fornecer acesso a funcionalidades específicas de uma aplicação. No ambiente TypeScript, criar e consumir APIs é uma tarefa acessível e eficiente, permitindo a construção de serviços web poderosos e escaláveis.
- ✓ No nosso, caso não vamos construir, ainda, mas vamos sim, consumir uma API.

# REACT – HOOK FORM

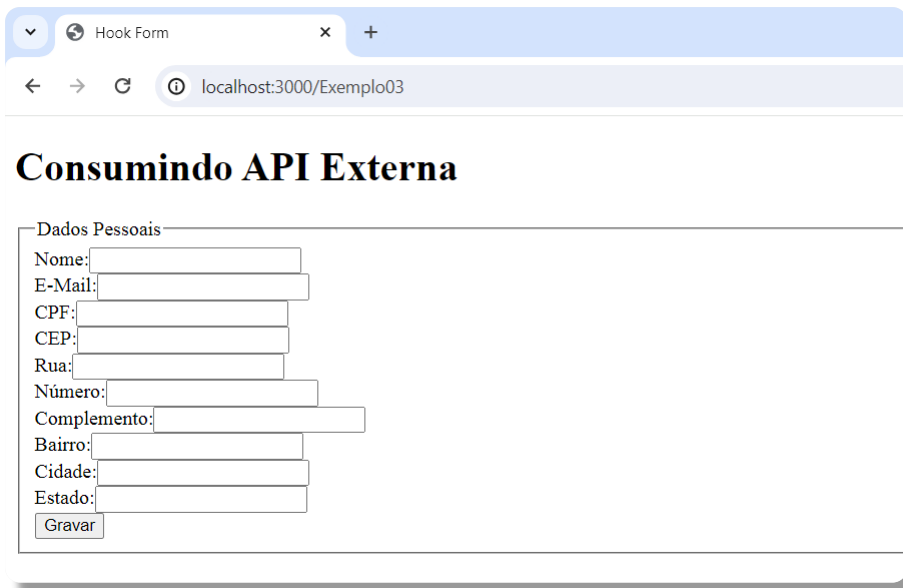
- ✓ **Introdução ao Hook Form – Consumindo API externa**
- ✓ Vamos criar a pasta 'Exemplo03', criado na aplicação anterior e criar o arquivo 'page.tsx'
- ✓ Vamos criar um formulário simples.



```
1 "use client"
2 import React from "react";
3
4 export default function Form(){
5
6   return(
7     <>
8       <h1>Etiquetas HOOKFORM</h1>
9       <form>
10         <fieldset>
11           <legend>Dados Pessoais</legend>
12           <label htmlFor="nome">Nome: <input type="text" /></label>
13           <br />
14           <label htmlFor="email">E-Mail: <input type="text" /></label>
15           <br />
16           <label htmlFor="cpf">CPF: <input type="text" /></label>
17           <br />
18           <label htmlFor="cep">CEP: <input type="text" /></label>
19           <br />
20           <label htmlFor="rua">Rua: <input type="text" /></label>
21           <br />
22           <label htmlFor="numero">Número: <input type="text" /></label>
23           <br />
24           <label htmlFor="comp">Complemento: <input type="text" /></label>
25           <br />
26           <label htmlFor="bairro">Bairro: <input type="text" /></label>
27           <br />
28           <label htmlFor="cidade">Cidade: <input type="text" /></label>
29           <br />
30           <label htmlFor="estado">Estado: <input type="text" /></label>
31           <br />
32           <button type="submit">Gravar</button>
33         </fieldset>
34       </form>
35     </>
36   )
37 }
```

# REACT – HOOK FORM

- ✓ **Introdução ao Hook Form – Consumindo API externa**
- ✓ Executando nossa aplicação, teremos o resultado abaixo.



A screenshot of a web browser window. The browser's address bar shows 'localhost:3000/Exemplo03'. The page title is 'Hook Form'. The main heading of the page is 'Consumindo API Externa'. Below the heading is a form titled 'Dados Pessoais'. The form contains the following input fields: 'Nome:', 'E-Mail:', 'CPF:', 'CEP:', 'Rua:', 'Número:', 'Complemento:', 'Bairro:', 'Cidade:', and 'Estado:'. Each field is followed by a text input box. At the bottom of the form is a button labeled 'Gravar'.

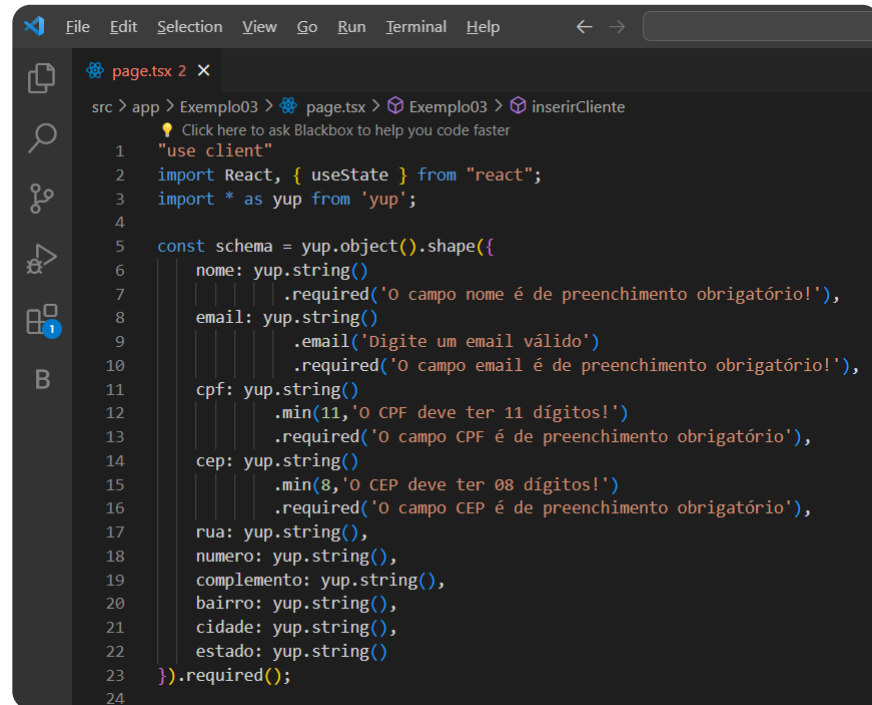
# REACT – HOOK FORM

- ✓ **Introdução ao Hook Form – Consumindo API externa**
- ✓ Vamos agora, modificar nosso código.
  - ✓ Vamos criar algumas variáveis no TypeScript que utilizaremos em algum momento em nosso código.
  - ✓ Criaremos também regras de validação definidas no esquema de *schema* que podem ser utilizadas para validar objetos JavaScript/TypeScript que seguem essa estrutura. Por exemplo, ao aplicar esse esquema a um objeto que contém os campos `firstName`, `lastName` e `cpf`, a biblioteca Yup verificará se esses campos atendem às regras de validação definidas no *schema*. Algo que já fizemos no Exemplo02.

# REACT – HOOK FORM

## ✓ Introdução ao Hook Form – Consumindo API externa

✓ Vamos agora, modificar nosso código.



```
src > app > Exemplo03 > page.tsx > Exemplo03 > inserirCliente
  ⚡ Click here to ask Blackbox to help you code faster
1  "use client"
2  import React, { useState } from "react";
3  import * as yup from 'yup';
4
5  const schema = yup.object().shape({
6    nome: yup.string()
7      .required('O campo nome é de preenchimento obrigatório!'),
8    email: yup.string()
9      .email('Digite um email válido')
10     .required('O campo email é de preenchimento obrigatório!'),
11    cpf: yup.string()
12      .min(11, 'O CPF deve ter 11 dígitos!')
13      .required('O campo CPF é de preenchimento obrigatório!'),
14    cep: yup.string()
15      .min(8, 'O CEP deve ter 08 dígitos!')
16      .required('O campo CEP é de preenchimento obrigatório!'),
17    rua: yup.string(),
18    numero: yup.string(),
19    complemento: yup.string(),
20    bairro: yup.string(),
21    cidade: yup.string(),
22    estado: yup.string()
23  }).required();
24
```

# REACT – HOOK FORM

## ✓ Introdução ao Hook Form – Consumindo API externa

### ✓ Exemplo 03 – Analisando linha a linha o arquivo page.tsx

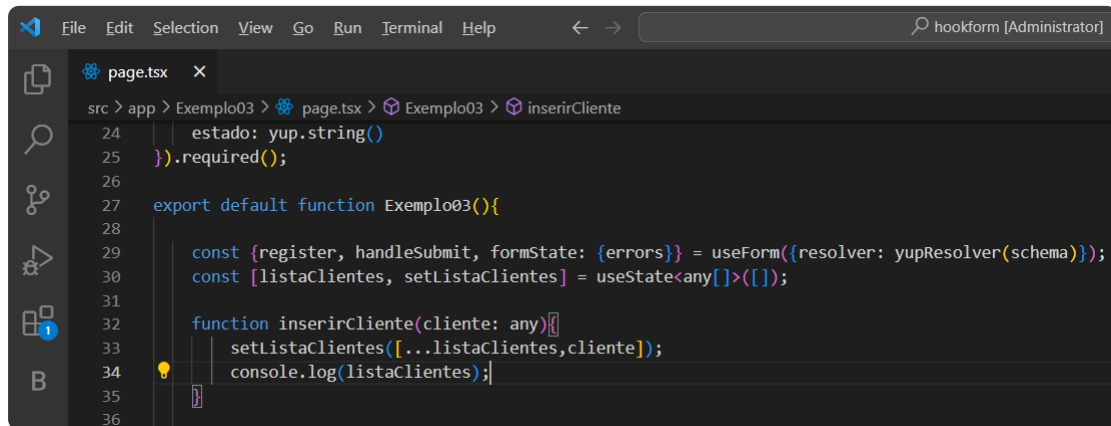
```
'use client'  
1 // É uma convenção para indicar que o componente é renderizado no cliente, significando que ele é executado no  
navegador do usuário, após o carregamento inicial da página. Essa convenção ajuda a identificar facilmente  
quais partes do código são específicas do cliente, em oposição às partes que podem ser executadas no servidor.  
Isso pode ser útil em projetos que utilizam renderização do lado do servidor (SSR) ou que têm lógica  
compartilhada entre o cliente e o servidor.  
  
import React, { useState } from "react";  
2 // Esta linha importa as funções React e useState do pacote "react". A função useState é um hook do React que  
permite adicionar estado a componentes de função.  
  
import * as yup from 'yup';  
3 // Esta linha importa todos os membros exportados do pacote yup e os coloca em um objeto chamado yup. O yup é uma  
biblioteca de validação usada para definir esquemas de validação para os dados do formulário.  
4  
  
const schema = yup.object().shape({  
5 // Esta linha define um esquema de validação utilizando o yup. O método object() cria um esquema para um objeto, e  
o método shape() permite definir as regras de validação para cada propriedade do objeto.  
  
  nome: yup.string()...  
6 // Aqui é definida a regra de validação para a propriedade nome do objeto. O método string() especifica que o  
valor deve ser uma string, e o método required() define que o campo é obrigatório. Se o valor estiver vazio, a  
mensagem de erro "O campo nome é de preenchimento obrigatório!" será exibida.  
  
7 As linhas restantes definem as regras de validação para as propriedades rua, número, complemento, bairro, cidade e  
estado, mas não definem que esses campos são obrigatórios.
```



# REACT – HOOK FORM

## ✓ Introdução ao Hook Form – Consumindo API externa

✓ Vamos agora, modificar nosso código.



```
File Edit Selection View Go Run Terminal Help
hookform [Administrator]

page.tsx x
src > app > Exemplo03 > page.tsx > Exemplo03 > inserirCliente
24 |   estado: yup.string()
25 | }).required();
26 |
27 | export default function Exemplo03(){
28 |
29 |   const {register, handleSubmit, formState: {errors}} = useForm({resolver: yupResolver(schema)});
30 |   const [listaClientes, setlistaClientes] = useState<any[]>([]);
31 |
32 |   function inserirCliente(cliente: any){
33 |     setlistaClientes([...listaClientes, cliente]);
34 |     console.log(listaClientes);
35 |   }
36 | }
```

# REACT – HOOK FORM

## ✓ Introdução ao Hook Form – Consumindo API externa

### ✓ Exemplo 03 – Analisando linha a linha o arquivo page.tsx

```
27 export default function Exemplo03(){
    // Declaração da função de componente React chamada Exemplo03 que será exportada como padrão do
    módulo.

28
    const {register, handleSubmit, formState: {errors}, , setValue, setFocus} = useForm({resolver: yupResolver(schema)});
    // Usa o hook useForm para inicializar as funcionalidades de gerenciamento de formulário. Ele retorna um objeto
    contendo várias funções e propriedades, incluindo register para registrar os campos de entrada do formulário,
29 handleSubmit para lidar com a submissão do formulário e formState.errors para acessar os erros de validação. O
    resolver é configurado com yupResolver(schema), onde schema é o esquema de validação definido pelo Yup, que será
    utilizado para validar os dados do formulário.

    const [listaClientes, setListaClientes] = useState<any[]>([]);
30 // Define o estado local listaClientes utilizando o useState, onde listaClientes é inicializado como um array
    vazio []. setListaClientes é uma função utilizada para atualizar o estado de listaClientes. O tipo any[] indica
    que listaClientes é um array que pode conter qualquer tipo de dado.

31
    function inserirCliente(cliente: any){
32 // Declaração da função inserirCliente, que recebe um parâmetro cliente de tipo any.

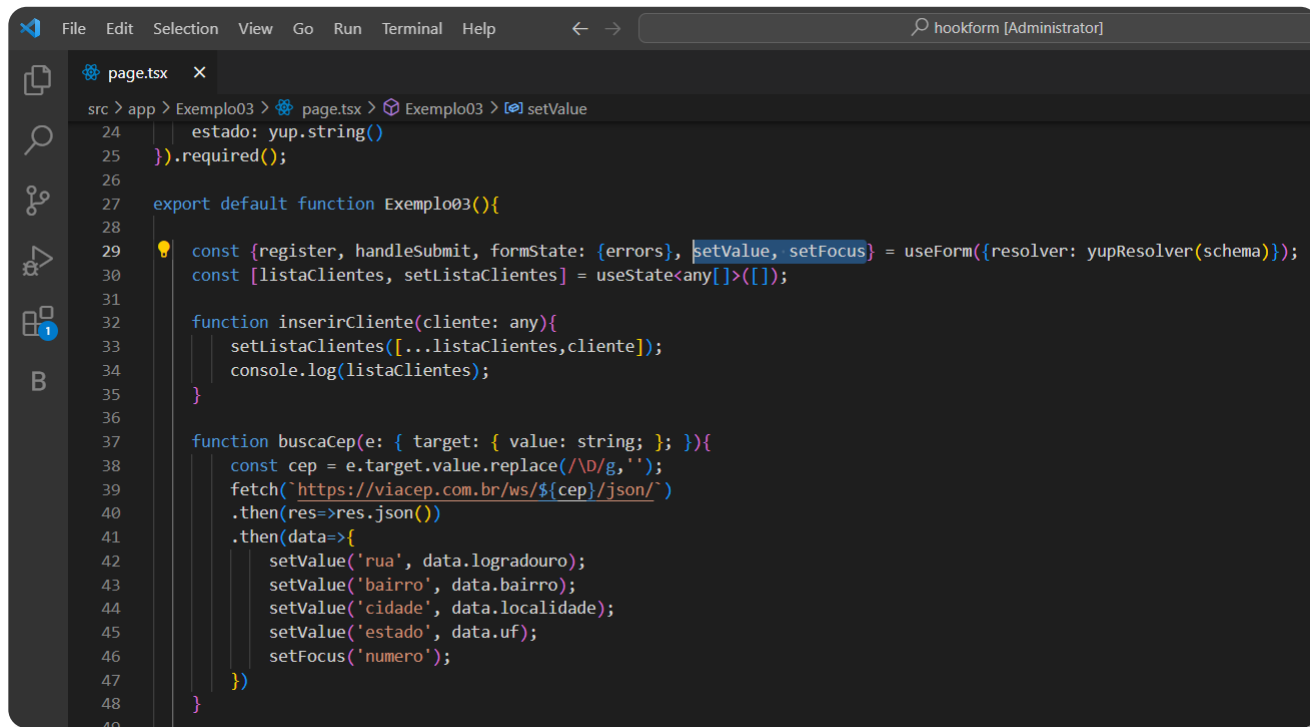
    setListaClientes([...listaClientes,cliente]);
33 // Utiliza setListaClientes para adicionar um novo cliente ao array listaClientes. O operador de propagação ... é
    utilizado para manter os itens anteriores do array listaClientes e adicionar o novo cliente no final.

34 console.log(listaClientes);
    // Exibe o array listaClientes no console do navegador para fins de depuração.

35 }
```

# REACT – HOOK FORM

- ✓ **Introdução ao Hook Form – Consumindo API externa**
- ✓ Vamos agora, modificar nosso código.



```
page.tsx X
src > app > Exemplo03 > page.tsx > Exemplo03 > setValue
24 |   estado: yup.string()
25 | }).required();
26 |
27 | export default function Exemplo03(){
28 |
29 |   const {register, handleSubmit, formState: {errors}, setValue, setFocus} = useForm({resolver: yupResolver(schema)});
30 |   const [listaClientes, setListaClientes] = useState<any[]>([]);
31 |
32 |   function inserirCliente(cliente: any){
33 |     setListaClientes([...listaClientes, cliente]);
34 |     console.log(listaClientes);
35 |   }
36 |
37 |   function buscaCep(e: { target: { value: string; }; }){
38 |     const cep = e.target.value.replace(/\D/g, '');
39 |     fetch(`https://viacep.com.br/ws/${cep}/json/`)
40 |       .then(res=>res.json())
41 |       .then(data=>{
42 |         setValue('rua', data.logradouro);
43 |         setValue('bairro', data.bairro);
44 |         setValue('cidade', data.localidade);
45 |         setValue('estado', data.uf);
46 |         setFocus('numero');
47 |       })
48 |   }
49 | }
```

# REACT – HOOK FORM

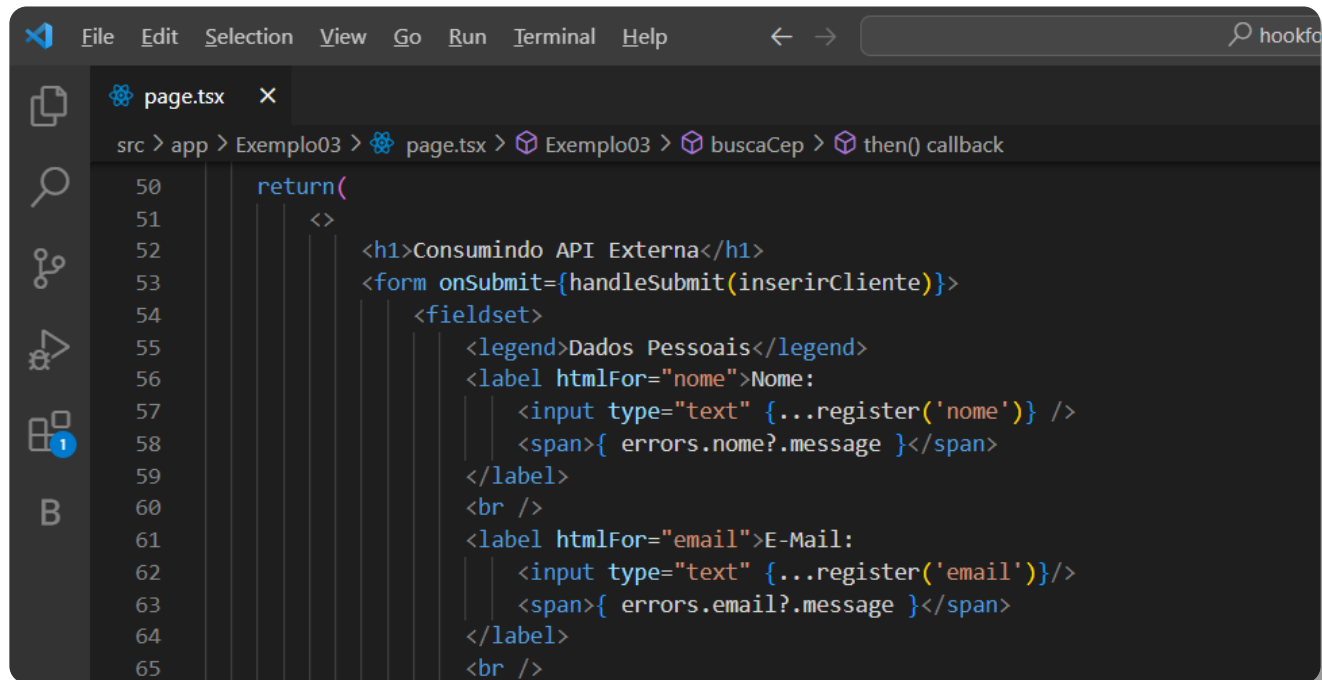
## ✓ Introdução ao Hook Form – Consumindo API externa

### ✓ Exemplo 03 – Analisando linha a linha o arquivo page.tsx

```
function buscaCep(e: { target: { value: string; }; }) {  
  // Define uma função chamada buscaCep que recebe um evento e como argumento. O evento e é esperado para ter uma  
  // propriedade target, que é um objeto que possui uma propriedade value do tipo string. Este padrão de tipos é usado  
  // para garantir que a função seja chamada apenas com um evento que tenha uma estrutura específica.  
  
  const cep = e.target.value.replace(/\D/g, "");  
  // Extrai o valor do CEP do evento e remove todos os caracteres que não são dígitos utilizando uma  
  // expressão regular (/D/g). Isso garante que apenas os dígitos do CEP sejam mantidos.  
  
  fetch('https://viacep.com.br/ws/${cep}/json/')  
  // Faz uma requisição HTTP GET para a API ViaCEP, passando o CEP como parte da URL. Esta API retorna  
  // informações de endereço no formato JSON com base no CEP fornecido.  
  
  .then(res=>res.json())  
  // Processa a resposta da requisição HTTP como JSON. O método json() retorna uma promise que resolve com os  
  // dados JSON da resposta.  
  
  .then(data=>{  
    // Manipula os dados retornados pela API ViaCEP. A função de retorno de chamada dentro deste método é  
    // executada quando os dados JSON são recebidos com sucesso.  
  
    setValue('rua', data.logradouro);  
  
    setValue('bairro', data.bairro);  
  
    setValue('cidade', data.localidade);  
  
    setValue('estado', data.uf);  
  
    setFocus('numero');  
  
  })  
}
```

# REACT – HOOK FORM

- ✓ **Introdução ao Hook Form – Consumindo API externa**
- ✓ Vamos agora, modificar nosso código.



```
File Edit Selection View Go Run Terminal Help
page.tsx
src > app > Exemplo03 > page.tsx > Exemplo03 > buscaCep > then() callback

50   return(
51     <>
52       <h1>Consumindo API Externa</h1>
53       <form onSubmit={handleSubmit(inserirCliente)}>
54         <fieldset>
55           <legend>Dados Pessoais</legend>
56           <label htmlFor="nome">Nome:
57             <input type="text" {...register('nome')} />
58             <span>{ errors.nome?.message }</span>
59           </label>
60           <br />
61           <label htmlFor="email">E-Mail:
62             <input type="text" {...register('email')} />
63             <span>{ errors.email?.message }</span>
64           </label>
65           <br />
```

# REACT – HOOK FORM

## ✓ Introdução ao Hook Form – Consumindo API externa

### ✓ Exemplo 03 – Analisando linha a linha o arquivo page.tsx

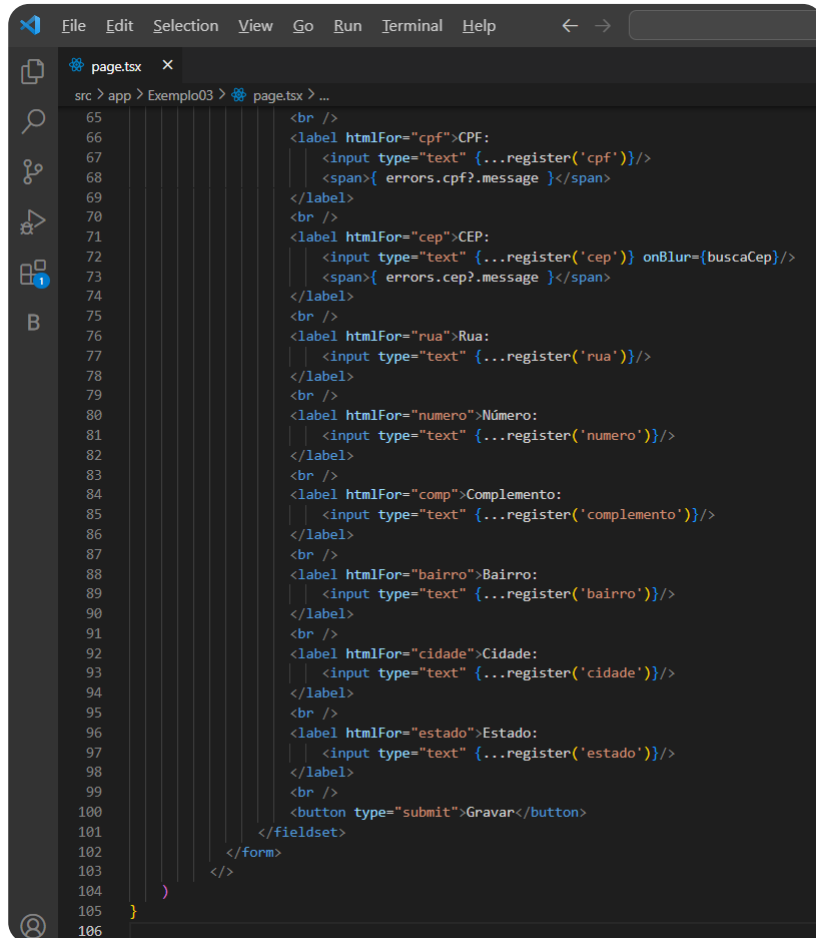
```

37   return(
38       <>
39       <h1>Consumindo API Externa</h1>
40       <form onSubmit={handleSubmit(inserirCliente)}>
41           // Este é um formulário React que será renderizado na interface do usuário. Ele possui um atributo onSubmit
42           // que define a função handleSubmit(inserirCliente) como o callback a ser chamado quando o formulário for
43           // submetido. A função handleSubmit é fornecida pelo useForm do React Hook Form e inserirCliente é uma função
44           // definida pelo usuário que será chamada quando o formulário for submetido.
45       <fieldset>
46           <legend>Dados Pessoais</legend>
47           <label htmlFor="nome">Nome:
48               <input type="text" {...register('nome')} />
49               // Este é um elemento HTML <input> que define um campo de entrada de texto. O atributo type
50               // define o tipo de entrada como texto. O ...register('nome') é uma propriedade que registra este
51               // campo de entrada no formulário React Hook Form e o associa ao campo 'nome' do schema de
52               // validação.
53               <span>{ errors.nome?.message }</span>
54               // Este é um elemento HTML <span> que será usado para exibir mensagens de erro relacionadas ao
55               // campo 'nome'. O conteúdo dentro das chaves {} é uma expressão JavaScript que acessa a
56               // propriedade message do objeto errors associado ao campo 'nome'. Se não houver erro, essa
57               // expressão resultará em null ou undefined, caso contrário, exibirá a mensagem de erro associada
58               // ao campo 'nome'.
59           </label>

```

# REACT – HOOK FORM

- ✓ **Introdução ao Hook Form – Consumindo API externa**
- ✓ Vamos agora, modificar nosso código.
- ✓ Faça o mesmo para:
  - ✓ CPF;
  - ✓ CEP;
  - ✓ RUA;
  - ✓ NÚMERO;
  - ✓ COMPLEMENTO;
  - ✓ BAIRRO;
  - ✓ CIDADE e
  - ✓ ESTADO



```
65 <br />
66 <label htmlFor="cpf">CPF:
67   <input type="text" {...register('cpf')} />
68   <span>{ errors.cpf?.message }</span>
69 </label>
70 <br />
71 <label htmlFor="cep">CEP:
72   <input type="text" {...register('cep')} onBlur={buscaCep} />
73   <span>{ errors.cep?.message }</span>
74 </label>
75 <br />
76 <label htmlFor="rua">Rua:
77   <input type="text" {...register('rua')} />
78 </label>
79 <br />
80 <label htmlFor="numero">Número:
81   <input type="text" {...register('numero')} />
82 </label>
83 <br />
84 <label htmlFor="comp">Complemento:
85   <input type="text" {...register('complemento')} />
86 </label>
87 <br />
88 <label htmlFor="bairro">Bairro:
89   <input type="text" {...register('bairro')} />
90 </label>
91 <br />
92 <label htmlFor="cidade">Cidade:
93   <input type="text" {...register('cidade')} />
94 </label>
95 <br />
96 <label htmlFor="estado">Estado:
97   <input type="text" {...register('estado')} />
98 </label>
99 <br />
100 <button type="submit">Gravar</button>
101 </fieldset>
102 </form>
103 </>
104 )
105 }
106
```

# REACT – HOOK FORM

## ✓ Introdução ao Hook Form – Consumindo API externa

### ✓ Exemplo 03 – Resultado

A screenshot of a web browser window. The browser's address bar shows 'localhost:3000/Exemplo03'. The page title is 'Hook Form'. The main heading of the page is 'Consumindo API Externa'. Below the heading is a form titled 'Dados Pessoais'. The form contains several input fields with associated validation messages:

- Nome:  O campo nome é de preenchimento obrigatório!
- E-Mail:  O campo email é de preenchimento obrigatório!
- CPF:  O CPF deve ter 11 dígitos!
- CEP:  O CEP deve ter 08 dígitos!
- Rua:
- Número:
- Complemento:
- Bairro:
- Cidade:
- Estado:

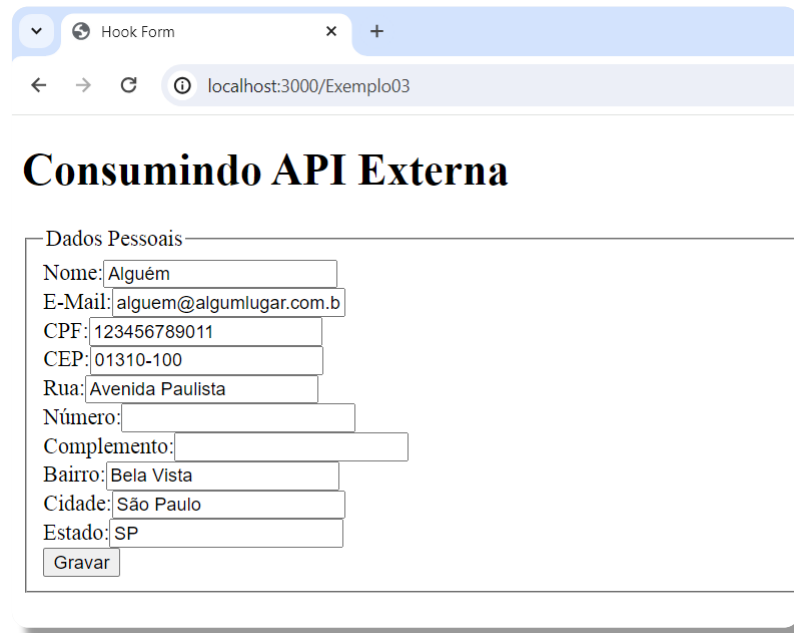
At the bottom of the form is a button labeled 'Gravar'.



# REACT – HOOK FORM

## ✓ Introdução ao Hook Form – Consumindo API externa

### ✓ Exemplo 03 – Resultado



A screenshot of a web browser window. The browser's address bar shows 'localhost:3000/Exemplo03'. The page title is 'Consumindo API Externa'. Below the title is a form titled 'Dados Pessoais'. The form contains several input fields with pre-filled or placeholder text: 'Nome: Alguém', 'E-Mail: alguem@algunlugar.com.b', 'CPF: 123456789011', 'CEP: 01310-100', 'Rua: Avenida Paulista', 'Número:', 'Complemento:', 'Bairro: Bela Vista', 'Cidade: São Paulo', and 'Estado: SP'. At the bottom of the form is a button labeled 'Gravar'.

# REACT – HOOK FORM

## ✓ Introdução ao Hook Form

### ✓ Exercício 01

- ✓ Crie um formulário com os seguintes campos:
  - ✓ **Email; Confirmação de Senha; Data de Nascimento; Número de Telefone; Endereço; Cidade; Estado; CEP e Gênero**
- ✓ Utilize o React Hook Form para registrar os campos do formulário.
- ✓ Utilize a biblioteca Yup para definir um esquema de validação para os dados do formulário. O esquema deve garantir que todos os campos sejam obrigatórios e que o CPF seja válido.
- ✓ Exiba mensagens de erro abaixo de cada campo do formulário, indicando os problemas de validação.
- ✓ Teste o formulário inserindo dados válidos e inválidos para verificar se a validação está funcionando corretamente.
- ✓ O envio do formulário não é necessário para esta atividade. O foco está na configuração do React Hook Form e Yup para criar e validar o formulário.

# REACT – HOOK FORM

## ✓ Introdução ao Hook Form

### ✓ Exercício 02

- ✓ Desenvolva um aplicativo de lista de tarefas em React. Crie dois componentes: ListaTarefas e Tarefa. O componente ListaTarefas receberá uma lista de objetos de tarefa como propriedade. Cada objeto de tarefa deve ter uma descrição e um status (completo ou não completo). O componente ListaTarefas deve renderizar várias instâncias do componente Tarefa, passando as propriedades apropriadas para cada uma delas.
- ✓ O componente Tarefa deve exibir a descrição da tarefa e um indicador visual de seu status. Adicione funcionalidade para alternar o status de uma tarefa (de completo para não completo e vice-versa) quando o usuário clicar na tarefa.

# REACT – HOOK FORM

## ✓ Introdução ao Hook Form

### ✓ Exercício 03

#### ✓ Objetivo

- ✓ O objetivo deste exercício é praticar o uso do React Hook Form para criar um formulário de busca de clima que consome uma API externa para obter dados meteorológicos.

#### ✓ Descrição

- ✓ Você foi contratado para desenvolver uma aplicação web que permita aos usuários verificar o clima de uma determinada cidade. A aplicação deve conter um formulário simples onde o usuário pode inserir o nome da cidade desejada. Ao enviar o formulário, a aplicação deve realizar uma requisição à API externa de clima e exibir os resultados na tela.

# REACT – HOOK FORM

## ✓ Introdução ao Hook Form

### ✓ Exercício 03

#### ✓ Tarefas

- ✓ Configure um projeto React com TypeScript.
  - ✓ Instale e configure o React Hook Form para gerenciar o formulário de busca.
  - ✓ Implemente um componente de formulário que contenha um campo de entrada para o nome da cidade e um botão de envio.
  - ✓ Utilize o React Hook Form para lidar com a validação do formulário e a submissão dos dados.
  - ✓ Faça uma requisição à API externa de clima, como OpenWeatherMap, utilizando o nome da cidade fornecido pelo usuário.
  - ✓ Exiba os dados de clima retornados pela API na tela de forma legível e organizada.
- 
- ✓ **Endereço da API:** <https://api.openweathermap.org/data/2.5/weather>
  - ✓ **Documentação API:** <https://openweathermap.org/current>

# FRONT-END DESIGN ENGINEERING

ANTONIO, C. Pro React: Build Complex Front-End Applications in a Composable Way With React. Apress, 2015.

BOSWELL, D; FOUCHER, T. The Art of Readable Code: Simple and Practical Techniques for Writing Better Code. Estados Unidos: O'Reilly Media, 2012.

BRITO, Robin Cris. Android Com Android Studio - Passo A Passo. Editora Ciência Moderna.

BUNA, S. React Succinctly. Estados Unidos: [s.n], 2016. Disponível em: <[www.syncfusion.com/ebooks/reactjs\\_succinctly](http://www.syncfusion.com/ebooks/reactjs_succinctly)>. Acesso em: 12 de janeiro de 2023.

FACEBOOK (2019a). React: Getting Started. React Docs, 2019. Disponível em: <[reactjs.org/docs/react-api.html](https://reactjs.org/docs/react-api.html)>. Acesso em: 13 de janeiro de 2023.

FACEBOOK (2019b). React Without ES6. React Docs, 2019. Disponível em: <[reactjs.org/docs/react-without-es6.html](https://reactjs.org/docs/react-without-es6.html)>. Acesso em: 10 de janeiro de 2023.

FACEBOOK (2019c). React Without JSX. React Docs, 2019. Disponível em: <[reactjs.org/docs/react-without-jsx.html](https://reactjs.org/docs/react-without-jsx.html)>. Acesso em: 10 de janeiro de 2023.

FORM BUILDER. Build your form with code and example. Disponível em <<https://react-hook-form.com/form-builder>> . Acessado em 25 de março de 2024.

FREEMAN, Eric ROBSON, Elisabeth. Use a Cabeça! Programação em HTML5. Rio de Janeiro: Editora Alta Books, 2014

GACKENHEIMER, C. Introduction to React: Using React to Build scalable and efficient user interfaces.[s.i.]: Apress, 2015.

HUDSON, P. Hacking with React. 2016. Disponível em: <[www.hackingwithreact.com/read/1/3/introduction-to-jsx](http://www.hackingwithreact.com/read/1/3/introduction-to-jsx)>. Acesso em: 13 janeiro de 2023.

KOSTRZEWA, D. Is React.js the Best JavaScript Framework in 2018? 2018. Disponível em: <[hackernoon.com/is-react-js-the-best-javascript-framework-in-2018-264a0eb373c8](https://hackernoon.com/is-react-js-the-best-javascript-framework-in-2018-264a0eb373c8)>. Acesso em: janeiro de 2023.

MARTIN, R. Clean Code: A Handbook of Agile Software Craftsmanship. Estados Unidos: Prentice Hall, 2009.

MDN WEB DOCS. Guia JavaScript. Disponível em <<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide>>. Acessado em 29 de janeiro de 2023.

NELSON, J. Learn React's Fundamentals Without the Buzzwords? 2018. Disponível em: <[jamesknelson.com/learn-react-fundamentals-sans-buzzwords](https://jamesknelson.com/learn-react-fundamentals-sans-buzzwords)>. Acesso em: 12 janeiro de 2023.

# FRONT-END DESIGN ENGINEERING

NIELSEN, J. Response Times: The 3 Important Limits. 1993. Disponível em: <[www.nngroup.com/articles/response-times-3-important-limits](http://www.nngroup.com/articles/response-times-3-important-limits)>. Acesso em: 10 janeiro de 2023.

O'REILLY, T. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. 2005. Disponível em: <[www.oreilly.com/pub/a/web2/archive/what-is-web-20.html#mememap](http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html#mememap)>. Acesso em: 10 de janeiro de 2023.

PANDIT, N. What Is ReactJS and Why Should We Use It? 2018. Disponível em: <[www.c-sharpcorner.com/article/what-and-why-reactjs](http://www.c-sharpcorner.com/article/what-and-why-reactjs)>. Acesso em: 12 de janeiro de 2023.

RAUSCHMAYER, A. Speaking JavaScript: An In-Depth Guide for Programmers. Estados Unidos: O'Reilly Media, 2014.

REACTIVA. O arquivo package-lock.json. Disponível em: <<https://nodejs.reativa.dev/0020-package-lock-json/index>>. Acessado em 13 de janeiro de 2023.

\_\_\_\_\_. O guia do package.json. Disponível em: <<https://nodejs.reativa.dev/0019-package-json/index>>. Acessado em 13 de janeiro de 2023.

RICOY, L. Desmitificando React: Uma Reflexão para Iniciantes. 2018. Disponível em: <[medium.com/trainingcenter/desmitificando-react-uma-reflex%C3%A3o-para-iniciantes-a57af90b6114](https://medium.com/trainingcenter/desmitificando-react-uma-reflex%C3%A3o-para-iniciantes-a57af90b6114)>. Acesso em: 13 janeiro de 2023.

SILVA, Maurício Samy. Ajax com jQuery: requisições Ajax com a simplicidade de jQuery. São Paulo: Novatec Editora, 2009.

\_\_\_\_\_. Construindo Sites com CSS e XHTML. Sites Controlados por Folhas de Estilo em Cascata. São Paulo: Novatec, 2010.

\_\_\_\_\_. CSS3 - Desenvolva aplicações web profissionais com o uso dos poderosos recursos de estilização das CSS. São Paulo: Novatec Editora, 2010.

STACKOVERFLOW. Most Popular Technologies: Web Frameworks. Developer Survey Results, StackOverflow, 2019. Disponível em: <[insights.stackoverflow.com/survey/2019#technology](https://insights.stackoverflow.com/survey/2019#technology)>. Acesso em: 13 de janeiro de 2023.

TYPESCRIPT SUPPORT. Disponível em <<https://react-hook-form.com/ts#UseFormProps>> . Acessado em 25 de março de 2024.

W3C. HTML5 - A linguagem de marcação que revolucionou a web. São Paulo: Novatec Editora, 2010.

\_\_\_\_\_. A vocabulary and associated APIs for HTML and XHTML. Disponível em <<https://www.w3.org/TR/2018/SPSD-html5-20180327/>>. Acessado em 28 de abril de 2020, às 20h53min.

# FRONT-END DESIGN ENGINEERING

W3C. Cascading Style Sheets, level 1. Disponível em <<https://www.w3.org/TR/2018/SPSD-CSS1-20180913/>>. Acessado em 28 de abril de 2020, às 21h58min.

\_\_\_\_\_. Cascading Style Sheets, level 2 Revision 2. Disponível em <<https://www.w3.org/TR/2016/WD-CSS22-20160412/>>. Acessado em 28 de abril de 2020, às 22h17min.

\_\_\_\_\_. Cascading Style Sheets, level 2. Disponível em <<https://www.w3.org/TR/2008/REC-CSS2-20080411/>>. Acessado em 28 de abril de 2020, às 22h03min.

\_\_\_\_\_. Cascading Style Sheets, level 3. Disponível em <<https://www.w3.org/TR/css-syntax-3/>>. Acessado em 28 de abril de 2020, às 22h18min.

\_\_\_\_\_. HTML 3.2 Reference Specification. Disponível em <<https://www.w3.org/TR/2018/SPSD-html32-20180315/>>. Acessado em 28 de abril de 2020, às 19h37min.

\_\_\_\_\_. HTML 4.0 Specification. Disponível em <<https://www.w3.org/TR/1998/REC-html40-19980424/>>. Acessado em 28 de abril de 2020, às 19h53min.

\_\_\_\_\_. HTML 4.01 Specification. Disponível em <<https://www.w3.org/TR/2018/SPSD-html401-20180327/>>. Acessado em 28 de abril de 2020, às 20h04min.

\_\_\_\_\_. Cascading Style Sheets, level 2 Revision 1. Disponível em <<https://www.w3.org/TR/CSS2/>>. Acessado em 28 de abril de 2020, às 22h13min.

WIKIPEDIA. JavaScript. Disponível em <<https://pt.wikipedia.org/wiki/JavaScript>>. Acessado em 29 de abril de 2020, às 10h.

YUP. Disponível em <<https://www.npmjs.com/package/yup>> . Acessado em 25 de março de 2024.