

PRÁTICAS AVANÇADAS EM DESENVOLVIMENTO WEB

Davi Schneid - davi.Schneid@gmail.com

16/07/2024

Agenda

- ▶ Consumir APIs externas
- ▶ Utilizar o pacote Axios
 - ▶ Introdução
 - ▶ Visitando <https://axios-http.com/>

Instalar npm install axios

- ▶ Executar o comando no terminal: `npm install axios`

```
PS C:\Users\SenacRs\PraticasAvancadasDesenvolvimentoWeb\api-web> npm install axios

added 3 packages, and audited 1549 packages in 8s

261 packages are looking for funding
  run `npm fund` for details

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

- ▶ Para usar o cliente HTTP, preciso importar na página
 - ▶ `import axios from 'axios';`

API Web

► Editar o arquivo Cadastro.js

```
14
15 function Cadastro() {
16
17   useEffect(() => {
18     axios.get('https://servicodados.ibge.gov.br/api/v1/localidades/estados')
19       .then(response => {
20         console.log(response.data);
21       })
22   }, []);
23
24   return (
```

Formulario de Cadastro

Dados de Cadastro

Nome:

Idade:

Cidade:

UF:

Selecione uma opção ▼

Salvar

Voltar

DevTools is now available in Portuguese!

Always match Chrome's language Switch DevTools to Portuguese Don't show again

Elements Console Sources Network Performance Memory >> Settings Help

top Filter Default levels No issues

react-dom.development.js:29895

Download the React DevTools for a better development experience:
<https://reactjs.org/link/react-devtools>

Array(27) Cadastro.js:20

- 0: {id: 11, sigla: 'RO', nome: 'Rondônia', regioao: {...}}
- 1: {id: 12, sigla: 'AC', nome: 'Acre', regioao: {...}}
- 2: {id: 13, sigla: 'AM', nome: 'Amazonas', regioao: {...}}
- 3: {id: 14, sigla: 'RR', nome: 'Roraima', regioao: {...}}
- 4: {id: 15, sigla: 'PA', nome: 'Pará', regioao: {...}}
- 5: {id: 16, sigla: 'AP', nome: 'Amapá', regioao: {...}}
- 6: {id: 17, sigla: 'TO', nome: 'Tocantins', regioao: {...}}
- 7: {id: 21, sigla: 'MA', nome: 'Maranhão', regioao: {...}}
- 8: {id: 22, sigla: 'PI', nome: 'Piauí', regioao: {...}}
- 9: {id: 23, sigla: 'CE', nome: 'Ceará', regioao: {...}}
- 10: {id: 24, sigla: 'RN', nome: 'Rio Grande do Norte', regioao: {...}}
- 11: {id: 25, sigla: 'PB', nome: 'Paraíba', regioao: {...}}
- 12: {id: 26, sigla: 'PE', nome: 'Pernambuco', regioao: {...}}
- 13: {id: 27, sigla: 'AL', nome: 'Alagoas', regioao: {...}}
- 14: {id: 28, sigla: 'SE', nome: 'Sergipe', regioao: {...}}
- 15: {id: 29, sigla: 'BA', nome: 'Bahia', regioao: {...}}
- 16: {id: 31, sigla: 'MG', nome: 'Minas Gerais', regioao: {...}}
- 17: {id: 32, sigla: 'ES', nome: 'Espírito Santo', regioao: {...}}
- 18: {id: 33, sigla: 'RJ', nome: 'Rio de Janeiro', regioao: {...}}
- 19: {id: 35, sigla: 'SP', nome: 'São Paulo', regioao: {...}}
- 20: {id: 41, sigla: 'PR', nome: 'Paraná', regioao: {...}}
- 21: {id: 42, sigla: 'SC', nome: 'Santa Catarina', regioao: {...}}
- 22: {id: 43, sigla: 'RS', nome: 'Rio Grande do Sul', regioao: {...}}
- 23: {id: 50, sigla: 'MS', nome: 'Mato Grosso do Sul', regioao: {...}}
- 24: {id: 51, sigla: 'MT', nome: 'Mato Grosso', regioao: {...}}
- 25: {id: 52, sigla: 'GO', nome: 'Goiás', regioao: {...}}
- 26: {id: 53, sigla: 'DF', nome: 'Distrito Federal', regioao: {...}}

length: 27

[[Prototype]]: Array(0)

API Web

- ▶ Renderizar os valores carregados no campo
- ▶ Editar o arquivo Cadastro.js adicionando o código abaixo.

```
src > paginas > JS Cadastro.js > ...
4   import '../App.css';
5
6   import BotaoVoltar from '../componentes/BotaoVoltar';
7
8   //Importar o cliente HTTP para requisicoes api externas
9   import axios from 'axios';
10
11  //Utilizada para auxiliar no controle de outras funcoes da aplicacao
12  import { useEffect, useState } from 'react';
13
```

```
14
15  function Cadastro() {
16
17    const [estados, setEstados] = useState([]);
18
19    useEffect(() => {
20      axios.get('https://servicodados.ibge.gov.br/api/v1/localidades/estados')
21        .then(response => {
22          setEstados(response.data);
23        })
24    }, []);
25
26    return (
```

```
    <div>
      <label>UF:
        <select name="cmbUF" id="cmbUF" >
          <option value="0">Selecione uma opção</option>
          {estados.map(estado => (<option key={estado.sigla} value={estado.sigla}>{estado.sigla}</option>))}
        </select>
      </label>
    </div>
```

API Web

- ▶ Ao recarregar a página de cadastro.
- ▶ As UF serão apresentadas no combobox

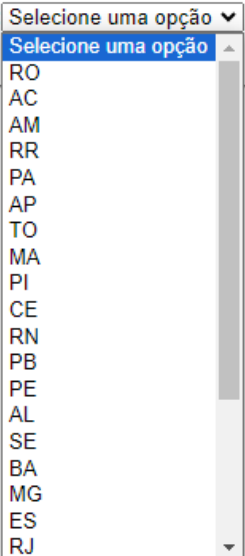
Formulario de Cadastro

Dados de Cadastro

Nome:

Idade:

Cidade:

UF: 

The dropdown menu for UF is open, displaying a list of Brazilian states. The list starts with 'Selezione uma opção' at the top, followed by the following states: RO, AC, AM, RR, PA, AP, TO, MA, PI, CE, RN, PB, PE, AL, SE, BA, MG, ES, and RJ. The list is scrollable, with a scrollbar visible on the right side.

API Router

- ▶ Abrir o Mysql Workbench
- ▶ Alterar a tabela usuários no banco de dados adicionando a coluna UF

The screenshot displays the MySQL Workbench interface with the 'API' database selected. The 'Navigator' pane on the left shows the 'api' database structure, with the 'usuarios' table highlighted. A right-click context menu is open over the 'usuarios' table, and the 'Alter Table...' option is selected. The 'Apply SQL Script to Database' dialog is open, showing the SQL script to be applied: `ALTER TABLE `api`.`usuarios` ADD COLUMN `uf` VARCHAR(2) NULL AFTER `updatedAt`;`. The 'Apply' button in the dialog is highlighted. At the bottom left, the 'Administration' tab is active, and the 'Schemas' sub-tab is selected.

MySQL Workbench

API x

File Edit View Query Database Server Tools

Navigator

SCHEMAS

Filter objects

api

Tables

usuarios

Select Rows - Limit 1000

Table Inspector

Copy to Clipboard

Table Data Export Wizard

Table Data Import Wizard

Send to SQL Editor

Create Table...

Create Table Like...

Alter Table...

Table Maintenance...

Drop Table...

Truncate Table...

Search Table Data...

Refresh All

Apply SQL Script to Database

Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

Algorithm: Default Lock Type: Default

```
1 ALTER TABLE `api`.`usuarios`
2 ADD COLUMN `uf` VARCHAR(2) NULL AFTER `updatedAt`;
3
```

Back Apply Cancel

Type: VARCHAR(2)

Default: NULL

Storage: ☐ Virtual ☐ Stored

☐ Primary Key ☐ Not Null ☐ Unique

☐ Binary ☐ Unsigned ☐ Zero Fill

☐ Auto Increment ☐ Generated

Apply Revert

Administration Schemas

API Router

- ▶ Alterar modelo Usuario.js incluindo novo campo UF.
- ▶ Alterar controller UsuarioController.js incluindo no campo UF

```
const Usuario = database.define('usuario', {
  id: {
    type: Sequelize.INTEGER,
    autoIncrement: true,
    allowNull: false,
    primaryKey: true
  },
  nome: {
    type: Sequelize.STRING,
    allowNull: false
  },
  idade: {
    type: Sequelize.INTEGER,
    allowNull: false
  },
  cidade: {
    type: Sequelize.STRING,
    allowNull: false
  },
  uf: {
    type: Sequelize.STRING,
    allowNull: true
  }
}, {
```

```
8
9 // Criar um novo usuário
10 exports.createusuario = async (req, res) => {
11   console.log('createusuario');
12   const { nome, idade, cidade, uf } = req.body;
13   console.log('Createusuario.Nome'+nome);
14   console.log('createusuario.Idade'+idade);
15   console.log('createusuario.Cidade'+cidade);
16   console.log('createusuario.UF'+uf);
17   try {
18     const novoUsuario = await Usuario.create({ nome, idade, cidade, uf });
19     res.status(201).json(novoUsuario);
20   } catch (err) {
21     console.log("Erro ao criar usuário");
22     res.status(500).json({ error: 'Erro ao criar usuário' });
23   }
24 };
```

```
36 // Atualizar um usuário
37 exports.updateusuario = async (req, res) => {
38   const { id } = req.params;
39   const { nome, idade, cidade, uf } = req.body;
40   console.log('updateusuario id: '+id+ ' - nome: '+nome+ ' - idade: '+idade+ ' - cidade: '+cidade+ ' - uf: '+uf);
41   try {
42     const usuario = await Usuario.findByPk(id);
43     if (usuario) {
44       usuario.nome = nome;
45       usuario.idade = idade;
46       usuario.cidade = cidade;
47       usuario.uf = uf;
48       usuario.updatedAt = new Date();
49       await usuario.save();
50       res.status(200).json(usuario);
51     } else {
52       res.status(404).json({ error: 'Usuário não encontrado' });
53     }
54   } catch (err) {
55     res.status(500).json({ error: 'Erro ao atualizar usuário' });
56   }
57 };
```


API Web

- ▶ Editar o Cadastro.js adicionando useEffect no import

```
//Utilizada para auxiliar no controle de outras funcoes da aplicacao
import React, { useState, useEffect } from 'react';
```

- ▶ Adicionar o campo uf no useState setCampos

```
//cria novo estado para os campos da tela
const [campos, setCampos] = useState({
  nome: '',
  idade: 0,
  cidade: '',
  uf: ''
});
```

- ▶ Iniciar o useState do array de estados

```
const [estados, setEstados] = useState([]);

useEffect(() => {
  axios.get('https://servicodados.ibge.gov.br/api/v1/localidades/estados')
    .then(response => {
      setEstados(response.data);
    })
}, []);
```

- ▶ Adicionar a validação do campo UF na função validarCampos

API Web

► Exercício

- Atualizar a página EditarRegistro.js adicionando o campo UF, validação e etc.
- Atualizar a página ListaRegistro.js, adicionando o campo UF.

API Web

- Adicionar a busca por CEP na aplicação, atualizando a página cadastro

```
//cria novo estado para os campos da tela
const [campos, setCampos] = useState({
  nome: '',
  idade: 0,
  cidade: '',
  uf: '',
  cep: '',
  complemento: '',
  bairro: '',
  numero:0
});
```

Passo 1

```
// Limpar os campos do formulário após o envio
setCampos({
  nome: '',
  idade: 0,
  cidade: '',
  uf: '',
  cep: '',
  complemento: '',
  bairro: '',
  numero:0
});
```

Passo 3

```
function buscarEnderecoPorCEP() {
  const cep = campos.cep.replace(/\D/g, '');
  if (cep.length === 8) {
    axios.get(`https://viacep.com.br/ws/${cep}/json/`)
      .then(response => {
        if (response.data.erro) {
          setErros(prevErros => ({ ...prevErros, cep: 'CEP inválido' }));
        } else {
          setCampos(prevCampos => ({
            ...prevCampos,
            cidade: response.data.localidade,
            complemento: response.data.complemento,
            uf: response.data.uf,
            bairro: response.data.bairro,
            logradouro: response.data.logradouro,
            complemento: response.data.complemento
          }));
        }
      })
      .catch(error => {
        setErros(prevErros => ({ ...prevErros, cep: 'Erro ao buscar CEP' }));
      });
  }
}
```

Passo 2

```
function handleInputChange(event) {
  const { name, value } = event.target;
  setCampos(prevCampos => ({
    ...prevCampos,
    [name]: value
  }));

  setErros(prevErros => ({
    ...prevErros,
    [name]: ''
  }));
}
```

Passo 4

API Web

- Atualizar o html com os novos campos

```
<div className="inline-fields">
  <div className="field-maior">
    <label>Nome:
      <input type="text" name="nome" id="nome" value={campos.nome} onChange={handleInputChange} />
      {erros.nome && <p className="error">{erros.nome}</p>}
    </label>
  </div>

  <div className="field-menor">
    <label>Idade:
      <input type="number" name="idade" id="idade" value={campos.idade} onChange={handleInputChange} />
      {erros.idade && <p className="error">{erros.idade}</p>}
    </label>
  </div>
</div>

<div className="inline-fields">
  <div className="field-menor">
    <label>CEP:
      <input type="text" name="cep" id="cep" value={campos.cep} onChange={handleInputChange} onBlur={buscarEnderecoPorCEP} />
      {erros.cep && <p className="error">{erros.cep}</p>}
    </label>
  </div>

  <div className="field-maior">
    <label>Cidade:
      <input type="text" name="cidade" id="cidade" value={campos.cidade} onChange={handleInputChange} />
      {erros.cidade && <p className="error">{erros.cidade}</p>}
    </label>
  </div>
</div>
```

API Web

- Atualizar o html com os novos campos

```
<div className="inline-fields">
  <div className="field-maior">
    <label>Bairro:
      <input type="text" name="bairro" id="cidabairrode" value={campos.bairro} onChange={handleInputChange} />
      {erros.bairro && <p className="error">{erros.bairro}</p>}
    </label>
  </div>

  <div className="field-menor">
    <label>Complemento:
      <input type="text" name="complemento" id="complemento" value={campos.complemento} onChange={handleInputChange} />
      {erros.complemento && <p className="error">{erros.complemento}</p>}
    </label>
  </div>
</div>

<div>
  <div className="inline-fields">
    <div className="field-maior">
      <label>Logradouro:
        <input type="text" name="logradouro" id="logradouro" value={campos.logradouro} onChange={handleInputChange} />
        {erros.logradouro && <p className="error">{erros.logradouro}</p>}
      </label>
    </div>

    <div className="field-menor">
      <label>Número:
        <input type="number" name="numero" id="numero" value={campos.numero} onChange={handleInputChange} />
        {erros.numero && <p className="error">{erros.numero}</p>}
      </label>
    </div>
  </div>
</div>
```

API Web

- Atualizar o html com os novos campos

```
<div className="form-group">
  <label>UF:
    <select name="uf" id="uf" value={campos.uf} onChange={handleInputChange}>
      <option value="0">Selecione uma opção</option>
      {estados.map(estado => (<option key={estado.sigla} value={estado.sigla}>{estado.sigla}</option>))}
    </select>
  </label>
  <label>
    <div>
      {erros.uf && <p className="error">{erros.uf}</p>}
    </div>
  </label>
</div>
```

- Atualizar o arquivo App.css com o código que está no github.

API Web

- ▶ Testar o envio dos campos para o backend
- ▶ Preencher os dados do formulário e inspecionar no navegador.

Formulario de Cadastro

Dados de Cadastro

Nome:	<input type="text"/>	Idade:	<input type="text" value="0"/>
CEP:	<input type="text"/>	Cidade:	<input type="text"/>
Bairro:	<input type="text"/>	Complemento:	<input type="text"/>
Logradouro:	<input type="text" value="Avenida João Gomes Nogueira"/>	Número:	<input type="text" value="0"/>
UF:	<div>Selecione uma opção ▼</div>		
<div>Salvar</div>			
<div>Voltar</div>			

Inspecionar no navegador os campos
estão sendo enviados para o backend



```
Submetendo: Cadastro.js:108
{nome: 'Jon Stewart Doe', idade: '12', cidade: 'Pelotas', uf: 'RS', cep: '9608520', ...}
  bairro: "Areal"
  cep: "96085200"
  cidade: "Pelotas"
  complemento: "casa"
  idade: "12"
  logradouro: "Avenida João Gomes Nogueira"
  nome: "Jon Stewart Doe"
  numero: "010"
  uf: "RS"
▶ [[Prototype]]: Object
```

APIRouter

- Na api backend, alterar o modelo Usuario.js incluindo novos campos.

```
5  const Usuario = database.define('usuario', {
6    id: {
7      type: Sequelize.INTEGER,
8      autoIncrement: true,
9      allowNull: false,
10     primaryKey: true
11   },
12   nome: {
13     type: Sequelize.STRING,
14     allowNull: false
15   },
16   idade: {
17     type: Sequelize.INTEGER,
18     allowNull: false
19   },
20   cidade: {
21     type: Sequelize.STRING,
22     allowNull: false
23   },
24   uf: {
25     type: Sequelize.STRING,
26     allowNull: true
27   },
28   cep: {
29     type: Sequelize.STRING,
30     allowNull: true
31   },
32   complemento: {
33     type: Sequelize.STRING,
34     allowNull: true
35   },
36   bairro: {
37     type: Sequelize.STRING,
38     allowNull: true
39   },
40   numero: {
41     type: Sequelize.INTEGER,
42     allowNull: true
43   },
44 }, {
45
```


APIRouter

- ▶ Alterar controller UsuarioController.js incluindo novos campos.

```
9 // Criar um novo usuário
10 exports.createusuario = async (req, res) => {
11   console.log('createusuario');
12   const { nome, idade, cidade, uf, cep, complemento, bairro, numero } = req.body;
13   console.log('Createusuario.Nome'+nome);
14   console.log('createusuario.Idade'+idade);
15   console.log('createusuario.Cidade'+cidade);
16   console.log('createusuario.UF'+uf);
17   console.log('createusuario.CEP'+cep);
18   console.log('createusuario.Complemento'+complemento);
19   console.log('createusuario.Bairro'+bairro);
20   console.log('createusuario.Numero'+numero);
21   try {
22     const novoUsuario = await Usuario.create({ nome, idade, cidade, uf, cep, complemento, bairro, numero});
23     res.status(201).json(novoUsuario);
24   } catch (err) {
25     console.log("Erro ao criar usuário");
26     res.status(500).json({ error: 'Erro ao criar usuário' });
27   }
28 };
29
```

```
40 // Atualizar um usuário
41 exports.updateusuario = async (req, res) => {
42   const { id } = req.params;
43   const { nome, idade, cidade, uf, cep, complemento, bairro, numero } = req.body;
44   console.log("updateusuario id:"+id+ " - nome:"+nome+ " - idade:"+idade+ " - cidade:"+cidade+ " - uf:"+uf+ " - cep:"+cep+ " - complemento:"+complemento+ " - bairro:"+bairro+ " - numero:"+numero);
45   try {
46     const usuario = await Usuario.findByPk(id);
47     if (usuario) {
48       usuario.nome = nome;
49       usuario.idade = idade;
50       usuario.cidade = cidade;
51       usuario.uf = uf;
52       usuario.updatedAt = new Date();
53       await usuario.save();
54       res.status(200).json(usuario);
55     } else {
56       res.status(404).json({ error: 'Usuário não encontrado' });
57     }
58   } catch (err) {
59     res.status(500).json({ error: 'Erro ao atualizar usuário' });
60   }
61 };
62
```

Mysql Workbench

- Alterar tabela com os novos campos.

usuarios - Table usuarios usuarios - Table x

Table Name: usuarios Schema: api

Charset/Collation: utf8mb4 utf8mb4_0900_ai_ci Engine: InnoDB

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
nome	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
idade	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
cidade	VARCHAR(255)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
uf	VARCHAR(2)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
cep	VARCHAR(10)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
complemento	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
bairro	VARCHAR(150)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
numero	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
createdAt	DATETIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
updatedAt	DATETIME	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Name:

Charset/Collation: Default Charset Default Collation

Comments:

Data Type:

Default:

Storage: ☐ Virtual ☐ Stored

☐ Primary Key ☐ Not Null ☐ Unique

☐ Binary ☐ Unsigned ☐ Zero Fill

☐ Auto Increment ☐ Generated

Columns Indexes Foreign Keys Triggers Partitioning Options

Apply Revert

Output

Terminar

- ▶ Iniciar a APIRouter.
- ▶ Testar o cadastro com os novos campos.
- ▶ Alterar a listagem de registros incluindo nos novos campos.
- ▶ Alterar a edição incluindo nos novos campos.