

PRÁTICAS AVANÇADAS EM DESENVOLVIMENTO WEB

Davi Schneid - davi.schneid@gmail.com

08/08/2024

Agenda

- ▶ Bcryptjs criptografia de senhas.
- ▶ Jsonwebtoken
- ▶ Registrar senha do usuário criptografada.
- ▶ Criar login com Token no backend.
- ▶ Criar tela de login no frontend.
- ▶ Permitir acesso ao sistema após login.
- ▶ Ajustar tela de cadastro, incluindo campo senha.
- ▶ Ajustar banco de dados, incluindo campo senha.
- ▶ Dependências
 - ▶ `npm install bcryptjs`
 - ▶ `npm install jsonwebtoken`
 - ▶ `npm install jwt-decode`
 - ▶ `npm install @fortawesome/react-fontawesome @fortawesome/free-solid-svg-icons @fortawesome/fontawesome-svg-core styled-components`

Bcryptjs

- ▶ Hashing de Senhas
- ▶ Salt Generation
- ▶ Verificação de Senhas
- ▶ Cross-Platform



BcryptJs

Jsonwebtoken

- ▶ Assinatura de Tokens
- ▶ Verificação de Tokens
- ▶ Decodificação de Tokens

APIRouter e APIWEB

- ▶ Instalar back bcryptjs: `npm install bcryptjs`

```
PS C:\Users\SenacRs\PraticasAvancadasDesenvolvimentoWeb\ApiRouter> npm install bcryptjs

added 1 package, and audited 167 packages in 3s

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

found 0 vulnerabilities
```

- ▶ Instalar **back e no front** jsonwebtoken: `npm install jsonwebtoken`

```
PS C:\Users\SenacRs\PraticasAvancadasDesenvolvimentoWeb\ApiRouter> npm install jsonwebtoken

added 12 packages, and audited 179 packages in 4s

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

found 0 vulnerabilities
```

- ▶ Instalar no **back e no front** jwt-decode: `npm install jwt-decode`

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

added 1 package, and audited 1569 packages in 8s

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

8 vulnerabilities (2 moderate, 6 high)

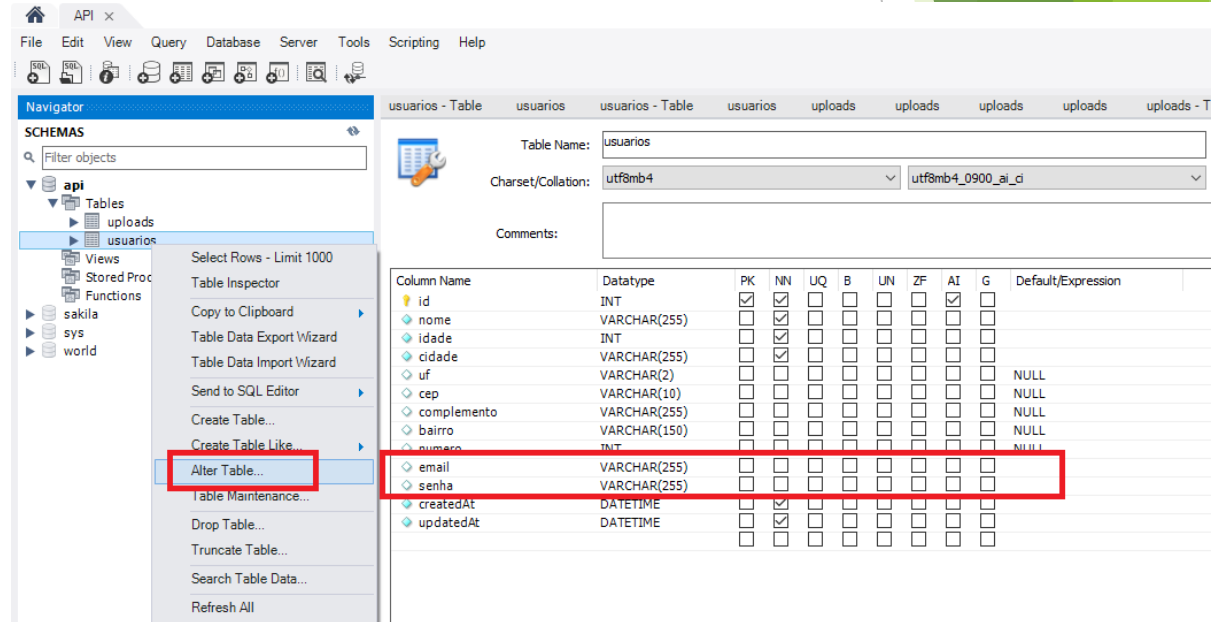
To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

APIRouter

- ▶ Alterar o modelo usuário, adicionando dois novos campos:
 - ▶ Email e Senha
 - ▶ Alterar o banco de dados incluindo novos campos

```
40     numero: {
41         type: Sequelize.INTEGER,
42         allowNull: true
43     },
44     email: {
45         type: Sequelize.STRING,
46         allowNull: true
47     },
48     senha: {
49         type: Sequelize.STRING,
50         allowNull: true
51     }
52 }, {
53
```

[illegible]

APIRouter

- ▶ Alterar o usuarioController.js, adicionando dois novos campos:
 - ▶ Alterar o createusuario e incluindo: Email, Senha
 - ▶ Criar a função getHashedPassword

```
// Criar um novo usuário
exports.createusuario = async (req, res) => {
  console.log('createusuario');
  const { nome, idade, cidade, uf, cep, logradouro, complemento, bairro, numero, email, senha } = req.body;
  console.log('Createusuario.Nome'+nome);
  console.log('createusuario.Idade'+idade);
  console.log('createusuario.Cidade'+cidade);
  console.log('createusuario.UF'+uf);
  console.log('createusuario.CEP'+cep);
  console.log('createusuario.Logradouro'+logradouro);
  console.log('createusuario.Complemento'+complemento);
  console.log('createusuario.Bairro'+bairro);
  console.log('createusuario.Numero'+numero);
  console.log('createusuario.Numero'+email);

  const hashedPassword = getHashedPassword(senha);

  try {
    const novoUsuario = await Usuario.create({ nome, idade, cidade, uf, cep, logradouro, complemento, bairro, numero, email, senha:hashedPassword});
    res.status(201).json(novoUsuario);
  } catch (err) {
    console.log("Erro ao criar usuário",err);
    res.status(500).json({ error: 'Erro ao criar usuário' });
  }
};
```

```
//importar o modulo de criptografia
const bcrypt = require('bcryptjs');

//importar o modulo de Web Token
const jwt = require('jsonwebtoken');
```

```
132 function getHashedPassword(senha) {
133   console.log('getHashedPassword',senha);
134   // valor 10 e o valor do custo para gerar o hash
135   const salt = bcrypt.genSaltSync(10);
136   const hashedPassword = bcrypt.hashSync(senha, salt);
137   console.log('getHashedPassword.hashedPassword:',hashedPassword);
138   return hashedPassword;
139 }
140
```

APIRouter

- ▶ Alterar o usuarioRota.js, adicionando dois novos campos no Swagger:
 - ▶ Email, Senha

```
11 //criar a rota criar usuario
12 /**
13  * @swagger
14  * /usuarios:
15  *   post:
16  *     summary: Cria um novo usuário
17  *     tags: [Usuario]
18  *     requestBody:
19  *       required: true
20  *       content:
21  *         application/json:
22  *           schema:
23  *             type: object
24  *             properties:
25  *               nome:
26  *                 type: string
27  *               idade:
28  *                 type: integer
29  *               cidade:
30  *                 type: string
31  *               uf:
32  *                 type: string
33  *                 minLength: 2
34  *                 maxLength: 2
35  *               cep:
36  *                 type: string
37  *               complemento:
38  *                 type: string
39  *               bairro:
40  *                 type: string
41  *               numero:
42  *                 type: integer
43  *               email:
44  *                 type: string
45  *                 description: O email do usuário
46  *               senha:
47  *                 type: string
48  *                 description: A senha do usuário
49  *     responses:
50  *       201:
51  *         description: Usuario criado
52  *       500:
53  *         description: Erro ao criar usuario
54  */
55 router.post('/usuarios', usuarioController.createusuario);
56
```


APIRouter

- Adicionar a função login no usuarioController.js

```
142 // Efetua o login do usuario
143 exports.login = async (req, res) => {
144
145     const { email, senha } = req.body;
146
147     console.log(['login',email]);
148     try {
149         const usuario = await Usuario.findOne({ where: { email } });
150         console.log('Usuario....:',usuario);
151         if (!usuario || usuario==null) {
152             console.log('Usuario nao encontrado',usuario.email);
153             return res.status(400).send('Dados incorretos - cod 001!');
154         }
155         else{
156             console.log('Usuario.email econtrado:',usuario.email);
157             const isPasswordValid = bcrypt.compareSync(senha, usuario.senha);
158
159             if (!isPasswordValid) {
160                 console.log('Dados incorretos - cod 002!');
161                 return res.status(400).send('Dados incorretos!');
162             }
163             const token = jwt.sign({ usuarioId: usuario.id }, process.env.JWT_KEY);
164             res.send({ token });
165         }
166     } catch (err) {
167
168         console.log('Erro no login',err);
169         res.status(400).send('Erro no login : ' + err.message);
170     }
171 };
172
173
```

APIRouter

- Adicionar a rota login dentro do usuarioRotas.js

```
207 //cria a rota de login
208 /**
209  * @swagger
210  * /login:
211  *   post:
212  *     summary: Autentica um usuário
213  *     tags: [Autenticação]
214  *     requestBody:
215  *       required: true
216  *       content:
217  *         application/json:
218  *           schema:
219  *             type: object
220  *             properties:
221  *               email:
222  *                 type: string
223  *                 description: O email do usuário
224  *               senha:
225  *                 type: string
226  *                 description: A senha do usuário
227  *     responses:
228  *       200:
229  *         description: Login bem-sucedido, retorna o token JWT
230  *         content:
231  *           application/json:
232  *             schema:
233  *               type: object
234  *               properties:
235  *                 token:
236  *                   type: string
237  *       400:
238  *         description: Usuário não encontrado ou senha inválida
239  *       500:
240  *         description: Erro ao fazer login
241  */
242 router.post('/login', usuarioController.login);
243
244
245 //exporta as rotas criadas
246 module.exports = router;
```

APIRouter

- ▶ Criar o gerador de chave JWT
- ▶ Criar a pasta geradorSecret na raiz do projeto
 - ▶ Criar o arquivo gerarChavesJWT.js, escrever o código abaixo.

```
geradorSecret > JS gerarChavesJWT.js > ...  
1  
2  const crypto = require('crypto');  
3  const jwtSecret = crypto.randomBytes(64).toString('hex');  
4  console.log(jwtSecret);|
```

- ▶ Executar o arquivo gerarChavesJWT.js com o comando: node gerarChavesJWT.js
 - ▶ Abrir novo terminal
 - ▶ cd geradorSecret
 - ▶ Executar comando: node gerarChaveJWT.js
 - ▶ Copiar a chave gerada

```
● PS C:\Users\SenacRs\PraticasAvancadasDesenvolvimentoWeb\ApiRouter\geradorSecret> node gerarChavesJWT.js  
519d4a6e0b5d4bd284c8a58bbf0fb200192ca7f0e656d6a1cb60ef48687553cf518ce0e2ff7d7f81891db473fcd697f803b4b12f3c3c35ff5a9bbc238d98a7be  
● PS C:\Users\SenacRs\PraticasAvancadasDesenvolvimentoWeb\ApiRouter\geradorSecret>
```

APIRouter

- ▶ Alterar o .env, adicionando a chave gerada JWT:
 - ▶ Criar novo parâmetro JWT_KEY e colar a chave gerada

```
.env
1  # The webapi port. Ex: 3000
2  PORT=3001
3
4  #The Connection String to database. Ex: mysql://user:password@server:port/database
5  CONNECTION_STRING=mysql://root:12qwaszx@localhost:3306/API
6
7
8  JWT_KEY=519d4a6e0b5d4bd284c8a58bbf0fb200192ca7f0e656d6a1cb60ef48687553cf518ce0e2ff7d7f81891db473fcd697f803b4b12f3c3c35ff5a9bbc238d98a7be
9
```

APIRouter

- Criar o arquivo tokenController.js na pasta controller, escrever o código abaixo:

```
controllers > JS tokenController.js > validaToken > validaToken
1
2  const jwt = require('jsonwebtoken');
3
4  exports.validaToken = async (req, res) => {
5
6      const { token } = req.body;
7      console.log('Validar Token',token);
8      try{
9          if (!token) {
10             console.log('Retorna http 400');
11             res.status(400).json({ valid: false });
12          }else{
13             jwt.verify(token, process.env.JWT_KEY, (err, decoded) => {
14                 if (err) {
15                     console.log('Retorna http 401');
16                     res.status(401).json({ valid: false });
17                 }else{
18                     console.log('Retorna http 200');
19                     res.status(200).json({ valid: true });
20                 }
21             });
22         }
23     }catch(err){
24         console.log('Erro ao validar token',err);
25     }
26 };
```

APIRouter

- Criar o arquivo tokenRotas.js na pasta rotas, escrever o código abaixo:

```
rotas > JS tokenRotas.js > tokenController
1
2 //Importa o modulo Express
3 const express = require('express');
4
5 //Cria o objeto rotas
6 const router = express.Router();
7
8 //Importa o modulo tokenController
9 const tokenController = require('../controllers/tokenController');
10
11
12 /**
13  * @swagger
14  * components:
15  *   schemas:
16  *     Token:
17  *       type: object
18  *       required:
19  *         - token
20  *       properties:
21  *         token:
22  *           type: string
23  *           description: JWT token
24  *       example:
25  *         token: 'your_jwt_token'
26  */
27
```

APIRouter

- Continuação do o arquivo tokenRotas.js

```
28  /**
29  * @swagger
30  * /validarToken:
31  *   post:
32  *     summary: Verifica a validade do token JWT
33  *     tags: [Token]
34  *     requestBody:
35  *       required: true
36  *       content:
37  *         application/json:
38  *           schema:
39  *             $ref: '#/components/schemas/Token'
40  *     responses:
41  *       200:
42  *         description: Token é válido
43  *         content:
44  *           application/json:
45  *             schema:
46  *               type: object
47  *               properties:
48  *                 valid:
49  *                   type: boolean
50  *                   description: Token é válido
51  *                   example: true
52  *       401:
53  *         description: Token inválido ou expirado
54  *         content:
55  *           application/json:
56  *             schema:
57  *               type: object
58  *               properties:
59  *                 valid:
60  *                   type: boolean
61  *                   description: Token é inválido
62  *                   example: false
63  *       400:
64  *         description: Nenhum token fornecido
65  *         content:
66  *           application/json:
67  *             schema:
68  *               type: object
69  *               properties:
70  *                 valid:
71  *                   type: boolean
72  *                   description: Nenhum token fornecido
73  *                   example: false
74  */
75  router.post('/validarToken', tokenController.validaToken);
76
77  //exporta as rotas criadas
78  module.exports = router;
```

APIRouter

- Adicionar a rota no arquivo server.js

```
JS server.js > ...
1
2  const express = require('express');
3  const sequelize = require('./data_base/db');
4  const usuariosRotas = require('./rotas/usuarioRotas');
5  const uploadArquivoRotas = require('./rotas/uploadArquivoRotas');
6  const validarToken = require('./rotas/tokenRotas');
7
8  //Importar o modulo Swagger
9  const setupSwagger = require('./swagger');
10
11 //importar o modulo cors para receber requisicoes de diferente origem
12 const cors = require('cors');
13
14
15 const app = express();
16 const PORT = process.env.PORT;
17
18
19 app.use(require("cors")());
20
21 //restringir chamadas somente da origem conhecida
22 const corsOptions = {
23   origin: 'http://localhost:3000', // Permitir apenas essa origem
24   methods: 'GET,HEAD,PUT,PATCH,POST,DELETE', // Métodos permitidos
25   credentials: true, // Permitir envio de cookies
26   optionsSuccessStatus: 204 // Status para requisições preflight
27 };
28
29 app.use(cors(corsOptions));
30
31 // Configurar Swagger
32 setupSwagger(app);
33
34 app.use(express.json());
35 app.use('/api', usuariosRotas);
36 app.use('/api', uploadArquivoRotas);
37 app.use('/api', validarToken);
38
39
40 sequelize.sync().then(() => {
41   app.listen(PORT, () => {
42     console.log(`Servidor rodando na porta ${PORT}`);
43   });
44 });
```


APIRouter

- ▶ Testar o cadastro de usuário com os novos campos pelo Swagger

Usuario

Busca todos os usu rios

POST

/usuarios

Cr a um novo usu rio

Parameters

No parameters

Try it out

Request body

required

application/json

Example Value

Schema

```
{
  "nome": "string",
  "idade": 0,
  "cidade": "string",
  "uf": "st",
  "cep": "string",
  "complemento": "string",
  "bairro": "string",
  "numero": 0,
  "email": "string",
  "senha": "string"
}
```

- ## ► Testar o login pelo Swagger

Autenticação

POST /login Autentica um usuário

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{  "email": "string",  "senha": "string"}
```

Code

Details

200

Response body

```
{  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c3VhcmIvSWQ0IjIsIm1hdCI6MTcyMzEyMTk3OX0.CA6DFq0znn-H-P4pQLYAtm_n2wMyRMPXVpc_aI8NYZdg"
```

APIWEB

- No frontend alterar a página Cadastro.js incluindo os campos email e senha

```
16 function Cadastro() {
17
18   //cria novo estado para os campos da tela
19   const [campos, setCampos] = useState({
20     nome: '',
21     idade: 0,
22     cidade: '',
23     uf: '',
24     cep: '',
25     logradouro: '',
26     complemento: '',
27     bairro: '',
28     numero: 0,
29     email: '',
30     senha: '',
31     confirmarsenha: ''
32   });
33
```

```
98
99
100   if (!campos.senha) {
101     novosErros.senha = 'Senha é obrigatório';
102   }
103
104   if (!campos.confirmarsenha) {
105     novosErros.confirmarsenha = 'Confirmar Senha é obrigatório';
106   } else if (campos.confirmarsenha !== campos.senha) {
107     novosErros.senha = 'Senha e Confirmar Senha devem ser iguais!';
108   }

```

validarCampos()

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

```

handleFormSubmit

APIWEB

- Ainda na página Cadastro.js Incluir email e senha no formulário html

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

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

  <div className="field-menor">
    <label>Confirmar Senha:
      <input type="password" name="confirmarsenha" id="confirmarsenha" value={campos.confirmarsenha} onChange={handleInputChange} onBlur={validaConfirmacaoSenha}/>
      {erros.confirmarsenha && <p className="error">{erros.confirmarsenha}</p>}
    </label>
  </div>
</div>
```

- Criar a função validaConfirmacaoSenha, na página cadastro.js.

```
121 function validaConfirmacaoSenha(){
122   const novosErros = {};
123   if (!campos.confirmarsenha) {
124     novosErros.confirmarsenha = 'Confirmar Senha é obrigatório';
125   } else if (campos.confirmarsenha !== campos.senha) {
126     novosErros.confirmarsenha = 'Senha e Confirmar Senha devem ser iguais!';
127   }
128   setErros(novosErros);
129 }
130
```

APIWEB

- No frontend alterar a página EditarRegistro.js incluindo os campos email.

```
14 //cria novo estado para os campos da tela
15 const [campos, setCampos] = useState({
16   nome: '',
17   idade: 0,
18   cidade: '',
19   uf: '',
20   cep: '',
21   logradouro: '',
22   complemento: '',
23   bairro: '',
24   numero: 0,
25   email: ''
26 });
27
```

validarCampos()

```
if (!campos.email) {
  novosErros.email = 'E-mail é obrigatório';
}
```

APIWEB

- ▶ Ainda no EditarRegistro.js incluir email e senha no formulário html, porém campo senha tem que ficar desabilitado.

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

  <div className="field-menor">
    <label>Senha:
      <input type="password" name="senha" id="senha" value={campos.senha} disabled onChange={handleInputChange} />
      {erros.senha && <p className="error">{erros.senha}</p>}
    </label>
  </div>

  <div className="field-menor">
    <label>Confirmar Senha:
      <input type="password" name="confirmarsenha" id="confirmarsenha" value={campos.confirmarsenha} disabled onChange={handleInputChange} />
      {erros.confirmarsenha && <p className="error">{erros.confirmarsenha}</p>}
    </label>
  </div>
</div>
```

APIWEB

- Criar a página de Login.js dentro da pasta paginas.

```
1
2 import React, { useState, useContext } from 'react';
3 import axios from 'axios';
4 import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
5 import { faUser, faLock } from '@fortawesome/free-solid-svg-icons';
6 import { useNavigate } from 'react-router-dom';
7
8 import '../CSS/login.css'; // Importando o arquivo CSS
9
10 //Utilizado para armazenar o token no localStorage ou sessionStorage após o login
11 import { AuthContext } from '../autenticacao/autenticacao';
12
13 const Login = () => {
14   const [email, setEmail] = useState('');
15   const [senha, setSenha] = useState('');
16   const [token, setToken] = useState('');
17   const { setAuthToken } = useContext(AuthContext);
18   const navigate = useNavigate();
19
20   const login = async () => {
21     try {
22       const response = await axios.post('http://localhost:3001/api/login', { email, senha });
23       setAuthToken(response.data.token);
24       //localStorage.setItem('token', response.data.token); // Armazena o token
25       setToken(response.data.token);
26       navigate('/'); // Redireciona para a página interna
27     } catch (error) {
28       alert('Erro no login: ' + error.response.data);
29     }
30   };
31 }
```

APIWEB

- ▶ Continuação do código da página de Login.js.

```
34
35
36   return (
37     <div className="login-container">
38       <div className="login-box">
39
40         <h1>Login</h1>
41         <div className="input-container">
42           <FontAwesomeIcon icon={faUser} className="input-icon" />
43           <input
44             type="email"
45             placeholder="E-mail do usuario"
46             value={email}
47             onChange={(e) => setEmail(e.target.value)}
48           />
49         </div>
50         <br></br>
51         <div className="input-container">
52           <FontAwesomeIcon icon={faLock} className="input-icon" />
53           <input
54             type="password"
55             placeholder="Senha"
56             value={senha}
57             onChange={(e) => setSenha(e.target.value)}
58           />
59         </div>
60         <br></br>
61         <a href="#" className="link">Esqueceu a senha?</a>
62         <div className="button-container">
63           <button onClick={login} className="button">Login</button>
64         </div>
65         <br></br>
66         <Link to="/cadastro">Acessar cadastro</Link>
67       </div>
68     </div>
69   );
70 };
71 export default Login;
```

APIWEB

- ▶ No frontend, criar a pasta axios na raiz do projeto e configurar uma instância do Axios
 - ▶ Enviar token nas requisições para o backend
 - ▶ Essa instancia vai enviar no Headers o token para o backend

```
src > axios > JS configuracaoAxios.js > ...
1
2
3 import axios from 'axios';
4
5 const token = localStorage.getItem('token');
6
7 const axiosInstance = axios.create({
8   baseURL: 'http://localhost:3001/api',
9   headers: {
10     'Authorization': `Bearer ${token}`
11   }
12 });
13
14 export default axiosInstance;
```

- ▶ Alterar a requisições para backend usando o axiosInstance:
- ▶ Cadastro.js, EditarRegistro.js, ListaRegistros.js, Upload.js

```
axiosInstance.post('/usuarios', campos)
```


APIWEB

- ▶ Criar uma pasta autenticação na raiz do projeto
- ▶ Criar o arquivo rotasPrivadas.js, escrever o código abaixo

```
1
2  import React, { useContext, useEffect, useState } from 'react';
3  import { Navigate, Outlet } from 'react-router-dom';
4  import { AuthContext } from '../autenticacao';
5  import axios from 'axios';
6
7  const PrivateRoute = () => {
8
9      const { authToken } = useContext(AuthContext);
10     const [isValid, setIsValid] = useState(null);
11
12     const verifyToken = async () => {
13         try {
14             const response = await axios.post('http://localhost:3001/api/validarToken', { token: authToken });
15             setIsValid(response.data.valid);
16         } catch {
17             localStorage.removeItem('token');
18             setIsValid(false);
19         }
20     };
21
22     verifyToken();
23
24     useEffect(() => {
25         if (authToken) {
26             console.log('Token valido ');
27         } else {
28             console.log('PrivateRoute Remove o Token authToken', authToken);
29             localStorage.removeItem('token');
30             setIsValid(false);
31         }
32     }, [authToken]);
33
34     if (isValid === null) {
35         return <div>Loading...</div>;
36     }
37
38     return isValid ? <Outlet /> : <Navigate to="/login" />;
39 };
40
41 export default PrivateRoute;
```

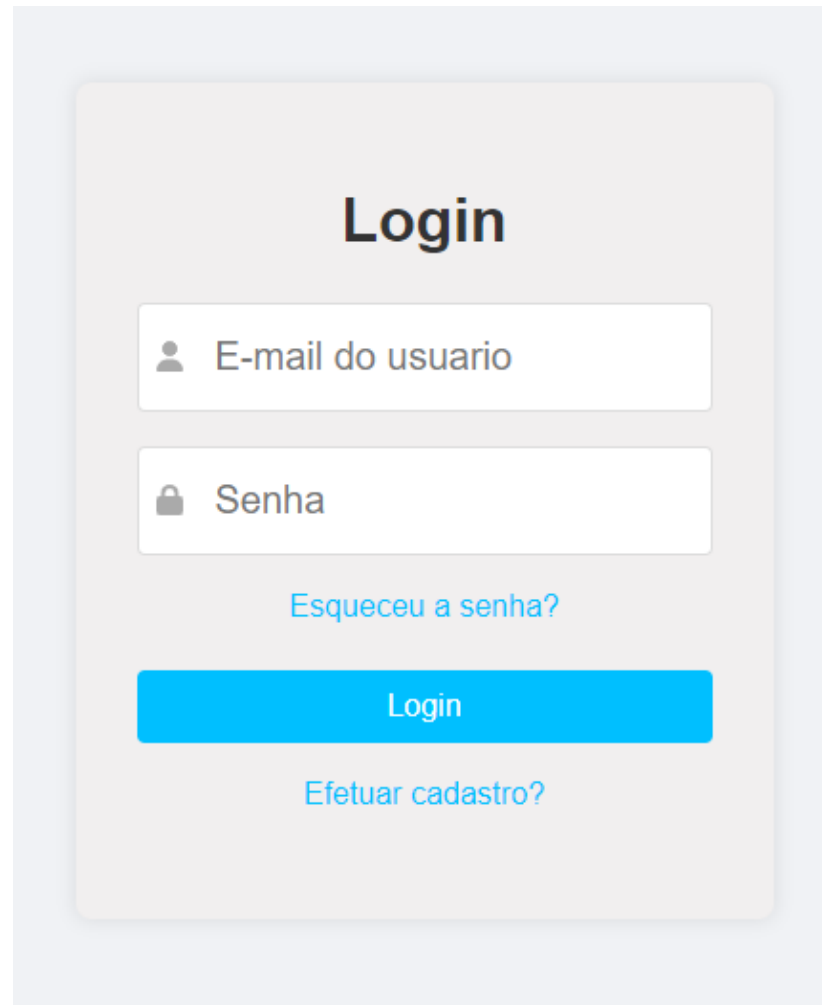
APIWEB

- ▶ Dentro da pasta autenticação
- ▶ Criar o arquivo autenticaao.js, escrever o código abaixo


```
1 import React, { createContext, useState, useEffect } from 'react';
2 import axios from 'axios';
3 export const AuthContext = createContext();
4
5 const verificarTokenServidor = async (token) => {
6   try {
7     const response = await axios.post('http://localhost:3001/api/validarToken', { token });
8     return response.data.valid;
9   } catch (error) {
10     return false;
11   }
12 };
13
14 export const AuthProvider = ({ children }) => {
15   const [authToken, setAuthToken] = useState('');
16
17   useEffect(() => {
18     const checkToken = async () => {
19       const token = localStorage.getItem('token');
20       console.log('Token', token);
21       if (token) {
22         const isValid = await verificarTokenServidor(token);
23         if (isValid) {
24           console.log('Token valido');
25           setAuthToken(token);
26         } else {
27           console.log('Remove Token');
28           localStorage.removeItem('token');
29         }
30       }
31     };
32     checkToken();
33   }, []);
34
35   useEffect(() => {
36     if (authToken) {
37       console.log('Seta o Token ', authToken);
38       localStorage.setItem('@Auth:token', authToken);
39     } else {
40       console.log('Remove o Token authToken', authToken);
41       localStorage.removeItem('@Auth:token', authToken);
42     }
43   }, [authToken]);
44
45   return (
46     <AuthContext.Provider value={{ authToken, setAuthToken }}>
47       {children}
48     </AuthContext.Provider>
49   );
50 };
51
```


APIWEB

- ▶ Criar a pasta CSS e criar o arquivo login.css, copiar o código disponibilizado no Github

A mockup of a login form centered on a light gray background. The form itself is a light beige rectangle with rounded corners. At the top, the word "Login" is written in a bold, black, sans-serif font. Below the title are two input fields. The first field has a small gray user icon to its left and the placeholder text "E-mail do usuario". The second field has a small gray lock icon to its left and the placeholder text "Senha". Below these fields is a link "Esqueceu a senha?" in a blue, sans-serif font. Underneath the link is a solid blue rectangular button with the word "Login" in white, bold, sans-serif font. At the bottom of the form is another link "Efetuar cadastro?" in a blue, sans-serif font.

Login

 E-mail do usuario

 Senha

[Esqueceu a senha?](#)

Login

[Efetuar cadastro?](#)

APIWEB

- ▶ Ainda no frontend editar o arquivo Rotas.js, alterando o formato das rotas.

```
1
2 import React from 'react';
3
4 //importa 3 objetos da lib
5 import { Route, Routes, BrowserRouter } from 'react-router-dom';
6
7 //Importa a página Home
8 import Home from '../paginas/Home';
9
10 //Importa a página Cadastro
11 import Cadastro from '../paginas/Cadastro';
12 import ListaRegistros from '../paginas/ListaRegistros';
13 import EditarRegistro from '../paginas/EditarRegistro';
14 import Upload from '../componentes/Upload';
15 import Login from '../paginas/Login';
16 import { AuthProvider } from '../autenticacao/autenticacao';
17 import PrivateRoute from '../autenticacao/rotasPrivadas';
18
19 function Rotas() {
20   return (
21     <AuthProvider>
22       <BrowserRouter>
23         <Routes>
24           <Route path="/cadastro" element={<Cadastro />} />
25           <Route path="/login" element={<Login />} />
26
27           <Route path="/" element={<PrivateRoute />}>
28             <Route path="/" element={<Home />} />
29           </Route>
30
31           <Route path="/lista" element={<PrivateRoute />}>
32             <Route path="/lista" element={<ListaRegistros />} />
33           </Route>
34           <Route path="/editar/:id" element={<PrivateRoute />}>
35             <Route path="/editar/:id" element={<EditarRegistro />} />
36           </Route>
37           <Route path="/upload" element={<PrivateRoute />}>
38             <Route path="/upload" element={<Upload />} />
39           </Route>
40         </Routes>
41       </BrowserRouter>
42     </AuthProvider>
43   )
44
45   export default Rotas;
```