

# Validações para inserção, atualização e exclusão de dados

Ewerton José da Silva

# Exemplo update endereço principal em um insert

Quando um usuário insere um novo endereço e o define como principal o endereço que era o principal anteriormente deve ter o campo correspondente atualizado para “false”.

```
37  async cadastrarClienteEnderecos(request, response) {
38      try {
39
40          const { idUsuario, logradouro, num, bairro, complemento, idCidade, principal } = request.body;
41          const end_excluido = false;
42          let end_principal = principal;
43
44          // 1. Verificar se já existem endereços para este usuário
45          const sqlChecarEndereco = `SELECT COUNT(*) AS total_enderecos FROM cliente_enderecos WHERE usu_id = ? AND end_excluido = false`;
46          const [resultCheck] = await db.query(sqlChecarEndereco, [idUsuario]);
47          const totalEnderecos = resultCheck[0].total_enderecos;
48
49          // 2. Se não houver endereços, defina o novo como principal
50          if (totalEnderecos === 0) {
51              end_principal = true;
52          } else {
53              // Se já houver endereços e o que está sendo cadastrado for definido como principal
54              if (end_principal === true) {
55                  const sqlUpdateEnd = `UPDATE cliente_enderecos SET end_principal = 0 WHERE usu_id = ?`;
56                  await db.query(sqlUpdateEnd, [idUsuario]);
57              }
58          }
59      }
60  }
```

```
60     const sql = `
61         INSERT INTO cliente_enderecos
62         (usu_id, end_logradouro, end_num, end_bairro, end_complemento, cid_id, end_principal, end_excluido)
63         VALUES
64         (?, ?, ?, ?, ?, ?, ?, ?);
65     `;
66
67     const values = [idUserario, logradouro, num, bairro, complemento, idCidade, principal, end_excluido];
68
69     const [result] = await db.query(sql, values);
70
71     const end_id = result.insertId;
72
73     return response.status(200).json({
74         sucesso: true,
75         mensagem: `Cadastro de endereço do cliente ${idUserario} realizado com sucesso.`,
76         dados: { end_id }
77     });
78
79     } catch (error) {
80         return response.status(500).json({
81             sucesso: false,
82             mensagem: 'Erro na requisição.',
83             dados: error.message
84         });
85     }
86 },
```

```
{
  "id": 8,
  "idUsuarios": 4,
  "logradouro": "Rua Brasil",
  "numero": "1645",
  "bairro": "Vila Abarca",
  "complemento": null,
  "idCidade": 3886,
  "principal": 1, ←
  "excluido": 0
}
```

POST ▼ http://localhost:3333/cliente-enderecos

Params	Body ●	Auth	Headers 4	Sc
JSON ▼				
<pre>1 { 2   "idUsuario": 4, 3   "logradouro": "Avenida Tamoios", 4   "num": "500", 5   "bairro": "Centro", 6   "complemento": null, 7   "idCidade": 3886, 8   "principal": true ← 9 }</pre>				

```
1 {
2   "sucesso": true,
3   "mensagem": "Cadastro de endereço do cliente 4 realizado com sucesso.",
4   "dados": {
5     "end_id": 9
6   }
7 }
```

```
{
  "id": 8,
  "idUsuarios": 4,
  "logradouro": "Rua Brasil",
  "numero": "1645",
  "bairro": "Vila Abarca",
  "complemento": null,
  "idCidade": 3886,
  "principal": 0, ←
  "excluido": 0
},
```

```
{
  "id": 9,
  "idUsuarios": 4,
  "logradouro": "Avenida Tamoios",
  "numero": "500",
  "bairro": "Centro",
  "complemento": null,
  "idCidade": 3886,
  "principal": 1, ←
  "excluido": 0
}
```

# Validações básicas – Ex: cadastro de produto

```
76   async cadastrarProdutos(request, response) {
77     try {
78
79       const { nome, valor, unidade, tipo, disponivel, descricao, img, imagemDestaque } = request.body;
80
81       if (!nome || !valor || !unidade || !tipo || typeof disponivel === 'undefined') {
82         return response.status(400).json({
83           sucesso: false,
84           mensagem: 'Campos obrigatórios estão ausentes ou inválidos.', ←
85         });
86       } // bibliotecas como Joi (sem typescript) ou Zod (typescript) podem auxiliar nas validações.
87
88       // Verificar se o tipo existe
89       const sqlIngrediente = `SELECT ptp_id FROM produto_tipos WHERE ptp_id = ?`;
90       const [tipoResult] = await db.query(sqlIngrediente, [tipo]);
91
92       if (tipoResult.length === 0) {
93         return response.status(404).json({
94           sucesso: false,
95           mensagem: 'Tipo de produto não encontrado.', ←
96           dados: null
97         });
98       }
99
100      const destaque = imagemDestaque ? 1 : 0;
101      const img_destaque = imagemDestaque ? imagemDestaque : null;
102
103      // instrução sql para inserção
104      const sql = `
105        INSERT INTO produtos
106          (prd_nome, prd_valor, prd_unidade, ptp_id, prd_disponivel, prd_img, prd_destaque, prd_img_destaque, prd_descricao)
107        VALUES
108          (?, ?, ?, ?, ?, ?, ?, ?, ?);
109      `;
```

```
111 // definição de array com os parâmetros que receberam os valores do front-end
112 const values = [nome, parseFloat(valor), unidade, parseInt(tipo), parseInt(disponivel), img, destaque, img_destaque, descricao];
113
114 // executa a instrução de inserção no banco de dados
115 const [result] = await db.query(sql, values);
116
117 // Exibe o id do registro inserido
118 const prd_id = result.insertId;
119 // Mensagem de retorno no formato JSON
120 const dados = {
121   id: prd_id,
122   nome,
123   valor: parseFloat(valor).toFixed(2),
124   unidade,
125   tipo,
126   disponivel,
127   img
128 };
129
130 return response.status(200).json({
131   sucesso: true,
132   mensagem: 'Produto cadastrado com sucesso!',
133   dados
134 });
135 } catch (error) {
136   return response.status(500).json({
137     sucesso: false,
138     mensagem: 'Erro na requisição.',
139     dados: error.message
140   });
141 }
142 },
```

# Validação em cadastro da relação entre produtos e ingredientes

```
42  async cadastrarProdutoIngredientes(request, response) {
43    try {
44      const { produto, ingrediente, adicional } = request.body;
45
46      // Validação manual dos dados
47      if (!produto || !ingrediente || adicional === undefined) {
48        return response.status(400).json({
49          sucesso: false,
50          mensagem: 'Campos obrigatórios: produto, ingrediente e adicional.',
51          dados: null
52        });
53      }
54
55      if (isNaN(produto) || isNaN(ingrediente)) {
56        return response.status(400).json({
57          sucesso: false,
58          mensagem: 'Os campos produto e ingrediente devem ser números.',
59          dados: null
60        });
61      }
62
63      if (adicional !== 0 && adicional !== 1) {
64        return response.status(400).json({
65          sucesso: false,
66          mensagem: 'O campo adicional deve ser 0 (não é adicional) ou 1 (é adicional).',
67          dados: null
68        });
69      }
70
71      // Verificar se o produto existe
72      const sqlProduto = `SELECT prd_id FROM produtos WHERE prd_id = ?`;
73      const [produtoResult] = await db.query(sqlProduto, [produto]);
```

```
75     if (produtoResult.length === 0) {
76         return response.status(404).json({
77             sucesso: false,
78             mensagem: 'Produto não encontrado.', ←
79             dados: null
80         });
81     }
82
83     // Verificar se o ingrediente existe
84     const sqlIngrediente = `SELECT ing_id FROM ingredientes WHERE ing_id = ?`;
85     const [ingredienteResult] = await db.query(sqlIngrediente, [ingrediente]);
86
87     if (ingredienteResult.length === 0) {
88         return response.status(404).json({
89             sucesso: false,
90             mensagem: 'Ingrediente não encontrado.', ←
91             dados: null
92         });
93     }
94
95     // Verificar se o registro já existe
96     const sqlCheck = `
97         SELECT * FROM produto_ingredientes
98         WHERE prd_id = ? AND ing_id = ?
99     `;
100     const valuesCheck = [produto, ingrediente];
101     const [check] = await db.query(sqlCheck, valuesCheck);
102
103     if (check.length > 0) {
104         return response.status(409).json({
105             sucesso: false,
106             mensagem: 'Este ingrediente já está relacionado a este produto.', ←
107             dados: null
108         });
109     }
```



```
111 // Inserir registro
112 const sql = `
113     INSERT INTO produto_ingredientes
114     |     (prd_id, ing_id, prd_ing_adicional)
115     |     VALUES (?, ?, ?);
116 `;
117 const values = [produto, ingrediente, adicional];
118
119 await db.query(sql, values);
120
121 return response.status(201).json({
122     sucesso: true,
123     mensagem: 'Ingrediente adicionado ao produto com sucesso.',
124     dados: { produto, ingrediente, adicional }
125 });
126
127 } catch (error) {
128     return response.status(500).json({
129         sucesso: false,
130         mensagem: 'Erro ao cadastrar ingrediente no produto.',
131         dados: error.message
132     });
133 }
134 },
```

# Insert cliente

Todo cliente inserido deve ser um usuário, então no cadastro do cliente é recomendado fazer também um cadastro de usuário junto, em nosso exemplo também estamos inserindo o endereço.

```
53  async cadastrarClientes(request, response) {  
54      try {  
55          const {  
56              usu_nome,  
57              usu_email,  
58              usu_senha,  
59              usu_dt_nasc,  
60              usu_cpf,  
61              end_logradouro,  
62              end_num,  
63              end_bairro,  
64              end_complemento,  
65              cid_id,  
66              cli_cel  
67          } = request.body;  
68      }
```

```
69 // Verifica campos obrigatórios
70 if (
71     !usu_nome || !usu_email || !usu_senha || !usu_dt_nasc ||
72     !usu_cpf || !end_logradouro || !end_num || !end_bairro ||
73     !cid_id || !cli_cel
74 ) {
75     return response.status(400).json({
76         sucesso: false,
77         mensagem: 'Todos os campos obrigatórios devem ser preenchidos.',
78         dados: null
79     });
80 }
81
82 // Validação de e-mail
83 const emailRegex = /^[^\\s@]+@[^\\s@]+\\.\\.[^\\s@]+$/;
84 if (!emailRegex.test(usu_email)) {
85     return response.status(400).json({
86         sucesso: false,
87         mensagem: 'E-mail inválido.',
88         dados: null
89     });
90 }
91
92 // Validação de CPF
93 const cpf = cpfToInt(usu_cpf);
94 if (cpf.length !== 11 || isNaN(cpf)) {
95     return response.status(400).json({
96         sucesso: false,
97         mensagem: 'CPF inválido.',
98         dados: null
99     });
100 }
```

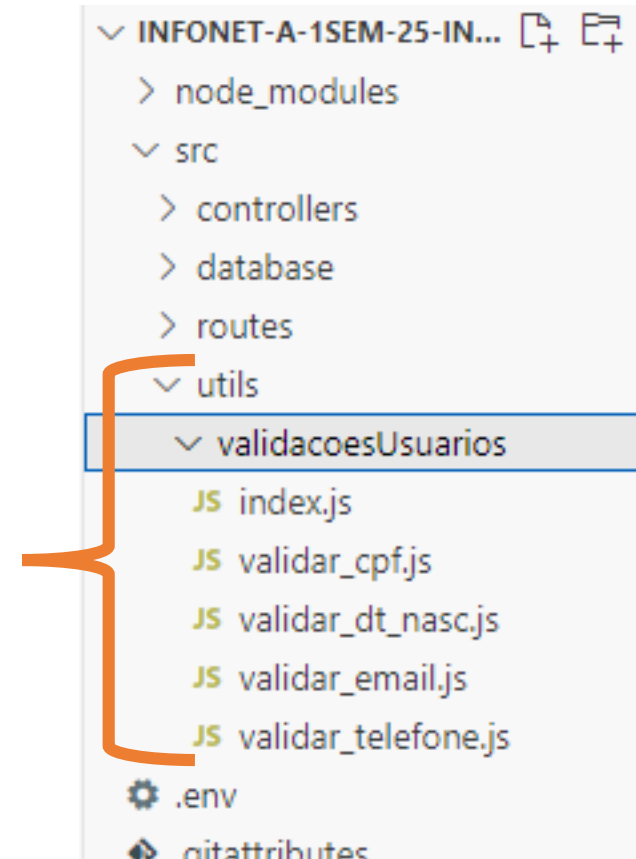
```
102 // Validação de data de nascimento (formato básico yyyy-mm-dd)
103 const dataRegex = /^\\d{4}-\\d{2}-\\d{2}$/;
104 if (!dataRegex.test(usu_dt_nasc)) {
105     return response.status(400).json({
106         sucesso: false,
107         mensagem: 'Data de nascimento inválida. Use o formato YYYY-MM-DD.',
108         dados: null
109     });
110 }
111
112 // Remove máscara do telefone e valida
113 const telefoneSemMascara = cli_cel.replace(/\\D/g, '');
114 if (telefoneSemMascara.length < 10 || telefoneSemMascara.length > 11) {
115     return response.status(400).json({
116         sucesso: false,
117         mensagem: 'Telefone inválido.',
118         dados: null
119     });
120 }
121
122 // Verifica se o e-mail já existe
123 const [emailExiste] = await db.query(`SELECT usu_id FROM usuarios WHERE usu_email = ?`, [usu_email]);
124 if (emailExiste.length > 0) {
125     return response.status(409).json({
126         sucesso: false,
127         mensagem: 'E-mail já cadastrado.',
128         dados: null
129     });
130 }
131
```

```
132 // Verifica se o CPF já existe
133 const [cpfExiste] = await db.query(`SELECT usu_id FROM usuarios WHERE usu_cpf = ?`, [cpf]);
134 if (cpfExiste.length > 0) {
135     return response.status(409).json({
136         sucesso: false,
137         mensagem: 'CPF já cadastrado.', ←
138         dados: null
139     });
140 }
141
142 const usu_tipo = 2;
143 const usu_ativo = 1;
144 const cli_pts = 0;
145 const end_principal = true;
146 const end_excluido = false;
147
148 // Inserir usuário
149 const sqlUsu = `
150     INSERT INTO usuarios
151     |     (usu_nome, usu_email, usu_senha, usu_dt_nasc, usu_cpf, usu_tipo, usu_ativo)
152     |     VALUES (?, ?, ?, ?, ?, ?, ?)
153 `;
154 const [usuarios] = await db.query(sqlUsu, [usu_nome, usu_email, usu_senha, usu_dt_nasc, cpf, usu_tipo, usu_ativo]);
155 const usu_id = usuarios.insertId;
156
157 // Inserir cliente
158 const sqlCli = `
159     INSERT INTO clientes (usu_id, cli_cel, cli_pts)
160     |     VALUES (?, ?, ?)
161 `;
162 await db.query(sqlCli, [usu_id, telefoneSemMascara, cli_pts]);
163
```

```
164 // Inserir endereço
165 const sqlEnd = `
166     INSERT INTO endereco_clientes
167     |     (usu_id, end_logradouro, end_num, end_bairro, end_complemento, cid_id, end_principal, end_excluido)
168     |     VALUES (?, ?, ?, ?, ?, ?, ?, ?)
169 `;
170 await db.query(sqlEnd, [usu_id, end_logradouro, end_num, end_bairro, end_complemento, cid_id, end_principal, end_excluido]);
171
172 return response.status(201).json({
173     sucesso: true,
174     mensagem: `Cadastro do cliente ${usu_id} realizado com sucesso!`,
175     dados: { usu_id }
176 });
177
178 } catch (error) {
179     return response.status(500).json({
180         sucesso: false,
181         mensagem: 'Erro interno ao cadastrar cliente.',
182         dados: error.message
183     });
184 }
185 },
```

# Validações compartilhadas

- Utilitários de validação
- Separação de arquivos que podem ser utilizados em mais de um método.
- Crie uma pasta em “src” com o nome utils e dentro dela um “index.js” e um arquivo para cada validação.



# Arquivos de validação para CPF e data de nascimento

src > utils > validacoesUsuarios > JS validar\_cpf.js > ...

```
1 function validarCPF(cpf) {
2   if (typeof cpf !== 'string') return false;
3   cpf = cpf.replace(/[^\d]+/g, '');
4   if (cpf.length !== 11 || /^(\\d)\\1+$/.test(cpf)) return false;
5
6   let soma = 0;
7   for (let i = 0; i < 9; i++) soma += Number(cpf[i]) * (10 - i);
8   let dig1 = (soma * 10) % 11;
9   if (dig1 === 10 || dig1 === 11) dig1 = 0;
10  if (dig1 !== Number(cpf[9])) return false;
11
12  soma = 0;
13  for (let i = 0; i < 10; i++) soma += Number(cpf[i]) * (11 - i);
14  let dig2 = (soma * 10) % 11;
15  if (dig2 === 10 || dig2 === 11) dig2 = 0;
16  if (dig2 !== Number(cpf[10])) return false;
17
18  return true;
19 }
20
21 module.exports = validarCPF;
```

src > utils > validacoesUsuarios > JS validar\_dt\_nasc.js > ...

```
1 function validarDataNascimento(data, idadeMinima = 12) {
2   if (!data) return false;
3   const dataNasc = new Date(data);
4   const hoje = new Date();
5   const idade = hoje.getFullYear() - dataNasc.getFullYear();
6   const mes = hoje.getMonth() - dataNasc.getMonth();
7
8   if (
9     idade > idadeMinima ||
10    (idade === idadeMinima && mes >= 0 && hoje.getDate() >= dataNasc.getDate())
11  ) {
12    return true;
13  }
14
15  return false;
16 }
17
18 module.exports = validarDataNascimento;
```



# Arquivos de validação para e-mail, telefone e o index que centraliza todas as validações

```
src > utils > validacoesUsuarios > JS validar_email.js > ...
```

```
1 function validarEmail(email) {  
2   const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;  
3   return regex.test(email);  
4 }  
5  
6 module.exports = validarEmail;
```

```
src > utils > validacoesUsuarios > JS validar_telefone.js > ...
```

```
1 function validarTelefone(telefone) {  
2   // Aceita 11 dígitos, começando com 9 após o DDD  
3   const telefoneSemMascara = telefone.replace(/\D/g, '');  
4   return /^(?d{2})9\d{8}$/.test(telefoneSemMascara);  
5 }  
6  
7 module.exports = validarTelefone;
```

```
src > utils > validacoesUsuarios > JS index.js > ...
```

```
1 module.exports = {  
2   validarCPF: require('./validar_cpf'),  
3   validarEmail: require('./validar_email'),  
4   validarTelefone: require('./validar_telefone'),  
5   validarDataNascimento: require('./validar_dt_nasc'),  
6 };
```

# Aplicando validações no controller “clientes”

Além das validações criadas, temos uma função que retira a máscara do cpf.

```
src > controllers > JS clientes.js > ...
1  const db = require('../database/connection');
2
3  const {
4      validarCPF,
5      validarEmail,
6      validarTelefone,
7      validarDataNascimento
8  } = require('../utils/validacoesUsuarios');
9
10 function cpfToInt(cpf) {
11     const cpfSemMascara = cpf.replace(/\D/g, '');
12     const cpfInteiro = parseInt(cpfSemMascara);
13     return cpfInteiro;
14 };
15
16 module.exports = {
17     async listarClientes(request, response) {
```

```
59  async cadastrarClientes(request, response) {
60      try {
61          const {
62              nome,
63              email,
64              senha,
65              dataNasc,
66              cpf,
67              logradouro,
68              num,
69              bairro,
70              complemento,
71              idCidade,
72              cel
73          } = request.body;
74
75          // Verifica campos obrigatórios
76          if (
77              !nome || !email || !senha || !dataNasc || !cpf ||
78              !logradouro || !num || !bairro || !idCidade || !cel
79          ) {
80              return response.status(400).json({
81                  sucesso: false,
82                  mensagem: 'Todos os campos obrigatórios devem ser preenchidos.',
83                  dados: null
84              });
85          }
86
87          // Validação de e-mail
88          if (!validarEmail(email)) {
89              return response.status(400).json({
90                  sucesso: false,
91                  mensagem: 'E-mail inválido.',
92                  dados: null
93              });
94          }
```

```
96 // Validação de CPF
97 if (!validarCPF(cpf)) {
98     return response.status(400).json({
99         sucesso: false,
100         mensagem: 'CPF inválido.',
101         dados: null
102     });
103 }
104
105 const usu_cpf = cpfToInt(cpf);
106
107 // Validação de data de nascimento (formato básico yyyy-mm-dd)
108 const dataRegex = /^\\d{4}-\\d{2}-\\d{2}$/;
109 if (!dataRegex.test(dataNasc)) {
110     return response.status(400).json({
111         sucesso: false,
112         mensagem: 'Data de nascimento inválida. Use o formato YYYY-MM-DD.',
113         dados: null
114     });
115 }
116
117 if (!validarDataNascimento(dataNasc)) {
118     return response.status(400).json({
119         sucesso: false,
120         mensagem: 'A data de nascimento não pode ser hoje!',
121         dados: null
122     });
123 }
124
125 // Remove máscara do telefone e valida
126 if (!validarTelefone(cel)) {
127     return response.status(400).json({
128         sucesso: false,
129         mensagem: 'Telefone inválido.',
130         dados: null
131     });
132 }
133
134 const cli_cel = cel.replace(/\\D/g, '');

```

```

136 // Verifica se o e-mail já existe
137 const [emailExiste] = await db.query(`SELECT usu_id FROM usuarios WHERE usu_email = ?`, [email]);
138 if (emailExiste.length > 0) {
139     return response.status(409).json({
140         sucesso: false,
141         mensagem: 'E-mail já cadastrado.',
142         dados: null
143     });
144 }
145
146 // Verifica se o CPF já existe
147 const [cpfExiste] = await db.query(`SELECT usu_id FROM usuarios WHERE usu_cpf = ?`, [usu_cpf]);
148 if (cpfExiste.length > 0) {
149     return response.status(409).json({
150         sucesso: false,
151         mensagem: 'CPF já cadastrado.',
152         dados: null
153     });
154 }
155
156 const usu_tipo = 2;
157 const usu_ativo = 1;
158 const cli_pts = 0;
159 const end_principal = true;
160 const end_excluido = false;
161
162 // Inserir usuário
163 const sqlUsu = `
164     INSERT INTO usuarios
165     |   (usu_nome, usu_email, usu_senha, usu_dt_nasc, usu_cpf, usu_tipo, usu_ativo)
166     |   VALUES (?, ?, ?, ?, ?, ?, ?)
167 `;
168 const [usuarios] = await db.query(sqlUsu, [nome, email, senha, dataNasc, usu_cpf, usu_tipo, usu_ativo]);
169 const usu_id = usuarios.insertId;

```

```

171 // Inserir cliente
172 const sqlCli = `
173     INSERT INTO clientes (usu_id, cli_cel, cli_pts)
174     VALUES (?, ?, ?)
175 `;
176 await db.query(sqlCli, [usu_id, cli_cel, cli_pts]);
177
178 // Inserir endereço
179 const sqlEnd = `
180     INSERT INTO cliente_enderecos
181     (usu_id, end_logradouro, end_num, end_bairro, end_complemento, cid_id, end_principal, end_excluido)
182     VALUES (?, ?, ?, ?, ?, ?, ?, ?)
183 `;
184 await db.query(sqlEnd, [usu_id, logradouro, num, bairro, complemento, idCidade, end_principal, end_excluido]);
185
186 return response.status(201).json({
187     sucesso: true,
188     mensagem: `Cadastro do cliente ${usu_id} realizado com sucesso!`,
189     dados: { usu_id }
190 });
191
192 } catch (error) {
193     return response.status(500).json({
194         sucesso: false,
195         mensagem: 'Erro interno ao cadastrar cliente.',
196         dados: error.message
197     });
198 }
199 },

```

# Teste de validações

POST http://localhost:3333/clientes Send 201 Created 70 ms 97 B

Params Body Auth Headers 4 Scripts Docs

JSON

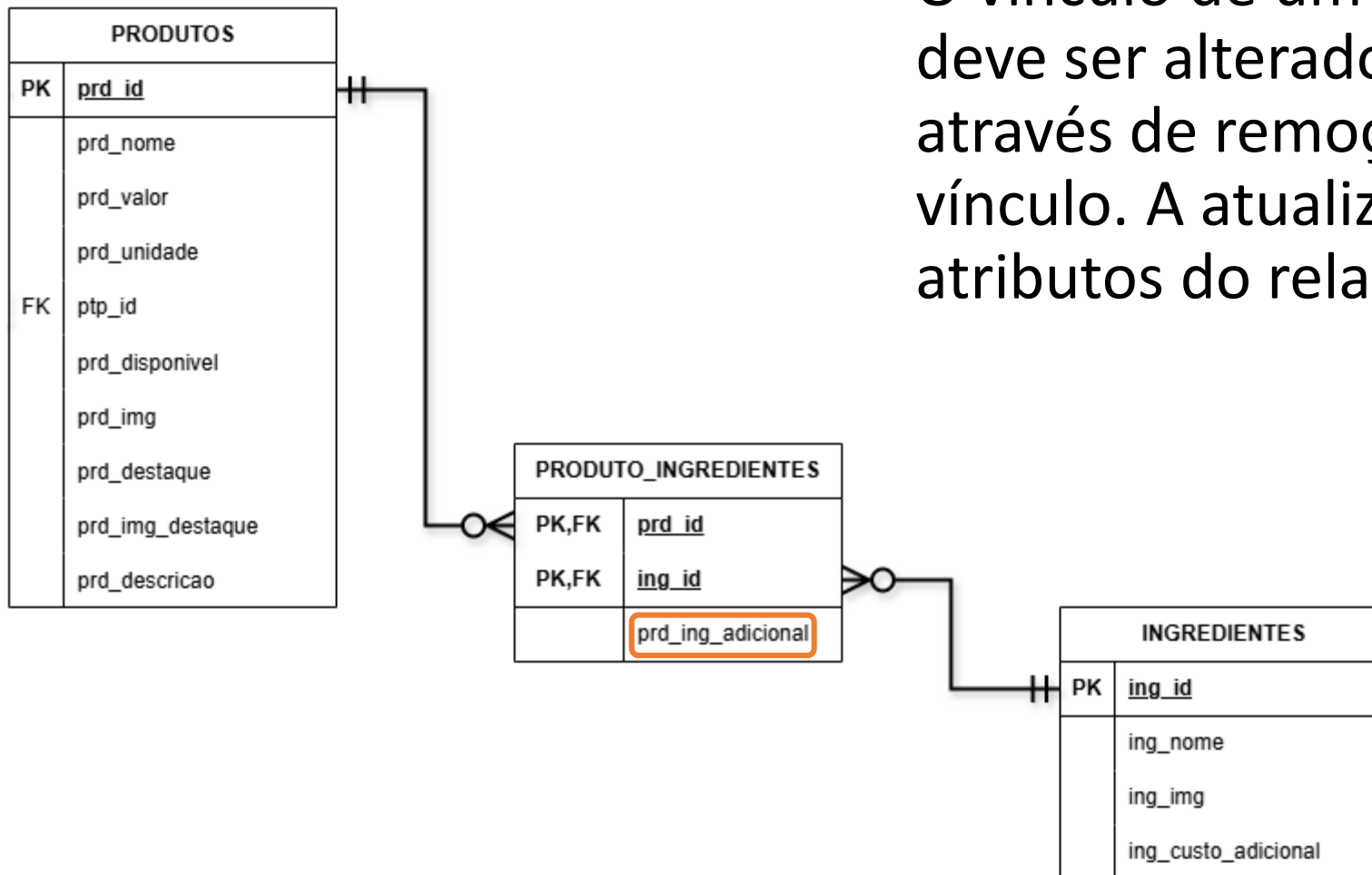
```
1 {
2   "nome": "Cecília Carmelo",
3   "email": "ceci@email.com",
4   "cpf": "075.799.330-39",
5   "dataNasc": "1977-01-03",
6   "senha": "123",
7   "logradouro": "Rua Alagoas",
8   "num": "18",
9   "bairro": "Centro",
10  "complemento": null,
11  "idCidade": 3674,
12  "cel": "(18) 99582-1519"
13 }
```

Preview Headers 8 Cookies Tests 0 / 0 → Mock Console

Preview

```
1 {
2   "sucesso": true,
3   "mensagem": "Cadastro do cliente 10 realizado com sucesso!",
4   "dados": {
5     "usu_id": 10
6   }
7 }
```

# Atualização em relacionamentos n:n



O vínculo de um relacionamento N:N não deve ser alterado via atualização, e sim através de remoção e criação de um novo vínculo. A atualização deve se limitar aos atributos do relacionamento






```
135 async editarProdutoIngredientes(request, response) {
136   try {
137     const { idProd, idIng } = request.params;
138     const { adicional } = request.body;
139
140     // Validação dos parâmetros
141     if (!idProd || !idIng) {
142       return response.status(400).json({
143         sucesso: false,
144         mensagem: 'Produto e Ingrediente são obrigatórios nos parâmetros.',
145         dados: null
146       });
147     }
148
149     // Verificar se o vínculo existe
150     const [vinculo] = await db.query(
151       `
152       SELECT prd_id AS idProduto, ing_id AS idIngrediente, prd_ing_adicional = 1 AS adicional
153       FROM produto_ingredientes
154       WHERE prd_id = ? AND ing_id = ?
155       `,
156       [idProd, idIng]
157     );
158
159     if (vinculo.length === 0) {
160       return response.status(404).json({
161         sucesso: false,
162         mensagem: 'Vínculo entre produto e ingrediente não encontrado.',
163         dados: null
164       });
165     }
166   }
```


```
166
167 // Validar e preparar campos para atualizar
168 if (adicional !== undefined) {
169     if (![0, 1].includes(adicional)) {
170         return response.status(400).json({
171             sucesso: false,
172             mensagem: 'O campo adicional deve ser 0 (não é adicional) ou 1 (é adicional).',
173             dados: null
174         });
175     }
176 }
177
178 const sql = `UPDATE produto_ingredientes SET prd_ing_adicional = ? WHERE prd_id = ? AND ing_id = ?`;
179 const values = [adicional, idProd, idIng];
180
181 await db.query(sql, values);
182
183 // Buscar dados atualizados
184 const [vinculoAtualizado] = await db.query(
185     `
186     SELECT prd_id AS idProduto, ing_id AS idIngrediente, prd_ing_adicional = 1 AS adicional
187     FROM produto_ingredientes
188     WHERE prd_id = ? AND ing_id = ?
189     `,
190     [idProd, idIng]
191 );
```

```
193         return response.status(200).json({
194             sucesso: true,
195             mensagem: 'Vínculo atualizado com sucesso.',
196             dados: {
197                 antigo: vinculo[0], ←
198                 atualizado: vinculoAtualizado[0] ←
199             }
200         });
201
202     } catch (error) {
203         return response.status(500).json({
204             sucesso: false,
205             mensagem: 'Erro ao atualizar vínculo.',
206             dados: error.message
207         });
208     }
209 },
```


# Teste update



PATCH  http://localhost:3333/produto/1/ingrediente/6 Send  200 OK 94 ms 190 B

Params Body  Auth Headers 4 Scripts Docs

JSON 

```
1 {
2   "adicional": 0
3 }
```

Preview 

```
1 {
2   "sucesso": true,
3   "mensagem": "Vínculo atualizado com sucesso.",
4   "dados": {
5     "antigo": {
6       "idProduto": 1,
7       "idIngrediente": 6,
8       "adicional": 1 
9     },
10    "atualizado": {
11      "idProduto": 1,
12      "idIngrediente": 6,
13      "adicional": 0 
14    }
15  }
16 }
```


# Update dinâmico


```
200  async editarClientes(request, response) {
201      // receber a pontuação que deve ser adicionada e retornar dados de antes e depois da atualização
202      try {
203          const { id } = request.params;
204          const dados = request.body;
205
206          // Mapeamento dos campos válidos para o banco de dados
207          const camposValidos = {
208              cel: 'cli_cel',
209              pontos: 'cli_pts'
210          };
211
212          // Arrays para montar a query dinamicamente
213          const setClauses = [];
214          const values = [];
```

```
216 // Monta dinamicamente os campos a serem atualizados
217 // Para cada campo válido, adiciona a string nome_do_campo_banco = ? no array setClauses
218 for (const key in dados) {
219     // exemplo, se key = 'cel', e camposValidos['cel'] = 'cli_cel'
220     if (camposValidos[key] && dados[key] !== undefined) {
221         setClauses.push(`${camposValidos[key]} = ?`);
222         values.push(dados[key]);
223     }
224 }
225 // Depois que todos os campos foram processados (se todos os campos forem passados), temos:
226 // setClauses = ['cli_cel = ?', 'cli_pts = ?'];
227
228 // Se nenhum campo válido foi enviado, retorna erro
229 if (setClauses.length === 0) {
230     return response.status(400).json({
231         sucesso: false,
232         mensagem: 'Nenhum campo válido enviado para atualização.',
233         dados: null
234     });
235 }
236
237 // Adiciona o ID ao final dos valores (para a cláusula WHERE)
238 values.push(id);
```

```
240 // Monta a query final
241 // SET cli_cel = ?, cli_pts = ?
242 const sql = `
243     UPDATE clientes
244     SET ${setClauses.join(', ')}
245     WHERE usu_id = ?;
246 `;
247
248 // Executa a query
249 const [result] = await db.query(sql, values);
250
251 // Se nenhum registro foi alterado
252 if (result.affectedRows === 0) {
253     return response.status(404).json({
254         sucesso: false,
255         mensagem: `Cliente com ID ${id} não encontrado.`,
256         dados: null
257     });
258 }
259
260 // Sucesso
261 return response.status(200).json({
262     sucesso: true,
263     mensagem: 'Atualização de dados do cliente realizada com sucesso.',
264     dados: { id }
265 });
266
267 } catch (error) {
268     return response.status(500).json({
269         sucesso: false,
270         mensagem: 'Erro ao atualizar cliente.',
271         dados: error.message
272     });
273 }
274 }
```

# Teste


PATCH  http://localhost:3333/clientes/10

Send 

200 OK

94 ms

106 B

Params Body  Auth Headers 4 Scripts Docs


Preview Headers 8 Cookies Tests 0/0 → Mock Console


JSON 

```
1 {  
2   "pontos": 18,  
3   "cel": "18995821521"  
4 }  
5  
6
```

Preview 

```
1 {  
2   "sucesso": true,  
3   "mensagem": "Atualização de dados do cliente realizada com sucesso.",  
4   "dados": {  
5     "id": "10"  
6   }  
7 }
```


PATCH  http://localhost:3333/clientes/10

Send 

200 OK

45 ms

106 B

Params Body  Auth Headers 4 Scripts Docs


Preview Headers 8 Cookies Tests 0/0 → Mock Console


JSON 

```
1 {  
2   "pontos": 23  
3 }  
4  
5
```

Preview 

```
1 {  
2   "sucesso": true,  
3   "mensagem": "Atualização de dados do cliente realizada com sucesso.",  
4   "dados": {  
5     "id": "10"  
6   }  
7 }
```


PATCH  http://localhost:3333/clientes/10

Send 

200 OK

45 ms

106 B

Params Body  Auth Headers 4 Scripts Docs

Preview Headers 8 Cookies Tests 0/0 → Mock Console

JSON 

```
1 {  
2   "cel": "18995821519"  
3 }  
4  
5
```

Preview 


```
1 {  
2   "sucesso": true,  
3   "mensagem": "Atualização de dados do cliente realizada com sucesso.",  
4   "dados": {  
5     "id": "10"  
6   }  
7 }
```



# Exclusão de itens relacionados

```
210 async apagarProdutoIngredientes(request, response) {
211   try {
212     const { produto, ingrediente } = request.params;
213
214     if (!produto || !ingrediente) {
215       return response.status(400).json({
216         sucesso: false,
217         mensagem: 'Campos obrigatórios: produto e ingrediente.', ←
218         dados: null
219       });
220     }
221
222     if (isNaN(produto) || isNaN(ingrediente)) {
223       return response.status(400).json({
224         sucesso: false,
225         mensagem: 'Os campos produto e ingrediente devem ser números.', ←
226         dados: null
227       });
228     }
229
230     // Verificar se há vínculo para excluir
231     const [vinculo] = await db.query(
232       `
233       SELECT prd_id AS idProduto, ing_id AS idIngrediente, prd_ing_adicional = 1 AS adicional ←
234       FROM produto_ingredientes
235       WHERE prd_id = ? AND ing_id = ?
236       `,
237       [produto, ingrediente]
238     );
```

```
240     if (vinculo.length === 0) {
241         return response.status(404).json({
242             sucesso: false,
243             mensagem: 'Vínculo não encontrado.', ←
244             dados: null
245         });
246     }
247
248     // Executar exclusão
249     await db.query(
250         `DELETE FROM produto_ingredientes WHERE prd_id = ? AND ing_id = ?`,
251         [produto, ingrediente]
252     );
253
254     return response.status(200).json({
255         sucesso: true,
256         mensagem: 'Vínculo removido com sucesso.',
257         dados: vinculo[0] // retorna o que foi deletado como confirmação
258     });
259
260 } catch (error) {
261     return response.status(500).json({
262         sucesso: false,
263         mensagem: 'Erro ao excluir vínculo.',
264         dados: error.message
265     });
266 }
267 },
```

DELETE  http://localhost:3333/produto/1/ingrediente/5

Send 

200 OK

108 ms

116 B

Params Body Auth Headers 3 Scripts Docs

Preview Headers 8 Cookies Tests 0 / 0 → Mock Console

No Body 

Preview 

```
1 {  
2   "sucesso": true,  
3   "mensagem": "Vínculo removido com sucesso.",  
4   "dados": {  
5     "idProduto": 1,  
6     "idIngrediente": 5,  
7     "adicional": 0  
8   }  
9 }
```

DELETE  http://localhost:3333/produto/1/ingrediente/5

Send 

404 Not Found


22 ms

69 B

Params Body Auth Headers 3 Scripts Docs

Preview Headers 8 Cookies Tests 0 / 0 → Mock Console

No Body 

Preview 

```
1 {  
2   "sucesso": false,  
3   "mensagem": "Vínculo não encontrado.",  
4   "dados": null  
5 }
```

# Criptografar senha

- `npm i bcrypt`

```
D:\TEMP\ewerton-a\NODE\infonet-a-1sem-25-intro-api-node>npm i bcrypt

added 3 packages, and audited 118 packages in 2s

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

1 low severity vulnerability

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
```

# Adicionando recurso em cadastro de usuário

```
src > controllers > JS usuarios.js > [?] <unknown> > [?] listarUsuarios > [?] sql
```

```
1 const db = require('../database/connection');  
2 const bcrypt = require('bcrypt'); ←
```

```
async cadastrarUsuarios(request, response) {  
  try {  
    const { nome, email, dt_nasc, senha, tipo, cpf } = request.body;  
    const usu_ativo = 1;  
  
    const saltRounds = 10; ←  
    const hashedPassword = await bcrypt.hash(senha, saltRounds); ←  
  
    const sql = `  
      INSERT INTO usuarios  
      (usu_nome, usu_email, usu_dt_nasc, usu_senha, usu_tipo, usu_ativo, usu_cpf)  
      VALUES  
      (?, ?, ?, ?, ?, ?, ?);  
    `;  
  
    const values = [nome, email, dt_nasc, hashedPassword, tipo, usu_ativo, cpf];  
  
    const [result] = await db.query(sql, values);  
  
    return response.status(200).json({  
      sucesso: true,  
      mensagem: 'Cadastro de usuário efetuado com sucesso!',  
      dados: {  
        id: result.insertId,  
        nome,  
        email,  
        tipo  
      }  
    });  
  } catch (error) {  
    return response.status(500).json({  
      sucesso: false,  
      mensagem: 'Erro na requisição.',  
      dados: error.message  
    });  
  }  
},
```

# Generate a Salt

Para gerar o salt, chame o método `bcrypt.genSalt()`. Esse método aceita um valor inteiro que é o fator de custo que determina o tempo necessário para fazer o hash de uma senha. Quanto maior o fator de custo, mais tempo o algoritmo leva e mais difícil é reverter o hash usando força bruta. Um bom valor deve ser alto o suficiente para proteger a senha, mas também baixo o suficiente para não retardar o processo. Geralmente varia entre 5 e 15.

# Senha criptografada

[illegible]

# Validação da senha criptografada




```
270 async login(request, response) {
271   try {
272     const { email, senha } = request.query;
273
274     const sql = `
275       SELECT
276         usu_id, usu_nome, usu_tipo, usu_senha
277       FROM
278         usuarios
279       WHERE
280         usu_email = ? AND usu_ativo = 1; ←
281     `;
282
283     const [rows] = await db.query(sql, [email]);
284
285     if (rows.length === 0) {
286       return response.status(403).json({
287         sucesso: false,
288         mensagem: 'Email não encontrado ou usuário inativo.',
289         dados: null,
290       });
291     }
292
293     const usuario = rows[0]; ←
294     const senhaCorreta = await bcrypt.compare(senha, usuario.usu_senha); ←
```

Importância do campo e-mail como UNIQUE





```
295
296     if (!senhaCorreta) {
297         return response.status(403).json({
298             sucesso: false,
299             mensagem: 'Senha incorreta.',
300             dados: null,
301         });
302     }
303
304     return response.status(200).json({
305         sucesso: true,
306         mensagem: 'Login efetuado com sucesso',
307         dados: {
308             id: usuario.usu_id,
309             nome: usuario.usu_nome,
310             tipo: usuario.usu_tipo
311         }
312     });
313
314 } catch (error) {
315     return response.status(500).json({
316         sucesso: false,
317         mensagem: 'Erro na requisição.',
318         dados: error.message
319     });
320 }
321 },
```


# Teste de login

GET  http://localhost:3333/login?email=brenodobarro@email.com&senha=123  Send  200 OK 92 ms 104 B

Params Body Auth Headers 3 Scripts Docs

No Body 

Preview Headers 8 Cookies Tests 0 / 0 

Preview 

```
1 {  
2   "sucesso": true,  
3   "mensagem": "Login efetuado com sucesso",  
4   "dados": {  
5     "id": 8,  
6     "nome": "Breno Barros",  
7     "tipo": 1  
8   }  
9 }
```

Atualização de senha.  
Dependendo de como  
for solicitada a  
atualização de senha  
pode ser necessário  
criar uma forma de  
exigir a senha antiga  
para adicionar uma  
nova.

```
async atualizaSenha(request, response) {  
  try {  
    const { senha } = request.body;  
    const { id } = request.params;  
  
    const saltRounds = 10;  
    const hashedPassword = await bcrypt.hash(senha, saltRounds);  
  
    const sql = `  
      UPDATE usuarios  
      SET usu_senha = ?  
      WHERE usu_id = ?;  
    `;  
  
    const [result] = await db.query(sql, [hashedPassword, id]);  
  
    if (result.affectedRows === 0) {  
      return response.status(404).json({  
        sucesso: false,  
        mensagem: `Usuário ${id} não encontrado!`,  
        dados: null  
      });  
    }  
  
    return response.status(200).json({  
      sucesso: true,  
      mensagem: `Senha do usuário ${id} atualizada com sucesso!`,  
      dados: null  
    });  
  } catch (error) {  
    return response.status(500).json({  
      sucesso: false,  
      mensagem: 'Erro na requisição.',  
      dados: error.message  
    });  
  }  
},
```

```
router.patch('/usuarios/:id', UsuariosController.editarUsuarios); // params  
router.patch('/usuarios/atualiza-senha/:id', UsuariosController.atualizaSenha); // params  
router.delete('/usuarios/:id', UsuariosController.apagarUsuarios); // params
```

Ver o link abaixo sobre performance em criptografia de senha

- <https://hcode.com.br/blog/criptografia-em-node-js-com-a-lib-bcrypt>