CRUD EM API'S REST

O QUE É CRUD?

CRUD é um acrônimo que representa as 4 operações básicas em sistemas que manipulam dados:

- **C** - Create (Criar)
- **R** - Read (Ler)
- **U** - Update (Atualizar)
- **D** - Delete (Deletar)

Essas operações estão diretamente relacionadas às ações em **bancos de dados** e são implementadas em **rotas de uma API** para interagir com os dados de forma organizada e segura.

CRUD NO CONTEXTO DE UMA API REST

Cada requisição percorre um **pipeline** de middlewares. Visualize como um fluxo:

Operação	Verbo HTTP	Caminho/Rota	Ação
Create	`POST`	`/usuarios`	Criar um novo usuário
Read	`GET`	`/usuarios`	Listar todos os usuários
Read	`GET`	`/usuarios/:id`	Buscar usuário pelo ID
Update	`PUT` ou `PATCH`	`/usuarios/:id`	Atualizar um usuário
Delete	`DELETE`	`/usuarios/:id`	Deletar um usuário

EXEMPLOS PRÁTICOS COM NODE.JS + EXPRESS

1. **Create (POST */usuarios*)**

```
app.post("/usuarios", async (req, res) => {
 const { nome, email } = req.body;
 const novoUsuario = await Usuario.create({ nome, email });
 res.status(201).json(novoUsuario);
});
```

2. Read (GET */usuarios*)

```
app.get("/usuarios", async (req, res) => {
 const usuarios = await Usuario.findAll();
 res.json(usuarios);
});
```

EXEMPLOS PRÁTICOS COM NODE.JS + EXPRESS

3. Read por ID (GET */usuarios/:id*)

```
app.get("/usuarios/:id", async (req, res) => {
 const usuario = await Usuario.findByPk(req.params.id);
 if (!usuario) return res.status(404).json({ mensagem: "Usuário não encontrado." });
 res.json(usuario);
});
```

EXEMPLOS PRÁTICOS COM NODE.JS + EXPRESS

4. Update (PUT */usuarios/:id*)

```
app.put("/usuarios/:id", async (req, res) => {
 const { nome, email } = req.body;
 const usuario = await Usuario.findByPk(req.params.id);

 if (!usuario) return res.status(404).json({ mensagem: "Usuário não encontrado." });

 usuario.nome = nome;
 usuario.email = email;
 await usuario.save();

 res.json(usuario);
});
```

EXEMPLOS PRÁTICOS COM NODE.JS + EXPRESS

5. Delete (DELETE *`/usuarios/:id`*)

```
app.delete("/usuarios/:id", async (req, res) => {
 const usuario = await Usuario.findByPk(req.params.id);

 if (!usuario) return res.status(404).json({ mensagem: "Usuário não encontrado." });

 await usuario.destroy();
 res.status(204).send(); // No content
});
```

BOAS PRÁTICAS EM CRUD DE APIS

- Valide dados antes de criar ou atualizar (**Joi**, **Zod**, etc).
- Retorne mensagens de erro significativas.
- Use status HTTP corretos (**200**, **201**, **404**, **400**, **500**, etc.).
- Proteja rotas sensíveis com autenticação (ex: **JWT**).
- Não exponha informações sensíveis nos retornos.

FILTROS, BUSCAR E PAGINAÇÃO

FILTROS E BUSCAS (QUERY PARAMS)

Podemos buscar usuários com base em campos específicos, como **nome** ou **email**.

Exemplo: **GET */usuarios?nome=joao***

```
app.get("/usuarios", async (req, res) => {
 const { nome } = req.query;
 const usuarios = await Usuario.findAll({
   where: {
     nome: { [Op.like]: `%${nome}%` }
   }
 });
 res.json(usuarios);
});
```

PAGINAÇÃO

Paginar é essencial para desempenho e organização. O padrão mais comum é usar **?page=** e **?limit=**.

Exemplo: **GET *//usuarios?page=2&limit=10***

```
app.get("/usuarios", async (req, res) => {
 const page = parseInt(req.query.page) || 1;
 const limit = parseInt(req.query.limit) || 10;
 const offset = (page - 1) * limit;

 const usuarios = await Usuario.findAll({ limit, offset });
 res.json(usuarios);
});
```

THANK YOU

@wallace027dev