API REST com Node, Express e MySQL

Parte 4 - Criando Rotas GET e POST + Integração com API Externa

Objetivos:

- Usar a **pool de conexões** (`db.js`) dentro das rotas da API.
- Criar rotas **GET** para listar usuários.
- Criar rotas **POST** para inserir usuários.
- Adicionar uma rota que consome uma **API externa (PokeAPI)**.
- Testar as rotas no navegador e em ferramentas como **Postman** ou `curl`.

Conceitos Fundamentais

Rotas GET e POST

- **GET**: usado para **buscar dados** (não altera o servidor).
- **POST**: usado para **enviar dados** e criar um novo recurso.

Exemplo:

- `GET /users` → retorna lista de usuários.
- `POST /users` → cria um novo usuário no banco.

Integração com Banco de Dados

- Quando chamamos `pool.query(...)`, o Node envia um comando SQL para o MySQL.
- A resposta vem em formato de array com linhas (rows).

Exemplo:

```
const [rows] = await pool.query('SELECT * FROM users')
```

Integração com APIs Externas

- APIs podem consumir **outras APIs**.

- Exemplo: sua API pode buscar informações de um Pokémon na **PokeAPI** e devolver para o cliente já “filtradas”.

Isso mostra como uma API pode ser um **conector** entre várias fontes de dados.

Criando o Arquivo de Rotas

src/routes.js

```
import { Router } from 'express'
import { pool } from './db.js'
const r = Router()
// ----------------------------------------------------
// Rota de teste de conexão com o banco
// GET http://localhost:3000/api/db/health
// ----------------------------------------------------
r.get('/db/health', async (_, res) => {
  try {
    const [rows] = await pool.query('SELECT 1 AS db_ok')
    res.json({ ok: true, db: rows[0].db_ok })
  } catch {
    res.status(500).json({ ok: false, db: 'down' })
  }
})
// ----------------------------------------------------
// GET usuários
// GET http://localhost:3000/api/users
// ----------------------------------------------------
r.get('/users', async (_, res) => {
  try {
    const [rows] = await pool.query(
      'SELECT id, name, email, created_at FROM users ORDER BY id DESC'
    )
    res.json(rows)
  } catch {
    res.status(500).json({ error: 'Erro ao listar usuários' })
  }
})
// ----------------------------------------------------
// POST usuário
// POST http://localhost:3000/api/users
// Body JSON: { "name": "Maria", "email": "maria@teste.com" }
// ----------------------------------------------------
r.post('/users', async (req, res) => {
  const { name, email } = req.body
  // Validação básica
```

```
  if (!name || !email) {
    return res.status(400).json({ error: 'name e email obrigatórios' })
  }
  try {
    // Insere no banco
    const [ins] = await pool.query(
      'INSERT INTO users (name, email) VALUES (?, ?)',
      [name, email]
    )
    // Busca o usuário recém-criado
    const [rows] = await pool.query(
      'SELECT id, name, email, created_at FROM users WHERE id = ?',
      [ins.insertId]
    )
    res.status(201).json(rows[0])
  } catch (err) {
    if (err.code === 'ER_DUP_ENTRY') {
      return res.status(409).json({ error: 'email já cadastrado' })
    }
    res.status(500).json({ error: 'Erro ao criar usuário' })
  }
})
// Integração com API externa (PokeAPI)
// GET http://localhost:3000/api/pokemon/pikachu
// ------------------------------------------------
r.get('/pokemon/:name', async (req, res) => {
  const { name } = req.params
  try {
    const resp = await fetch(
      `https://pokeapi.co/api/v2/pokemon/${encodeURIComponent(name)}`
    )
    if (!resp.ok) {
      return res.status(404).json({ error: 'Pokémon não encontrado' })
    }
    const data = await resp.json()
    res.json({
      id: data.id,
      name: data.name,
      types: data.types.map(t => t.type.name)
    })
  } catch {
    res.status(500).json({ error: 'Erro ao consultar API externa' })
  }
})
export default r
```

Atualizando o App Principal

src/app.js

```
import express from 'express'
import routes from './routes.js'

const app = express()
app.use(express.json())

// health check do servidor
app.get('/health', (_, res) => res.json({ ok: true, server: 'up' }))

// monta todas as rotas em /api
app.use('/api', routes)

export default app
```

src/server.js

```
import 'dotenv/config'
import app from './app.js'

const port = process.env.PORT || 3000
app.listen(port, () => {
  console.log(`✅ API ON em http://localhost:${port}`)
})
```

Testando as Rotas

Abra no navegador:

```
http://localhost:3000/health
```

→ `{ "ok": true, "server": "up" }`

```
http://localhost:3000/api/db/health
```

→ `{ "ok": true, "db": 1 }`

```
http://localhost:3000/api/users
```

→ Lista de usuários cadastrados.

PokeAPI

```
http://localhost:3000/api/pokemon/pikachu
```

Resposta resumida:

```
{
  "id": 25,
  "name": "pikachu",
  "types": ["electric"]
}
```

O que você aprendeu nesta etapa

- Criar rotas **GET** e **POST** usando Express e MySQL.
- Retornar dados reais do banco.
- Tratar erros comuns (400, 409, 500).
- Integrar sua API com uma API externa.

Erros comuns

- **ECONNREFUSED** → servidor MySQL não está rodando.
- **ER_DUP_ENTRY** → tentativa de inserir email já existente.
- **Cannot GET /users** → esqueceu do prefixo `/api` (`/api/users`).
- **fetch is not defined** → Node precisa estar na versão 18+ para usar fetch nativo.