


API Livraria com Express.js

Criando uma API RESTful

 **Professor:** Fabricio Bizotto

 **Disciplina:** Desenvolvimento Web I

 **Curso:** Ciência da Computação

 **Fase:** 4ª fase

Objetivo

Criar uma **API REST de Livros** usando **Node.js + Express** com:

- Rotas CRUD
- Middleware de log
- Tratamento de erros
- Uso de variáveis de ambiente
- Execução com `nodemon`

Estrutura do Projeto

```
livraria/  
├── server.js  
├── .env                <-- Variáveis de ambiente  
├── package.json  
├── src/  
│   └── app.js  
├── routes/  
│   └── livros.routes.js  <-- Rotas de livros
```

Configuração Inicial

```
mkdir livraria  
cd livraria  
npm init -y  
npm install express dotenv  
npm install --save-dev nodemon
```



Configuração do `package.json`

```
"scripts": {  
  "start": "node server.js",  
  "dev": "nodemon server.js"  
}
```

👉 Assim podemos rodar:

- `npm run start` (produção)
- `npm run dev` (desenvolvimento)

Arquivo `.env`

```
NODE_ENV=development  
PORT=3000
```

👉 Define porta e ambiente de execução.



Arquivo `server.js`

```
// server.js
require('dotenv').config();
const app = require("./src/app");

const PORT = process.env.PORT || 3000;
const NODE_ENV = process.env.NODE_ENV || 'development';

app.listen(PORT, () => {
  console.log(`Servidor iniciado na porta ${PORT} (${NODE_ENV})`);
});
```

Arquivo `src/app.js`

1. Configuração do Express, middlewares e rotas.

```
// src/app.js
const express = require("express");
const app = express();

app.use(express.json()); // interpretar JSON no corpo da requisição
app.use(express.urlencoded({ extended: true })); // suportar dados de formulários

// Rotas
const livrosRoutes = require("../routes/livros.routes");

// Middleware de log
app.use((req, res, next) => {
  console.log(`📥 ${req.method} ${req.originalUrl}`);
  next();
});
```


Arquivo `src/app.js` (cont.)

2. Rota inicial e uso das rotas de livros.

```
app.get("/", (req, res) => {  
  res.status(200).json({  
    mensagem: "Bem-vindo à API da Livraria! Use /livros.",  
    rotas: {  
      listar: "GET /livros",  
      adicionar: "POST /livros",  
      obter: "GET /livros/:id",  
      atualizar: "PUT /livros/:id",  
      remover: "DELETE /livros/:id",  
      filtrar: "GET /livros/categoria/:categoria"  
    }  
  });  
});  
  
app.use("/livros", livrosRoutes);
```

Arquivo `src/app.js` (cont.)

3. Tratamento de erros genéricos (500).

```
app.use((err, req, res, next) => {  
  console.error('✖ Erro:', err.message);  
  
  if (process.env.NODE_ENV === 'development') {  
    res.status(500).json({  
      erro: "Erro interno",  
      mensagem: err.message,  
      stack: err.stack  
    });  
  } else {  
    res.status(500).json({ erro: "Erro interno do servidor" });  
  }  
});
```

Arquivo `src/app.js` (cont.)

4. Middleware para rotas não encontradas (404).

```
app.use((req, res) => {  
  res.status(404).json({ erro: "Endpoint não encontrado" });  
});
```



Arquivo `src/routes/livros.routes.js`

1. Configuração das rotas de livros.

```
const express = require("express");
const router = express.Router(); // Roteador do Express

let livros = [
  {
    id: 1,
    titulo: "Clean Code",
    autor: "Robert C. Martin",
    categoria: "Programação",
    ano: 2008
  },
  {
    id: 2,
    titulo: "O Programador Pragmático",
    autor: "Andrew Hunt",
    categoria: "Programação",
    ano: 1999
  }
];
```

Arquivo `src/routes/livros.routes.js`

2. Listar todos os livros (GET) com filtros opcionais.

```
router.get("/", (req, res) => {  
  const { titulo, categoria } = req.query;  
  let resultados = livros;  
  
  if (titulo) {  
    resultados = resultados.filter(l => l.titulo.toLowerCase().includes(titulo.toLowerCase()));  
  }  
  if (categoria) {  
    resultados = resultados.filter(l => l.categoria.toLowerCase() === categoria.toLowerCase());  
  }  
  
  res.status(200).json(resultados);  
});
```



Arquivo `src/routes/livros.routes.js`

3. Adicionar novo livro (POST) com validação simples.

```
router.post("/", (req, res) => {  
  const { titulo, autor, categoria, ano } = req.body;  
  
  if (!titulo || !autor || !categoria || !ano) {  
    return res.status(400).json({ erro: "Preencha todos os campos" });  
  }  
  
  const novoLivro = { id: livros.length + 1, titulo, autor, categoria, ano };  
  livros.push(novoLivro);  
  
  res.status(201).json({ mensagem: "Livro adicionado", data: novoLivro });  
});
```

Arquivo `src/routes/livros.routes.js`

4. Obter livro por ID (GET) com tratamento de erro 404.

```
router.get("/:id", (req, res) => {  
  const id = parseInt(req.params.id);  
  const livro = livros.find(l => l.id === id);  
  
  if (!livro) {  
    return res.status(404).json({ erro: "Livro não encontrado" });  
  }  
  res.status(200).json(livro);  
});
```



Arquivo `src/routes/livros.routes.js`

5. Atualizar livro por ID (PUT) com validação simples.

```
router.put("/:id", (req, res) => {  
  const id = parseInt(req.params.id);  
  const { titulo, autor, categoria, ano } = req.body;  
  
  if (!titulo || !autor || !categoria || !ano) {  
    return res.status(400).json({ erro: "Preencha todos os campos" });  
  }  
  
  const livro = livros.find(l => l.id === id);  
  if (!livro) return res.status(404).json({ erro: "Livro não encontrado" });  
  
  // Object.assign: atualiza o objeto existente  
  Object.assign(livro, { titulo, autor, categoria, ano });  
  res.status(200).json({ mensagem: "Atualizado com sucesso", data: livro });  
});
```


Arquivo `src/routes/livros.routes.js`

6. Remover livro por ID (DELETE) com tratamento de erro 404.

```
router.delete("/:id", (req, res) => {  
  const id = parseInt(req.params.id);  
  const index = livros.findIndex(l => l.id === id);  
  
  if (index === -1) return res.status(404).json({ erro: "Livro não encontrado" });  
  
  const removido = livros.splice(index, 1);  
  res.status(200).json({ mensagem: "Livro removido", data: removido[0] });  
});
```

Arquivo `src/routes/livros.routes.js`

6. Alternativa para remover livro mas não remover do array (DELETE).

```
router.delete("/:id", (req, res) => {  
  const id = parseInt(req.params.id);  
  const index = livros.findIndex(l => l.id === id);  
  
  if (index === -1) return res.status(404).json({ erro: "Livro não encontrado" });  
  livros[index].removido = true; // Marca como removido  
  res.status(200).json({ mensagem: "Livro marcado como removido", data: livros[index] });  
});
```

Assim mantemos o histórico de livros. Mas, ao listar, podemos filtrar os removidos.

Arquivo `src/routes/livros.routes.js`

7. Filtrar livros por categoria (GET).

```
router.get("/categoria/:categoria", (req, res) => {  
  const categoria = req.params.categoria;  
  const filtrados = livros.filter(l => l.categoria.toLowerCase() === categoria.toLowerCase());  
  res.status(200).json(filtrados);  
});
```

Executando a API

```
npm run dev
```

Acesse `http://localhost:3000` no navegador ou use o Postman/Insomnia para testar as rotas.