


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 (Nova Arquitetura Modular)

```
livraria/
├── server.js           <-- Ponto de entrada
├── .env               <-- Variáveis de ambiente
├── package.json
├── src/
│   ├── app.js         <-- Orquestração das rotas e middlewares
│   ├── config/
│   │   └── express.js  <-- Configuração do Express + Morgan
│   ├── middleware/
│   │   └── errorHandler.js <-- Tratamento de erros
│   └── routes/
│       ├── index.js    <-- Centralizador de rotas
│       └── livros.routes.js <-- Rotas específicas de livros
```

Configuração Inicial

```
mkdir livraria  
cd livraria  
npm init -y  
npm install express dotenv morgan  
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`

```
// Carrega as variáveis de ambiente do arquivo .env
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`

Orquestração principal da aplicação:

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

// Todas as rotas da aplicação (centralizadas)
const routes = require("../routes");

// Configura o middleware de tratamento de erros
const errorHandler = require("../middleware/errorHandler");

// Configura as rotas com prefixo /api
app.use("/api", routes);

// Middleware de tratamento de erros
app.use(errorHandler);

// Handler para rotas não encontradas (404)
app.use((req, res) => {
  res.status(404).json({ erro: "Endpoint não encontrado" });
});

module.exports = app;
```


⚙️ Arquivo `src/config/express.js`

Configuração simplificada do Express:

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

const app = express();

// Middleware básicos do Express
app.use(express.json()); // Middleware para interpretar JSON
app.use(express.urlencoded({ extended: true })); // Suporte para dados de formulários
app.use(morgan("combined")); // Logging HTTP

module.exports = app;
```

Arquivo `src/routes/index.js` (Novo!)

Centralizador de todas as rotas:

```
// src/routes/index.js
const express = require("express");
const router = express.Router();

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

// Rota inicial (explicação do sistema)
router.get("/", (req, res) => {
  res.status(200).json({
    mensagem: "Bem-vindo à API da Livraria! Use /livros para gerenciar os livros.",
  });
});

// Usa as rotas de livros
router.use("/livros", livrosRoutes);

module.exports = router;
```

✗ Arquivo `src/middleware/errorHandler.js`

Tratamento de erros simplificado:

```
// Este middleware captura e trata todos os erros da aplicação
const errorHandler = (err, req, res, next) => {
  if (process.env.NODE_ENV === 'development') {
    res.status(500).json({
      erro: "Erro interno do servidor",
      mensagem: err.message,
      stack: err.stack,
      timestamp: new Date().toISOString(),
      url: req.originalUrl,
      method: req.method
    });
  } else {
    res.status(500).json({
      erro: "Erro interno do servidor",
      timestamp: new Date().toISOString()
    });
  }
};

module.exports = errorHandler;
```

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

Rotas específicas para gerenciar 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
  }
];
```

Rotas CRUD (Continuam Iguais)

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);  
});
```

Rotas CRUD (cont.)

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 });  
});
```

Rotas CRUD (cont.)

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);  
});
```

Rotas CRUD (cont.)

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 });  
});
```


Rotas CRUD (cont.)

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] });  
});
```

Rotas CRUD (cont.)

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);  
});  
  
module.exports = router;
```

Executando a API

```
npm run dev
```





Rotas disponíveis:

- Rota inicial: `http://localhost:3000/api/`
- Listar livros: `http://localhost:3000/api/livros`
- Adicionar livro: `POST http://localhost:3000/api/livros`
- Obter livro: `GET http://localhost:3000/api/livros/:id`
- Atualizar livro: `PUT http://localhost:3000/api/livros/:id`
- Remover livro: `DELETE http://localhost:3000/api/livros/:id`

Principais Melhorias Implementadas



Arquitetura:

-  **Separação clara** de responsabilidades
-  **Roteamento centralizado** com `routes/index.js`
-  **Configuração isolada** em `config/express.js`
-  **Prefixo** `/api` para organização de endpoints