


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         <-- Configuração principal (simplificado)
│   ├── config/
│   │   └── express.js  <-- Configuração do Express + Morgan
│   ├── middleware/
│   │   └── errorHandler.js <-- Tratamento de erros
│   └── routes/
│       └── livros.routes.js <-- Rotas de livros
```

✨ Benefícios da nova estrutura:

- Separação de responsabilidades
- Código modular e reutilizável
- Fácil manutenção e testes
- **Morgan para logging profissional**

Configuração Inicial

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

✨ **Morgan:** Biblioteca profissional para logging HTTP`



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` (Refatorado)

1. Nova abordagem modular: Importa configurações de arquivos especializados.

```
// src/app.js
const { createExpressApp, setupRoutes, setupErrorHandling } = require("../config/express");

// Cria a instância do Express com configurações básicas
const app = createExpressApp();

// Configura todas as rotas da aplicação
setupRoutes(app);

// Configura o tratamento de erros e handlers finais
setupErrorHandling(app);

module.exports = app;
```


⚙️ Arquivo `src/config/express.js`

Configuração centralizada com Morgan:

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

function createExpressApp() {
  const app = express();

  // Middleware básicos do Express
  app.use(express.json());
  app.use(express.urlencoded({ extended: true }));

  // Configuração do Morgan baseada no ambiente
  if (process.env.NODE_ENV === 'development') {
    // Formato detalhado para desenvolvimento
    app.use(morgan('dev'));
  } else {
    // Formato comum para produção (mais compacto)
    app.use(morgan('common'));
  }

  // Rota para favicon (evita logs desnecessários)
  app.get('/favicon.ico', (req, res) => {
    res.status(204).end();
  });

  return app;
}
```

⚙️ Arquivo `src/config/express.js` (cont.)

3. Configuração das rotas:

```
function setupRoutes(app) {  
  const livrosRoutes = require("../routes/livros.routes");  
  
  // Rota inicial (explicação do sistema)  
  app.get("/", (req, res) => {  
    const response = {  
      mensagem: "Bem-vindo à API da Livraria!",  
    };  
  
    res.status(200).json(response);  
  });  
  
  // Configura as rotas para livros  
  app.use("/livros", livrosRoutes);  
  
  // Adicionar outros grupos de rotas aqui se necessário  
}
```

✗ Arquivo `src/middleware/errorHandler.js`

```
// src/middleware/errorHandler.js
const errorHandler = (err, req, res, next) => {
  console.error('✗ Erro capturado:', err.message);

  if (process.env.NODE_ENV === 'development') {
    // Em desenvolvimento: retorna detalhes completos do erro
    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 {
    // Em produção: retorna apenas mensagem genérica
    res.status(500).json({
      erro: "Erro interno do servidor",
      timestamp: new Date().toISOString()
    });
  }
};

module.exports = errorHandler;
```

❖ Por que Morgan é Melhor que Logger Customizado?

✅ Vantagens do Morgan:

- 🏭 **Padrão da indústria** - Usado por milhões de desenvolvedores
- 🎨 **Formatos predefinidos** - `dev`, `common`, `tiny`, `combined`
- ⚡ **Performance otimizada** - Código testado e otimizado
- 🛠️ **Configuração flexível** - Tokens personalizáveis
- 📁 **Suporte a arquivos** - Logs para arquivos em produção

❌ Problemas do Logger Customizado:

- 🔄 **Reinventar a roda** - Código desnecessário para manter
- ⚠️ **Funcionalidades limitadas** - Sem colorização, métricas, etc.
- 🐛 **Bugs potenciais** - Código não testado extensivamente

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

As rotas permanecem iguais, mas agora estão melhor organizadas na nova estrutura:

```
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; // ➡ Não esqueça de exportar!
```

Executando a API

```
npm run dev
```

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

Comparação: Antes vs Depois

✗ Estrutura Anterior:

- app.js: 60+ linhas (tudo misturado)
- Configuração, rotas, erros no mesmo arquivo
- Difícil manutenção e teste

✓ Nova Estrutura:

- app.js: 15 linhas (apenas orquestração)
- config/express.js: Configuração isolada
- middleware/: Funcionalidades específicas