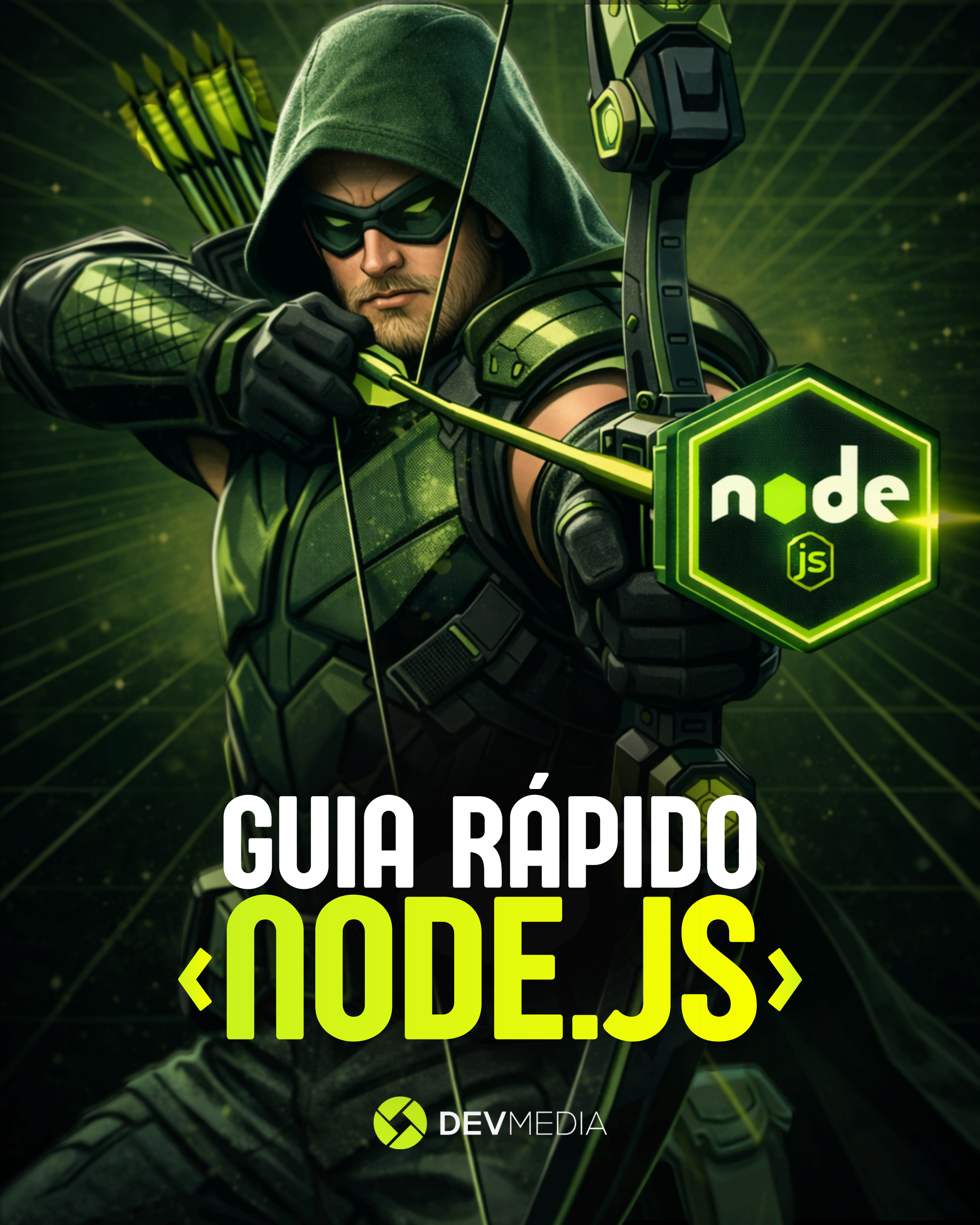


GUIA RÁPIDO
‹NODE.JS›

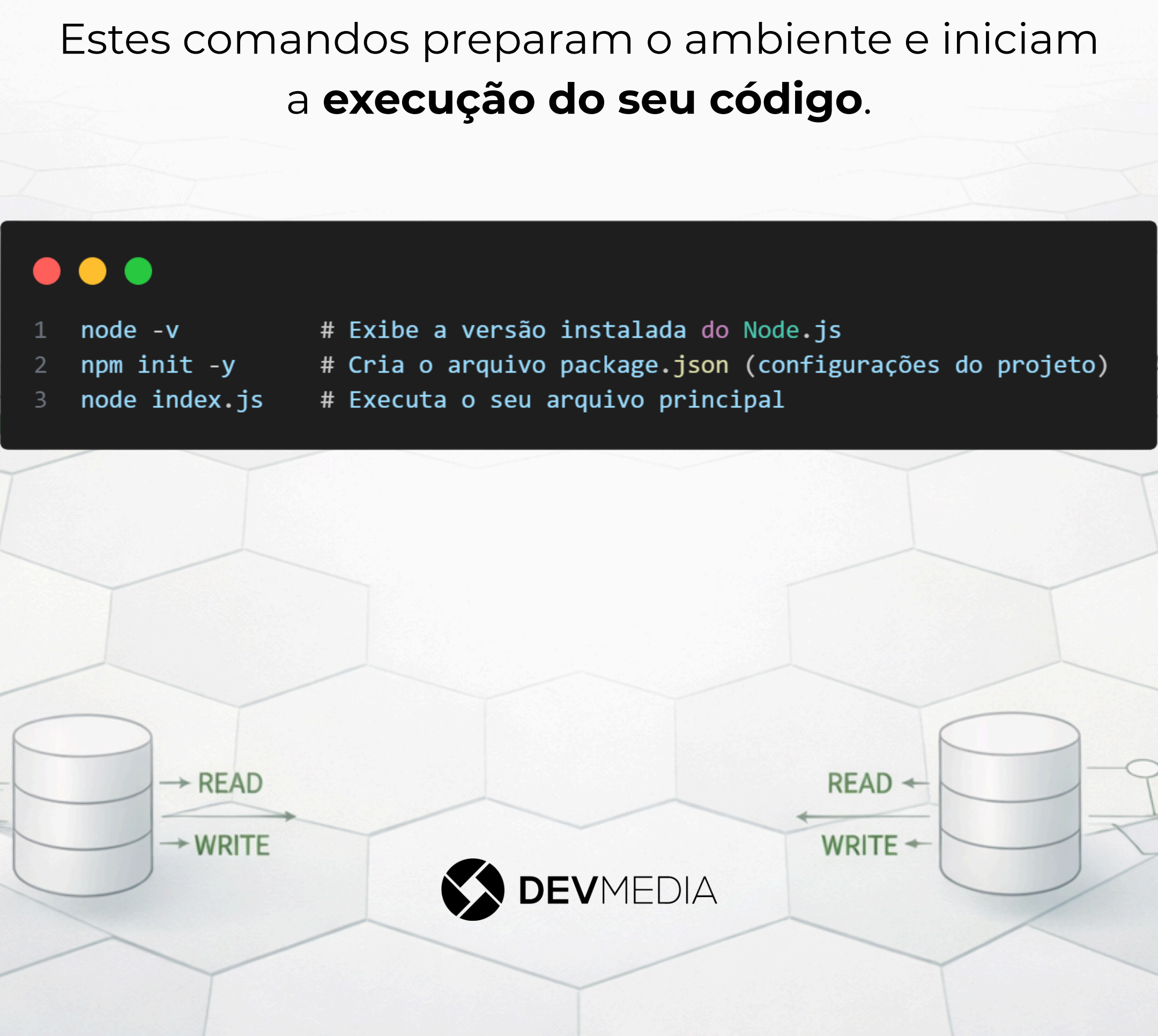
DEVMEDIA

Setup e Execução

Para começar um projeto, você precisa verificar a instalação e preparar o **arquivo de configuração**.

Estes comandos preparam o ambiente e iniciam a **execução do seu código**.

```
node -v          # Exibe a versão instalada do Node.js
npm init -y      # Cria o arquivo package.json (configurações do projeto)
node index.js    # Executa o seu arquivo principal
```



Módulos do Sistema (FS)

O Node.js permite que você manipule arquivos diretamente no **sistema operacional** do servidor.

Com o módulo 'fs', você consegue **ler e escrever dados** em arquivos de texto de forma simples.

```
1  const fs = require('fs'); // Importa o módulo de sistema de arquivos
2
3  // Lê o conteúdo de um arquivo existente
4  fs.readFile('dados.txt', 'utf8', (err, conteudo) => {
5    console.log(conteudo); // Exibe o texto lido no console
6  });
7
8  // Cria um novo arquivo com o texto informado
9  fs.writeFile('log.txt', 'Executado com sucesso', () => {});
```

Criando um Servidor (Express)

O **Express** é a biblioteca mais utilizada para transformar o Node.js em um **servidor web funcional.**

Ele gerencia as rotas e define o que o usuário recebe ao acessar o endereço do seu site.

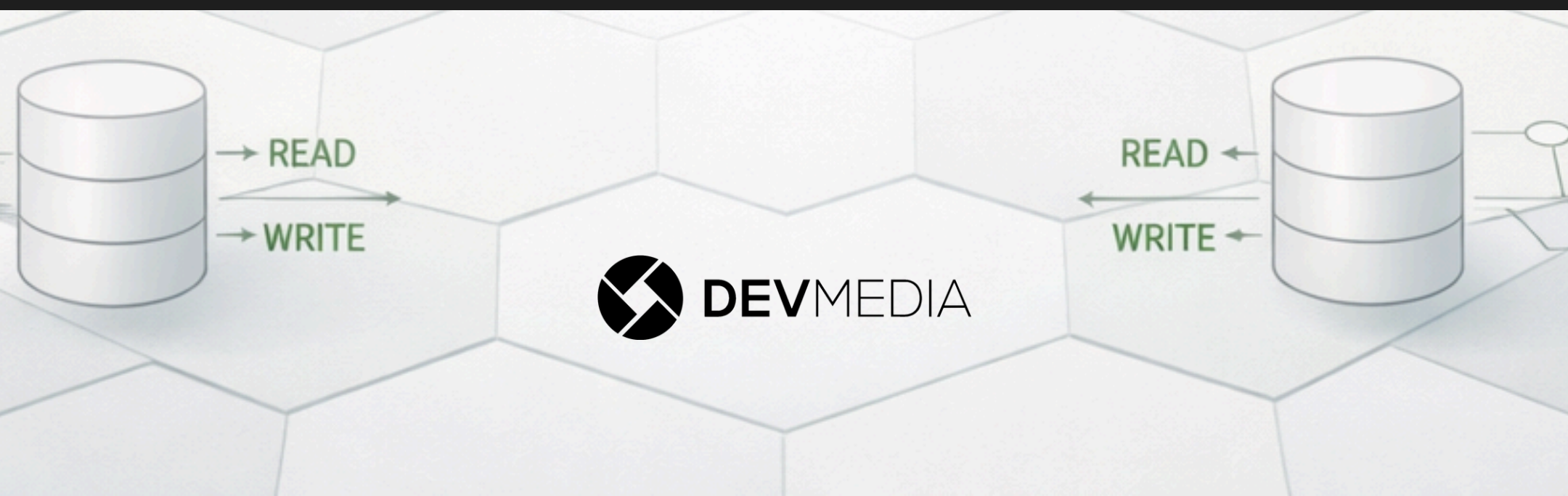
```
1  const express = require('express'); // Importa o framework
2  const app = express();              // Inicia a aplicação
3
4  // Define o que acontece quando alguém acessa o site
5  app.get('/', (req, res) => {
6    res.send('Servidor Ativo! 🚀'); // Resposta enviada ao navegador
7  });
8
9  app.listen(3000); // Define a porta de acesso (ex: localhost:3000)
```

Middlewares e JSON

Middlewares são funções que rodam antes da resposta final para **processar dados ou validar acessos.**

É aqui que configuramos o servidor para entender dados enviados no formato JSON.

```
1  app.use(express.json()); // Habilita o servidor a ler objetos JSON
2
3  // Função que roda em cada acesso para registrar logs
4  app.use((req, res, next) => {
5    console.log(`Acesso em: ${req.url}`); // Registra a URL acessada
6    next(); // Libera o fluxo para a rota final
7  });
```



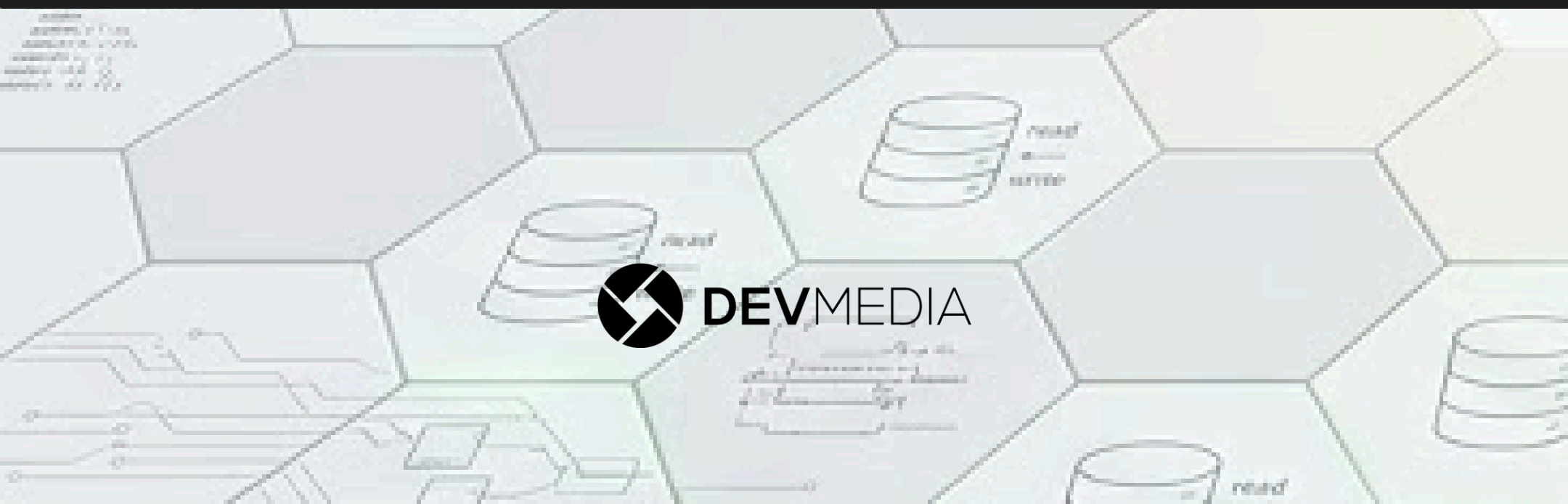
Rotas de API (CRUD)

Uma API organizada divide as funcionalidades pelos métodos HTTP (GET, POST, PUT, DELETE).

Cada método define uma ação diferente: **buscar, criar, atualizar ou remover dados**.

```
// Estrutura padrão de rotas para um recurso (ex: usuários)

app.get('/users', listar);     // Método GET para leitura
app.post('/users', criar);     // Método POST para criação
app.put('/users/:id', editar); // Método PUT para atualização
app.delete('/users/:id', remover); // Método DELETE para exclusão
```



Variáveis de Ambiente (.env)

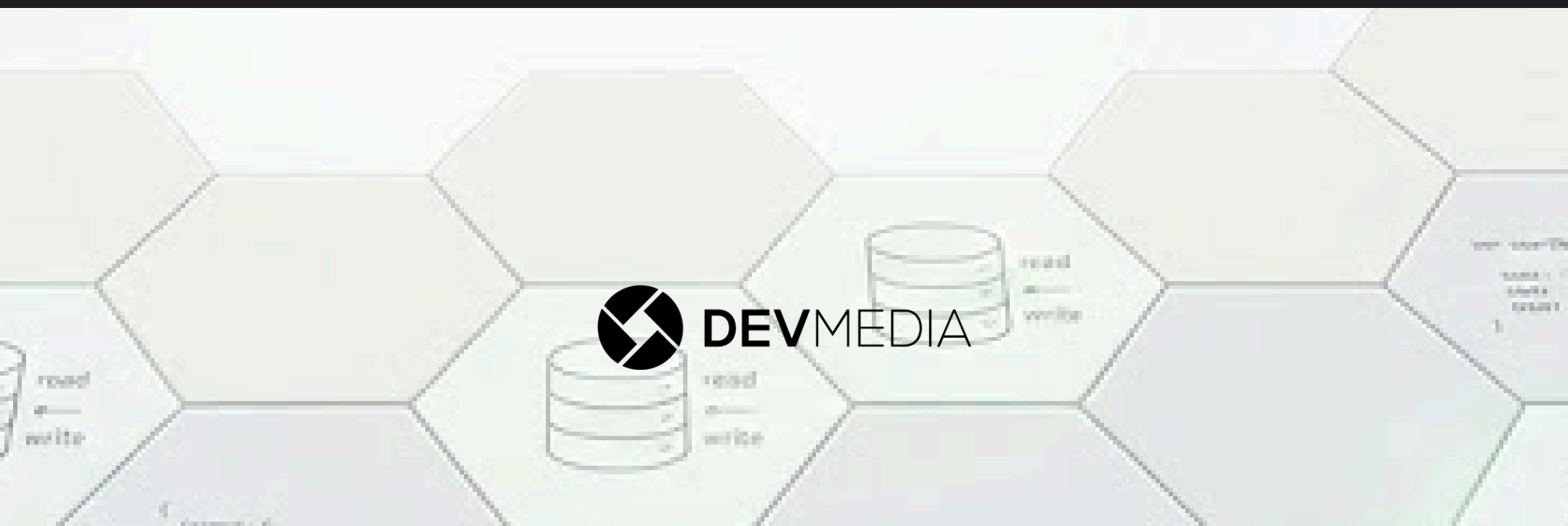
Dados sensíveis como senhas e portas de acesso devem ficar em um **arquivo separado e protegido.**

O uso do **'.env'** evita que informações críticas **fiquem expostas** diretamente no seu código-fonte.

```
// Arquivo .env: PORTA=5000

require('dotenv').config(); // Carrega as variáveis protegidas

const porta = process.env.PORTA; // Acessa o valor definido no .env
console.log(`Porta: ${porta}`); // Resultado: Porta: 5000
```



Autenticação com JWT

O JSON Web Token (JWT) é o padrão para **identificar usuários** logados em aplicações modernas.

Ele gera um **token criptografado** que serve como uma identidade digital para o cliente.

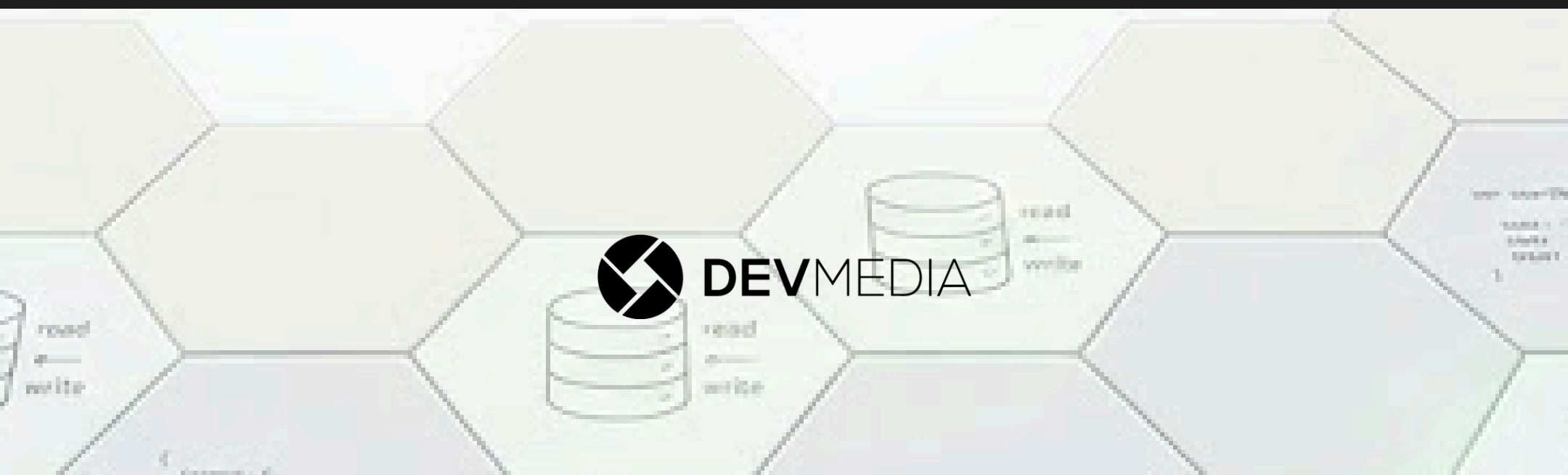
```
1  const jwt = require('jsonwebtoken'); // Biblioteca de autenticação
2
3  // Cria um token único que expira em 1 hora
4  const token = jwt.sign({ id: 1 }, 'segredo', { expiresIn: '1h' });
5
6  // Valida se o token recebido é legítimo e seguro
7  jwt.verify(token, 'segredo', (err, user) => {
8    if (err) return console.log('Acesso negado');
9    console.log('Logado como:', user.id);
10 });
```

Operações Assíncronas (Fetch)

Requisições para servidores externos levam tempo e por isso usamos funções assíncronas.

O uso de **'async/await'** permite que o código espere a **resposta da API** sem travar o sistema.

```
1  // Função preparada para esperar respostas externas
2  async function buscarDados() {
3    try {
4      const res = await fetch('https://api.exemplo.com'); // Aguarda a conexão
5      const dados = await res.json(); // Aguarda a conversão dos dados
6      console.log(dados); // Exibe o resultado final
7    } catch (erro) {
8      console.error('Falha na requisição'); // Trata possíveis erros
9    }
10 }
```



Reinicialização Automática (Nodemon)

Durante o desenvolvimento, reiniciar o servidor manualmente a cada mudança é ineficiente.

O Nodemon **monitora seus arquivos e reinicia o processo** automaticamente sempre que você salva o código.

```
npm install -D nodemon # Instala como ferramenta de auxílio

# No package.json: "scripts": { "dev": "nodemon index.js" }

npm run dev # Comando para rodar o projeto com auto-reload ativado
```

Curtiu o post?

Aqui na DevMedia, você aprende Node.js na prática, construindo projetos reais e resolvendo exercícios **direto no código**.

Clique na **legenda** para testar nossa plataforma de graça !

DEVMEDIA