Tutorial NODE.JS

🛠️ O que é o Node.js?

Com certeza! O Node.js costuma causar uma certa confusão no início porque ele não é uma linguagem de programação, mas sim um "ambiente" que permite que o JavaScript (que nasceu para rodar apenas no navegador) ganhe superpoderes no computador ou servidor.

Analogia

Imagine que o **JavaScript** é um piloto de corrida. Por muito tempo, ele só podia correr em uma pista específica: o **Navegador** (Chrome, Firefox, Edge).

O **Node.js** é como se alguém tivesse construído uma estrada de alta performance fora da pista, permitindo que esse mesmo piloto corra por cidades, desertos e montanhas.

Tecnicamente falando:

- É um **ambiente de execução** (runtime) de código aberto.
- É construído sobre o motor **V8** do Google Chrome (o que o torna extremamente rápido).
- Permite que você use JavaScript no **Backend** (servidores, bancos de dados, arquivos).

Características principais:

- **Single-threaded:** Ele executa uma tarefa por vez em um único "fio" de processamento.
- **I/O Não Bloqueante:** Diferente de outras linguagens, o Node não fica parado esperando um arquivo carregar ou uma busca no banco de dados terminar. Ele envia o pedido e continua trabalhando em outras tarefas até que a resposta chegue.

🚀 Para que serve o Node.js?

O Node.js brilha em cenários onde a velocidade e a capacidade de lidar com muitas conexões simultâneas são essenciais. Ele é ideal para:

1. **APIs REST:** Criar a "ponte" que conecta o banco de dados ao seu aplicativo.
2. **Aplicações em Tempo Real:** Chats (como o WhatsApp Web), jogos online ou ferramentas colaborativas (como o Google Docs).

3. **Streaming:** Plataformas de vídeo ou áudio (como Netflix e Spotify utilizam partes em Node).
4. **Ferramentas de Automação:** Scripts para otimizar tarefas repetitivas no computador.
5. **Ferramentas de Linha de Comando:** Automações que rodam direto no seu terminal.

Como ele funciona (A "Mágica")

A principal característica do Node é ser **Single-threaded** e **Non-blocking I/O**.

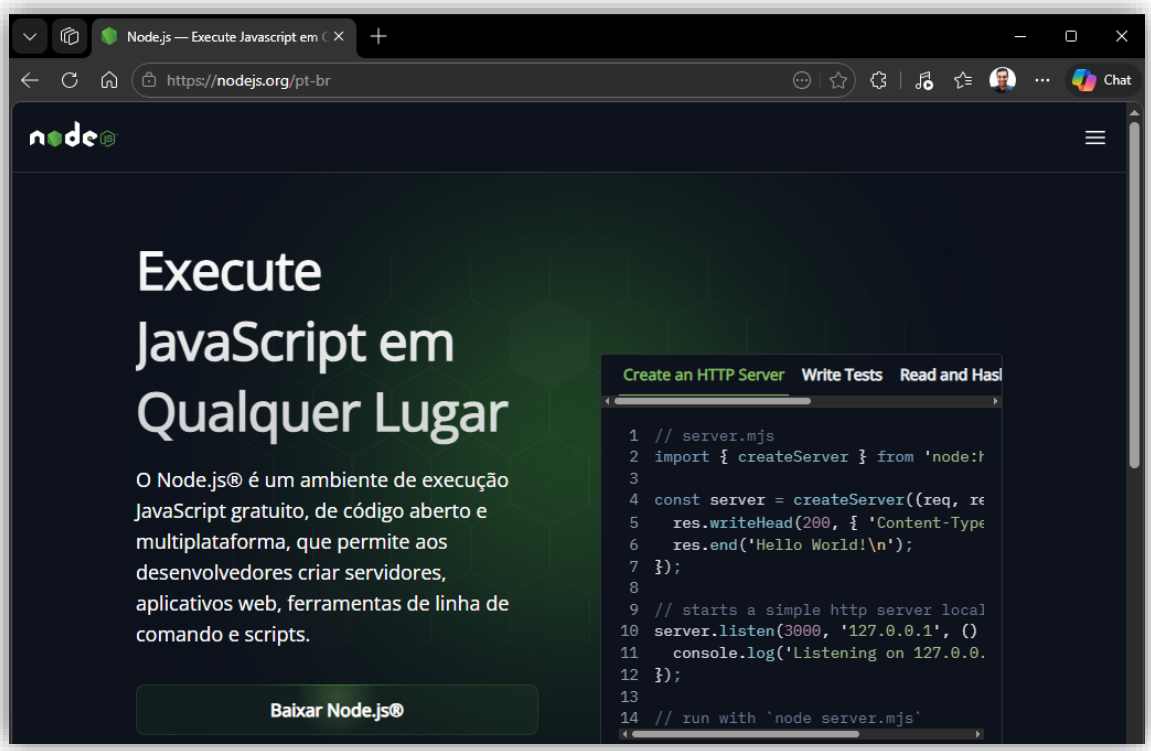
Diferente de outras tecnologias que param tudo para esperar uma resposta do banco de dados, o Node.js envia o pedido e já passa para a próxima tarefa. Quando o banco responde, ele avisa: "Ei, terminei aqui!", e o Node processa o resultado.

Tutorial: Primeiros Passos

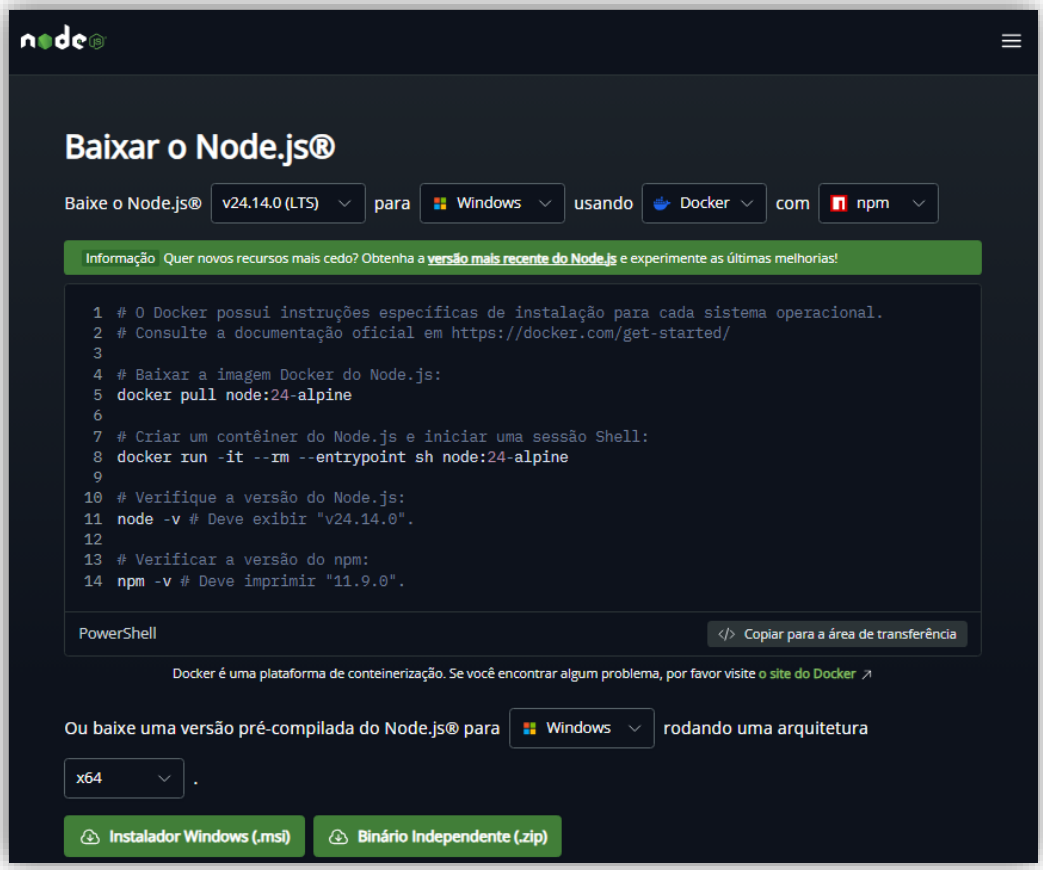
Se você quer ver o Node em ação agora, siga este roteiro:

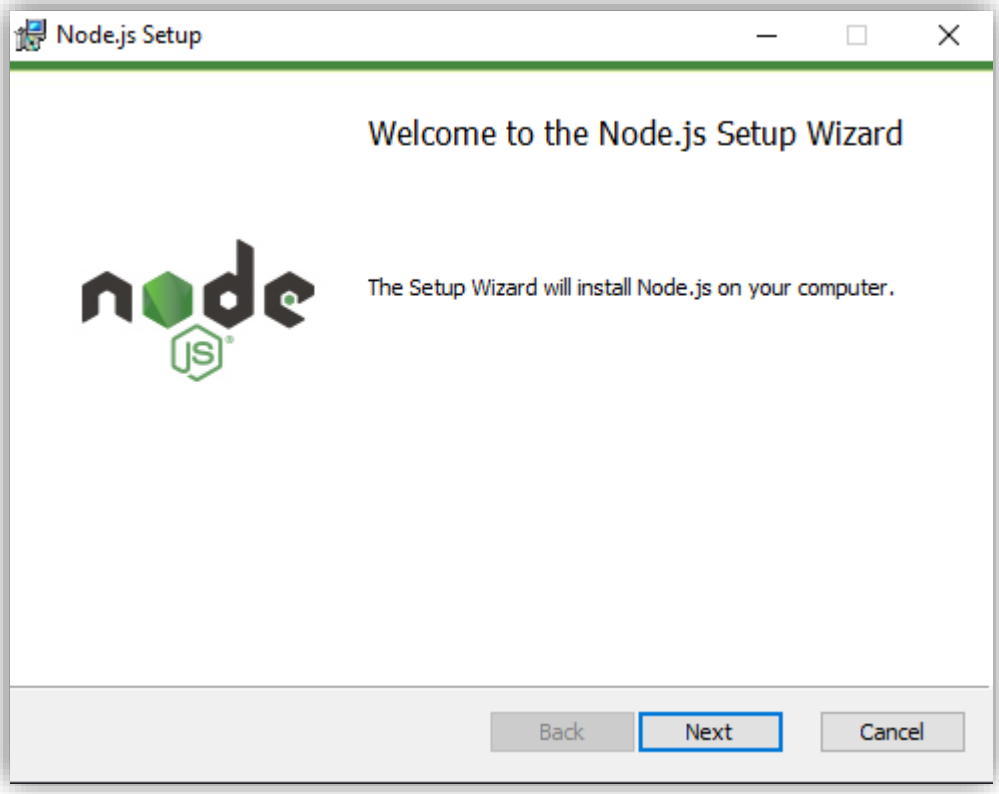
Passo A: Instalação

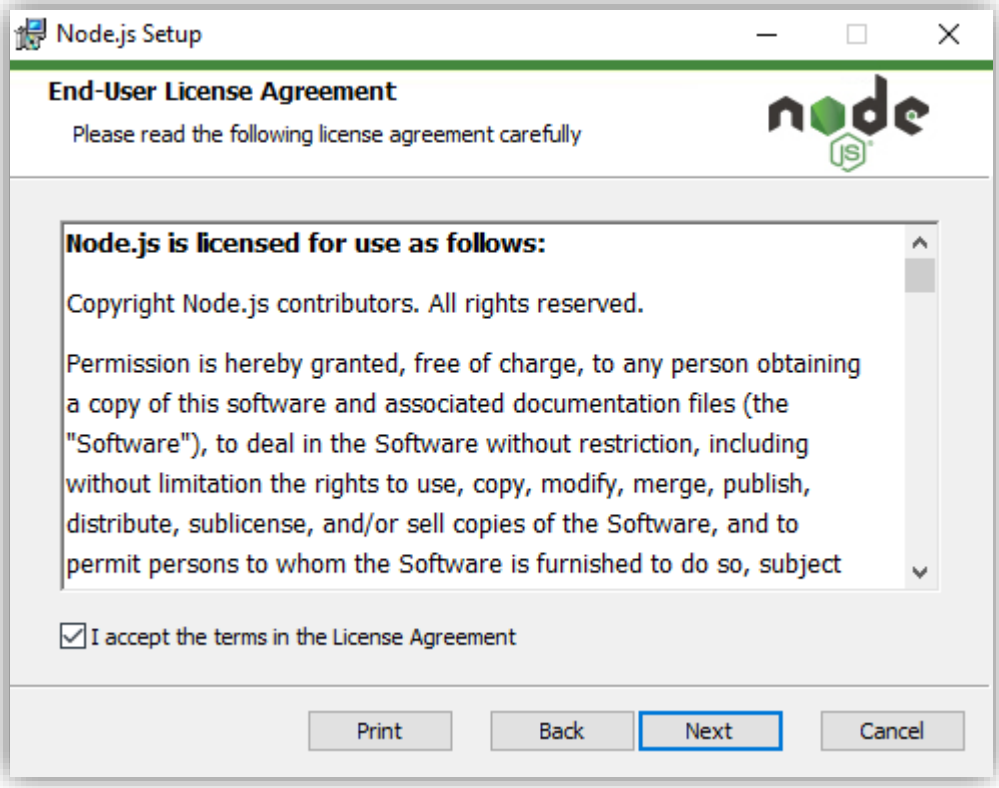
1. Vá ao site oficial nodejs.org.

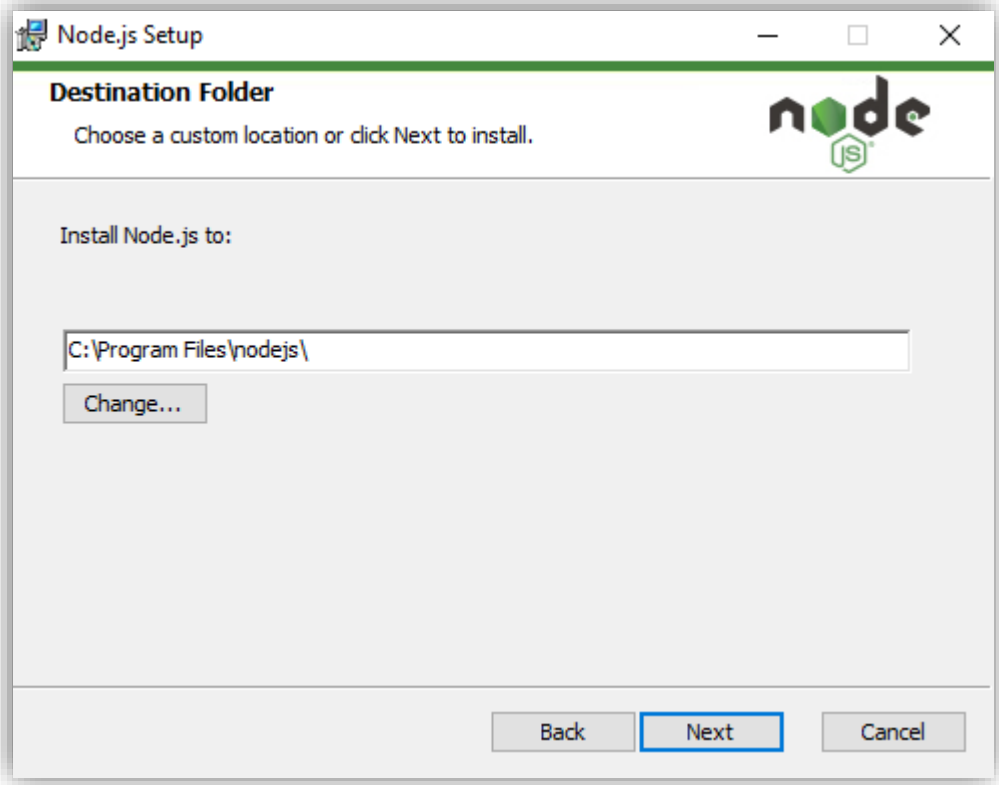


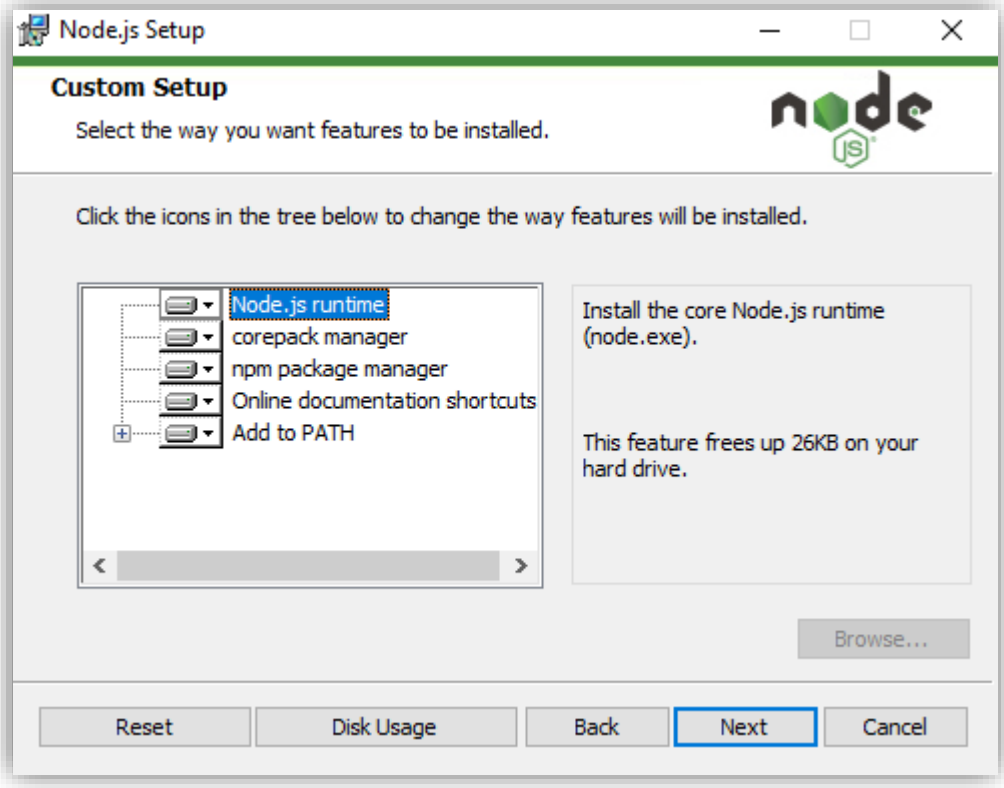
2. Baixe a versão **LTS** (Long Term Support), que é a mais estável para ensino e produção.

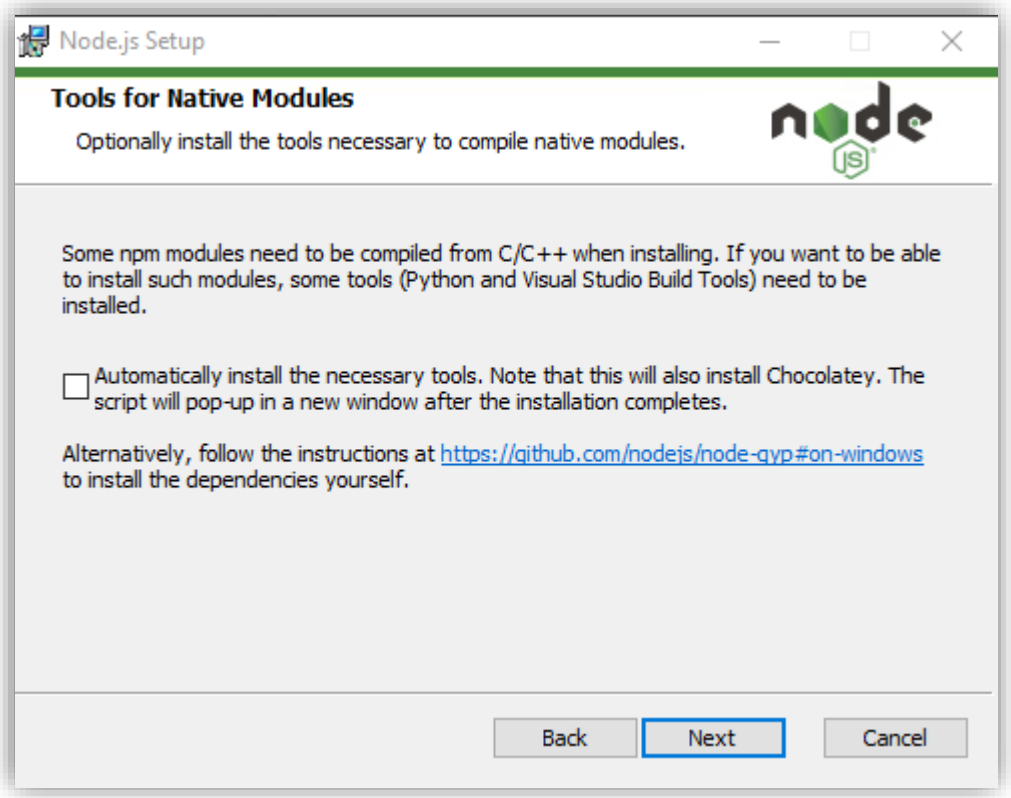

Baixar o Node.js®
Baixe o Node.js® v24.14.0 (LTS) para Windows usando Docker com npm
Informação Quer novos recursos mais cedo? Obtenha a versão mais recente do Node.js e experimente as últimas melhorias!
1 # O Docker possui instruções específicas de instalação para cada sistema operacional.
2 # Consulte a documentação oficial em https://docker.com/get-started/
3
4 # Baixar a imagem Docker do Node.js:
5 docker pull node:24-alpine
6
7 # Criar um contêiner do Node.js e iniciar uma sessão Shell:
8 docker run -it --rm --entrypoint sh node:24-alpine
9
10 # Verifique a versão do Node.js:
11 node -v # Deve exibir "v24.14.0".
12
13 # Verificar a versão do npm:
14 npm -v # Deve imprimir "11.9.0".
PowerShell
Copiar para a área de transferência
Docker é uma plataforma de conteinerização. Se você encontrar algum problema, por favor visite o site do Docker
Ou baixe uma versão pré-compilada do Node.js® para Windows rodando uma arquitetura
x64 .
Instalador Windows (.msi)
Binário Independente (.zip)

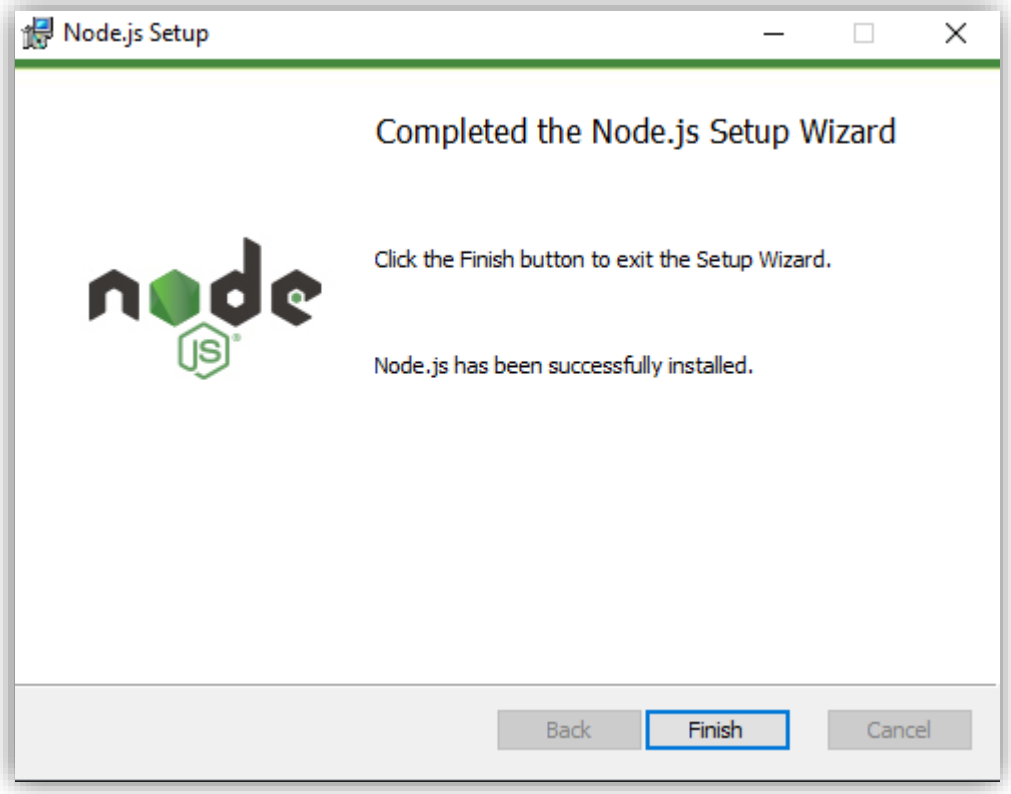

Node.js Setup
Welcome to the Node.js Setup Wizard
The Setup Wizard will install Node.js on your computer.
Back
Next
Cancel


Node.js Setup
End-User License Agreement
Please read the following license agreement carefully
Node.js is licensed for use as follows:
Copyright Node.js contributors. All rights reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject
I accept the terms in the License Agreement
Print
Back
Next
Cancel

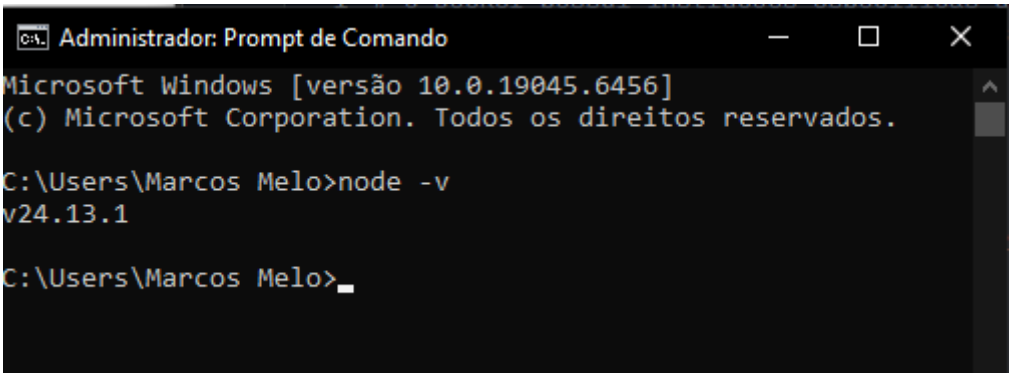

Node.js Setup
Destination Folder
Choose a custom location or click Next to install.
Install Node.js to:
C:\Program Files\nodejs\
Change...
Back
Next
Cancel


Node.js Setup
Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
corepack manager
npm package manager
Online documentation shortcuts
Add to PATH
Install the core Node.js runtime (node.exe).
This feature frees up 26KB on your hard drive.
Browse...
Reset
Disk Usage
Back
Next
Cancel





3. Após instalar, abra o seu terminal (CMD ou PowerShell) e digite: **node -v**. Se aparecer a versão, está tudo certo.



Passo B: Criando seu primeiro servidor

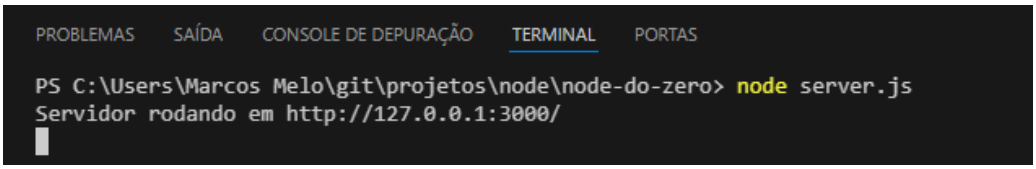
Não é necessário um servidor externo (como Apache). O Node **é** o servidor.

1. Crie uma pasta para o projeto.
2. Crie um arquivo chamado server.js.
3. Cole o seguinte código (exemplo simples de servidor HTTP):

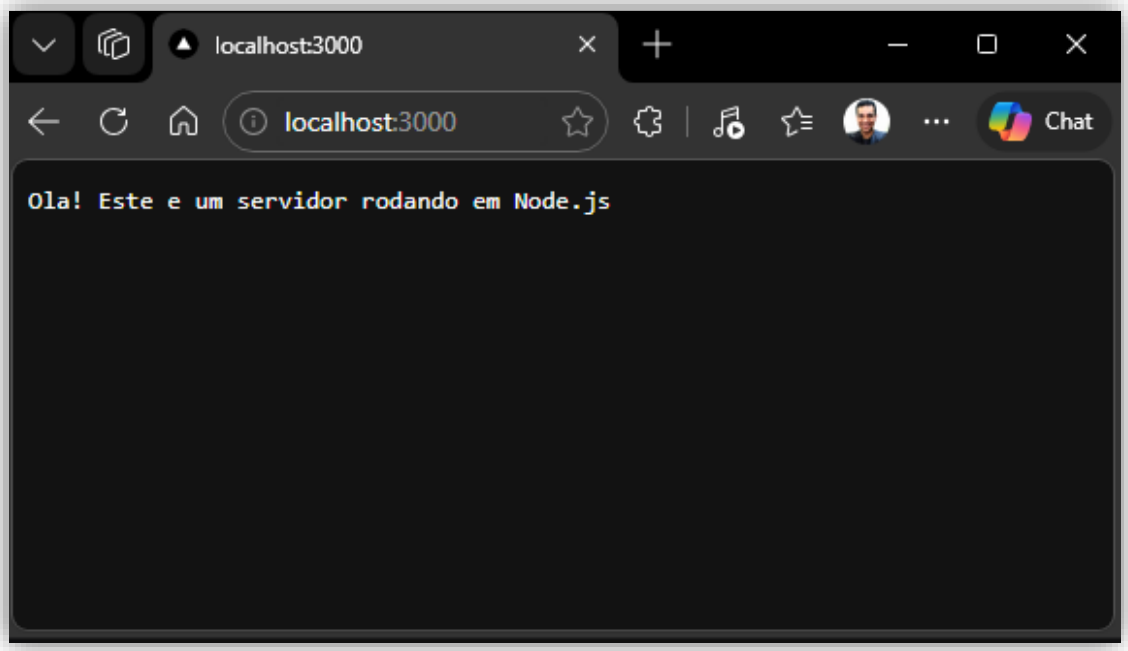
```
const http = require('http');

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Ola! Este e um servidor rodando em Node.js');
});

server.listen(3000, '127.0.0.1', () => {
  console.log('Servidor rodando em http://127.0.0.1:3000/');
});
```

Passo C: Executando

No terminal, dentro da pasta do arquivo, digite: **node server.js**

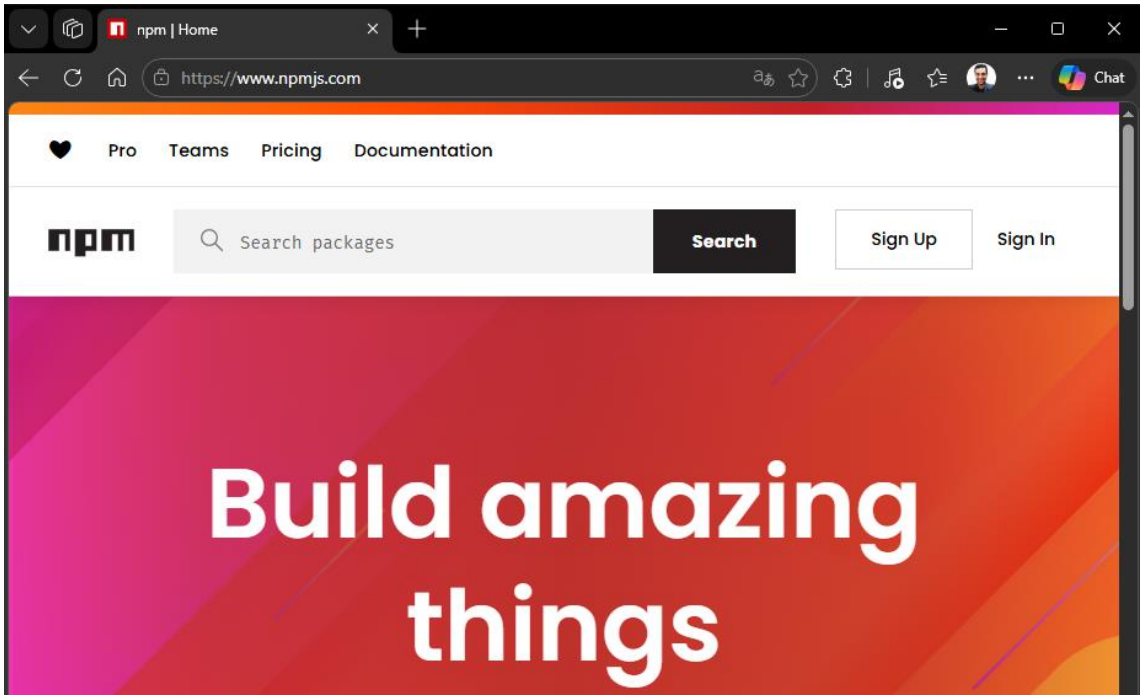


Agora abra o navegador e acesse **http://localhost:3000**. Você verá a mensagem enviada pelo seu servidor.

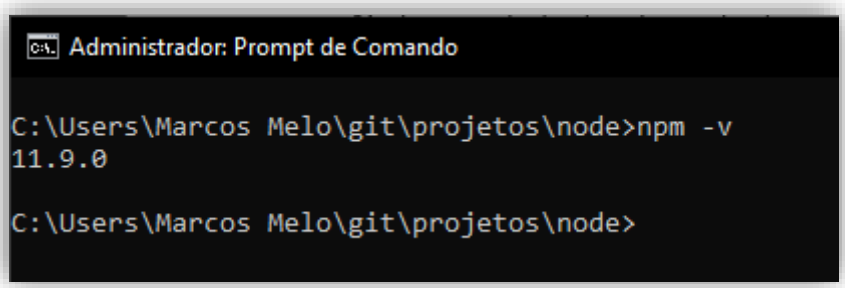


4. O Ecossistema NPM

Ao instalar o Node, você ganha o **NPM (Node Package Manager)**. Ele é a maior biblioteca de códigos do mundo. Precisa enviar e-mail? Existe um pacote. Precisa de um sistema de login? Existe um pacote. Isso acelera absurdamente o desenvolvimento.



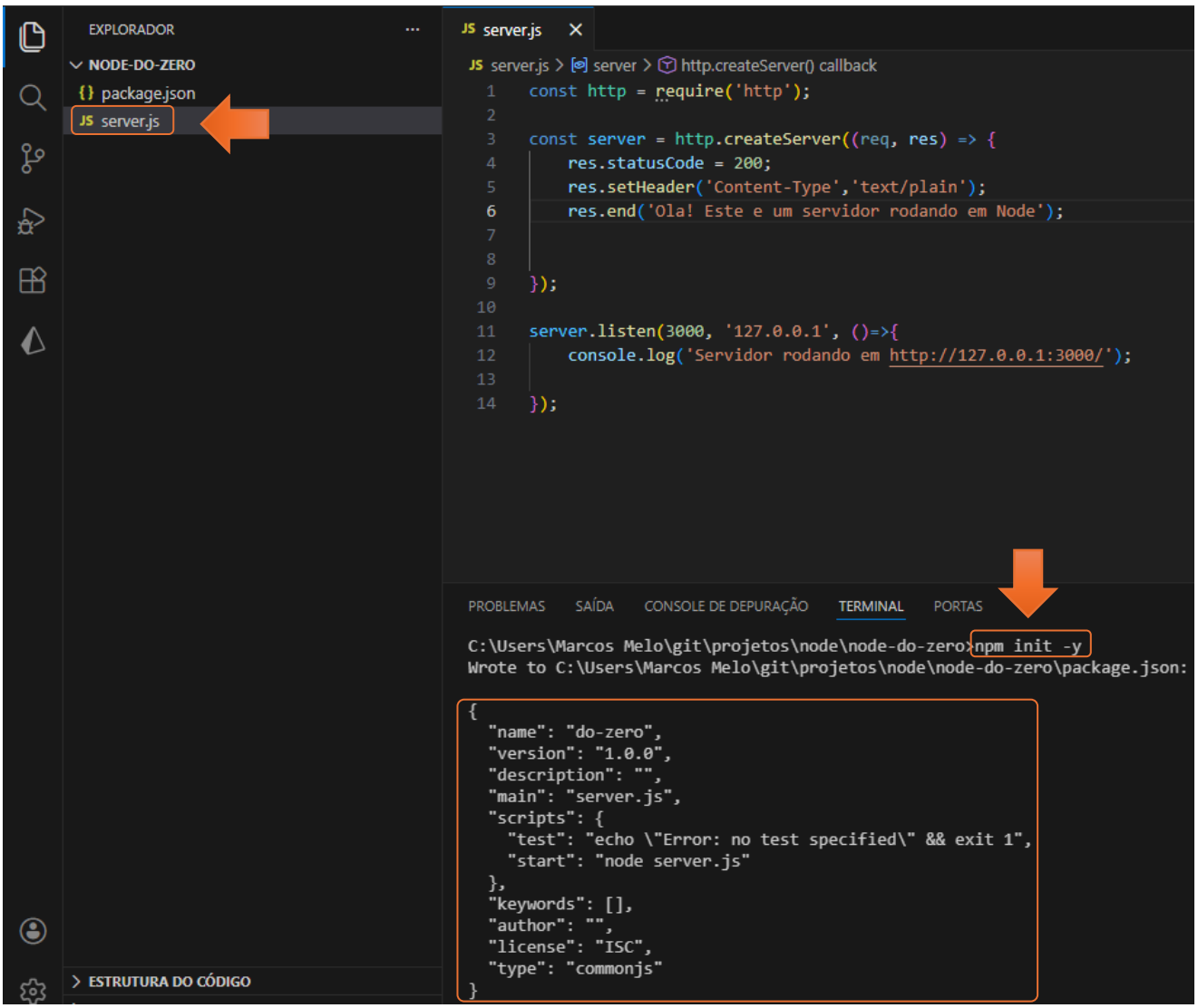
O **npm** já vem instalado junto com o node, e para verificar a versão do NPM digite no terminal: **npm -v**, se o **npm** estiver instalado deverá aparecer a versão instalada.



Usando o NPM

Para importar os vários pacotes de bibliotecas Javascript em um projeto servidor, api em node, é preciso inicializar o **npm** dentro da pasta do projeto.

Para iniciar o **npm** dentro da pasta do projeto digite no terminal estando dentro da pasta do projeto, o comendo: **npm init -y,** isso criará o arquivo **package.json,** este arquivo irá controlar a gestão de pacotes instalados no projeto.



O arquivo package.json é o "coração" de qualquer projeto Node.js. Ele funciona como um **manifesto**, contendo os metadados necessários para gerenciar as dependências, scripts e informações gerais do projeto.

Aqui estão os pontos principais para entender sua função:

---

1. Metadados do Projeto

Ele define o que o seu projeto é. Sem esse arquivo, o Node e o gerenciador de pacotes (NPM ou Yarn) não sabem como lidar com o código.

- **Nome e Versão:** Essenciais se você pretende publicar o pacote.
- **Main:** Define o ponto de entrada da aplicação (geralmente index.js).

2. Gerenciamento de Dependências

Esta é a parte mais importante. O arquivo lista todas as bibliotecas externas que seu projeto precisa para rodar:

- **dependencies:** Pacotes necessários para a aplicação funcionar em produção (ex: Express, React).
- **devDependencies:** Ferramentas usadas apenas durante o desenvolvimento (ex: Jest para testes, Nodemon).

Por que isso é útil? Em vez de enviar a pasta node_modules (que é enorme) para o GitHub, você envia apenas o package.json. Quando outra pessoa baixar o projeto, ela só precisa rodar npm install para que todas as bibliotecas sejam baixadas automaticamente.

3. Scripts de Automação

O campo "scripts" permite criar atalhos para comandos complexos do terminal.

- Exemplo: Em vez de digitar node src/server.js, você pode configurar um script "start": "node src/server.js" e rodar apenas npm start.

Estrutura Visual do Arquivo

JSON

```json
{
  "name": "meu-projeto",
  "version": "1.0.0",
  "description": "Uma breve explicação",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

Dica Extra: Você notará que, ao instalar um pacote, também surge o arquivo package-lock.json. Ele serve para registrar as versões exatas de cada sub-dependência, garantindo que o projeto rode da mesma forma em qualquer máquina.

Definindo o sistema de modulos

Essa é uma das dúvidas mais comuns hoje em dia, pois o ecossistema Node.js está em plena transição. Essa chave no package.json define como o Node deve interpretar os seus arquivos .js.

Basicamente, ela decide qual **sistema de módulos** será o padrão do seu projeto.

1. "type": "commonjs" (O Padrão Antigo)

Se você não colocar nada no package.json, o Node assume que o projeto é CommonJS. É o sistema que o Node usa desde o seu nascimento.

- **Sintaxe:** Usa require() para importar e module.exports para exportar.
- **Carregamento:** É **síncrono**. Os módulos são carregados um por um, na ordem em que aparecem.
- **Comportamento:** Permite importar arquivos sem extensão (ex: require('./auth')) e oferece variáveis globais automáticas como __dirname e __filename.

```JavaScript
// Exemplo CommonJS
const express = require('express');
module.exports = { app };
```

2. "type": "module" (O Padrão Moderno / ESM)

Ao definir isso, você diz ao Node para usar **ECMAScript Modules (ESM)**, que é o padrão oficial do JavaScript (o mesmo usado nos navegadores e em frameworks como React/Vue).

- **Sintaxe:** Usa import e export.
- **Carregamento:** É **assíncrono**, o que é mais eficiente para o desempenho em aplicações modernas.
- **Rigor:** Exige que você coloque a extensão do arquivo na importação (ex: import { auth } from './auth.js').
- **Limitação:** Não possui __dirname por padrão (você precisa usar import.meta.url para obter o caminho do arquivo).

```JavaScript
// Exemplo ESM
import express from 'express';
export const app = express();
```

Mudando o tipo **commonjs** para **module**

Ao fazer esta mudança seu projeto vai parar de funcionar devido a mudança no tipo de importação, então vamos mudar no arquivo as linhas a seguir;

Após esta mudança o arquivo voltara a funcionar normalmente.

Resumo Visual

Conceito	**Analogia**
JavaScript	O idioma falado.
Motor V8	O tradutor ultra-rápido.
Node.js	A casa onde o idioma é falado.
NPM	A caixa de ferramentas pronta para uso.

Por que aprender Node.js hoje?

- **Linguagem Única:** Você usa JavaScript no Front-end (com React/Vue) e no Back-end. É o famoso "Full Stack".
- **Comunidade Gigante:** O **NPM** (Node Package Manager) é o maior registro de bibliotecas do mundo. Precisa de algo? Provavelmente alguém já criou um pacote para o Node.
- **Site do NPM** (Node Package Manager): https://www.npmjs.com/
- **Mercado:** Empresas como Uber, LinkedIn, PayPal e NASA utilizam Node.js em seus sistemas críticos.

Por que usar Node.js e não outra tecnologia? (Conclusão)

Para fechar a aula, é importante destacar os diferenciais competitivos no mercado:

- **Isomorfismo:** O dev usa a mesma linguagem (JS) no Front-end e no Back-end.
- **Velocidade de Escrita:** O que levaria 50 linhas em Java/C#, o Node resolve em 10.
- **Escalabilidade:** Grandes empresas (PayPal, Uber) migraram para Node para aguentar milhões de acessos simultâneos com baixo consumo de memória.

Criando script de execução

Criando o servidor utilizando Framework

Acabamos de criar um servidor nativo usando somente o Node, na verdade os projetos de servidor em node são criados utilizando frameworks.

Usar o Node.js "puro" (nativo) para criar um servidor é como construir um carro comprando peça por peça: você tem controle total, mas gasta um tempo enorme montando coisas básicas que todo carro precisa, como o chassi ou o painel.

Um framework como o **Fastify** (ou Express) já te entrega o "carro montado", permitindo que você foque apenas em dirigir (a lógica do seu negócio).

Por que usar Fastify em vez do Node Nativo?

1. Abstração e Facilidade (Menos Código)

No Node nativo, você precisa lidar manualmente com fluxos de dados (streams), analisar cabeçalhos de requisição e gerenciar rotas com if/else complexos.

- **Nativo:** Você escreve 20 linhas para ler um JSON básico.
- **Fastify:** Você escreve 1 linha; o framework já entende o JSON e o entrega pronto.

2. Gerenciamento de Rotas

O Node nativo não sabe distinguir automaticamente entre um GET /usuarios e um POST /usuarios. Você precisa criar essa lógica do zero. O Fastify oferece um sistema de roteamento intuitivo e extremamente rápido.

3. Performance e Validação

O Fastify é famoso por ser um dos frameworks mais rápidos do ecossistema.

- **JSON Schema:** Ele usa esquemas para validar os dados que chegam. Se o usuário enviar um texto onde deveria ser um número, o Fastify barra a requisição antes mesmo de processá-la, poupando recursos.
- **Serialização:** Ele transforma objetos em texto (JSON) de forma muito mais veloz que o padrão JSON.stringify().

4. Ecossistema de Plugins

Precisa conectar ao banco de dados? Autenticação JWT? CORS? No nativo, você teria que configurar cada detalhe. No Fastify, você simplesmente instala um plugin oficial (@fastify/postgres, @fastify/jwt) e ele se integra ao ciclo de vida da aplicação de forma segura.

Quando usar cada um?

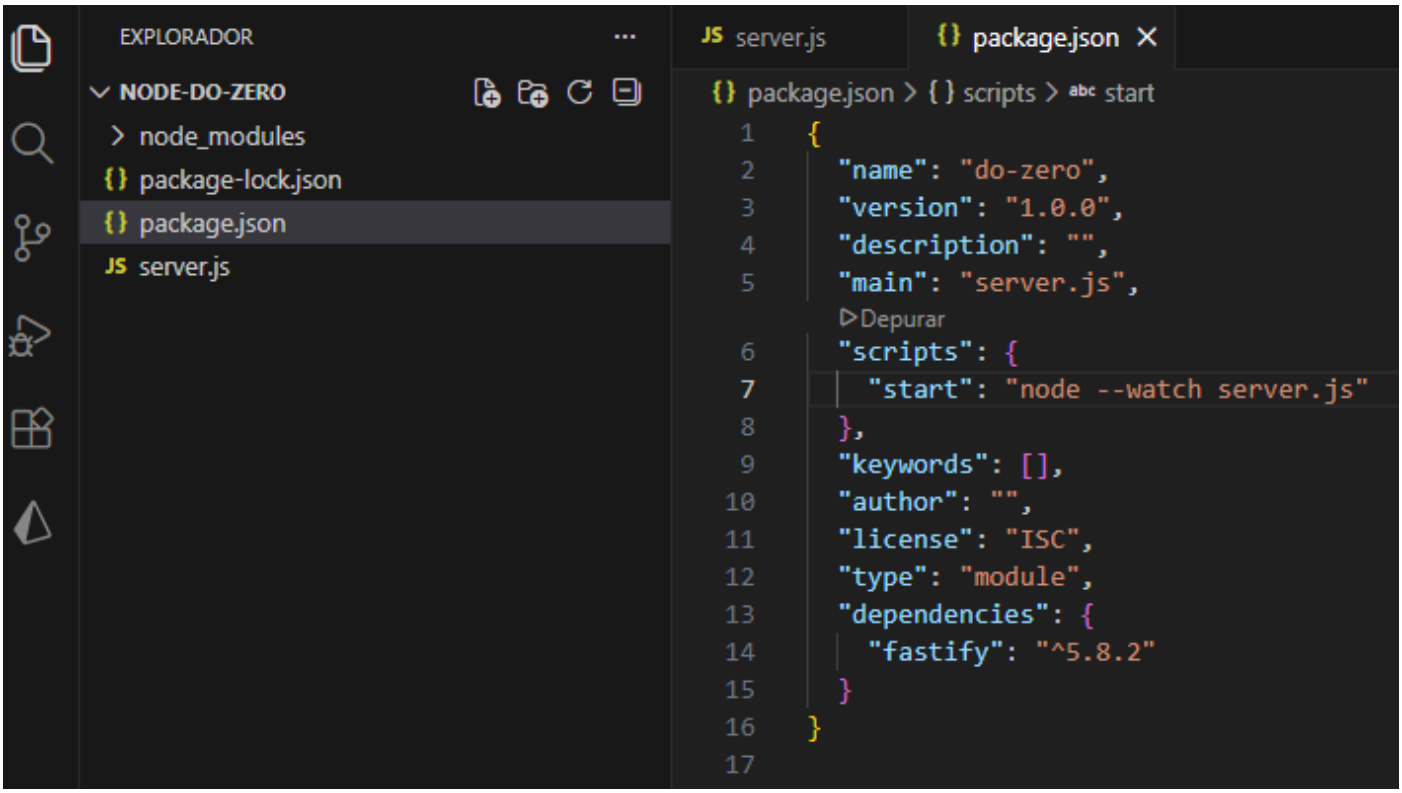
- **Node Nativo:** Use apenas para aprendizado (entender como o protocolo HTTP funciona por baixo dos panos) ou se estiver criando uma ferramenta extremamente minimalista onde cada byte de memória conta.
- **Fastify:** Use para quase tudo na vida real. É focado em **performance extrema** e baixo overhead, sendo ideal para microsserviços modernos.

Instalando o Fastify

Para instalar a dependência do Framework Fasstify digite o terminal o comando a seguir;

npm install fastify

Ao instalar o Fastify repare que foi criado a pasta **node_modules**, dentro esta pasta esta todas as dependências necessárias para o Fastify funcionar.



Criando rotas

Mais rotas

server.js

```
import { fastify} from 'fastify';

const server = fastify()

server.get('/', ()=>{
    return 'Hello World';
});
server.get('/etec', ()=>{
    return 'Ola ETEC Sumare';
});
server.get('/classe', ()=>{
    return 'Fala turma show!!';
});

server.listen({
    port:3000,
})
```

API – Interface que permite a comunicação entre diferentes sistemas.

Ex:

Paypal → API → Sistema A

→ Sistema B

REST – Padrão de arquitetura

Possui 6 partes

1. Uniformidade
2. Desacoplação
3. Statess
4. Cache
5. Arquivo de camadas
6. Code on demand

Benefícios

Padronização

Facilita;

- Uso
- Consumo
- Manutenção