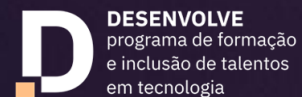



KORU





NPM, PACOTES E MÓDULOS

O QUE VAMOS APRENDER HOJE?

- **NPM** (Node Package Manager)
- **package.json** e arquivos de um Projeto Node
- Comandos Essenciais (npm install, npm run, npx, etc.)
- Versionamento Semântico (**SemVer**)
- Organizando Código com **Módulos** Javascript
 - **ES Modules** (import e export)
 - Configurando Node.js para ES Modules
 - (Breve menção a CommonJS)



PROJETO NODE

UM PROJETO NODE

Projeto Node

- Uma **pasta** (diretório) no seu computador.
- **Contém arquivos Javascript (.js)**, e potencialmente outros tipos de arquivos (dados, configuração, etc.).
- Contém um arquivo **package.json** e uma pasta **node_modules**

Projetos Node podem ser:

- Backends de aplicações web (servidores).
- Ferramentas de linha de comando (CLI).
- Scripts de automação.
- Ferramentas de build para frontend (como veremos adiante!).
- (Imagem simples: uma pasta com arquivos JS, uma seta apontando para o ícone do Node.js)





POR QUE GERENCIADORES DE PACOTES?

POR QUE GERENCIADORES DE PACOTES?

- Lembram dos projetos HTML/CSS/JS no navegador? Usávamos um **script .js**.
- E se precisarmos usar funcionalidades complexas que alguém já criou? (Ex: formatar datas, fazer requisições HTTP avançadas).
 - Copiar e colar código não escala.

Gerenciadores de Pacotes (como NPM):

- São como uma "loja de aplicativos" ou "biblioteca" para código Javascript. App/Play Store do JavaScript
- Permitem instalar, atualizar e gerenciar bibliotecas (pacotes) criadas por outras pessoas.
- Automatizam o download e a organização dessas "dependências" nos nossos projetos.

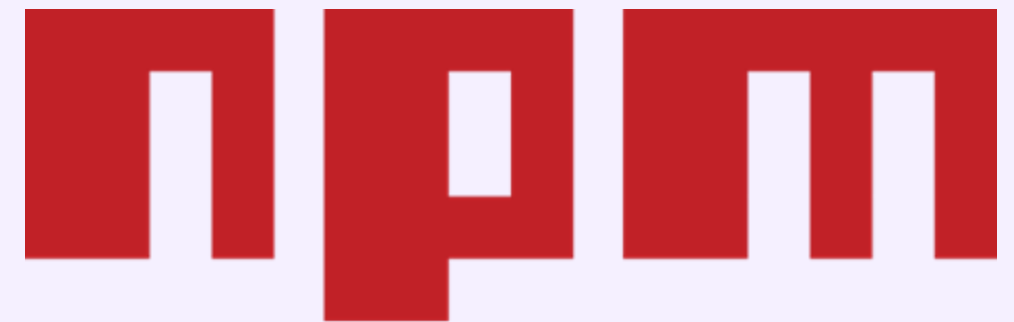


NPM

NPM

NPM = Node Package Manager

- É o **gerenciador de pacotes padrão para o Node.js** (e o *maior ecossistema de código open-source do mundo!*).
- Vem instalado junto com o Node.js.
- Permite interagir com o registro público de pacotes em npmjs.com.



A group of diverse, smiling people, overlaid with a purple gradient and the text PACKAGE.JSON.

PACKAGE.JSON

PACKAGE.JSON

- É o manifesto do nosso projeto Node.js.
- Contém informações importantes como:
 - **name**: Nome do projeto.
 - **version**: Versão atual (usando SemVer, veremos já!).
 - **description**: Descrição do projeto.
 - **main**: Arquivo de entrada principal (geralmente index.js ou src/index.js).
 - **scripts**: Atalhos para executar tarefas comuns (ex: rodar o projeto, executar testes, compilar código).
 - **dependencies**: Pacotes que seu projeto precisa para rodar em produção.
 - **devDependencies**: Pacotes que seu projeto precisa apenas durante o desenvolvimento (ex: ferramentas de build, testes, linters).
- Criamos ele com npm init.

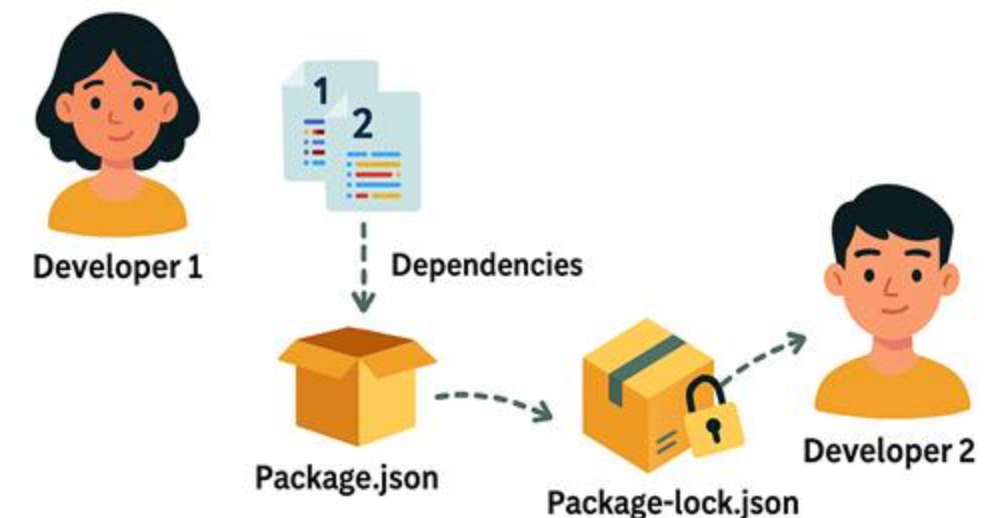
```
{
  "name": "nome-do-projeto",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev --turbo",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "@ai-sdk/openai": "^1.3.20",
    "@ai-sdk/react": "^1.2.9",
    "ai": "^4.3.10",
    "gray-matter": "^4.0.3",
    "next": "15.3.1",
    "openai": "^4.96.0",
    "playwright": "^1.52.0",
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "react-markdown": "^10.1.0",
    "slugify": "^1.6.6"
  },
  "devDependencies": {
    "@eslint/eslintrc": "^3",
    "@tailwindcss/postcss": "^4",
    "@types/node": "^20",
    "@types/react": "^19",
    "@types/react-dom": "^19",
    "eslint": "^9",
    "eslint-config-next": "15.3.1",
    "tailwindcss": "^4",
    "typescript": "^5"
  }
}
```



PACKAGE-LOCK.JSON

PACKAGE-LOCK.JSON

- Garantia de Consistência!
- Registra as **versões exatas** de cada pacote instalado e suas sub-dependências.
- Assegura que, se duas pessoas rodarem npm install no mesmo projeto, elas terão a mesma estrutura de node_modules.
- Este arquivo SIM **deve ser commitado** para o Git!

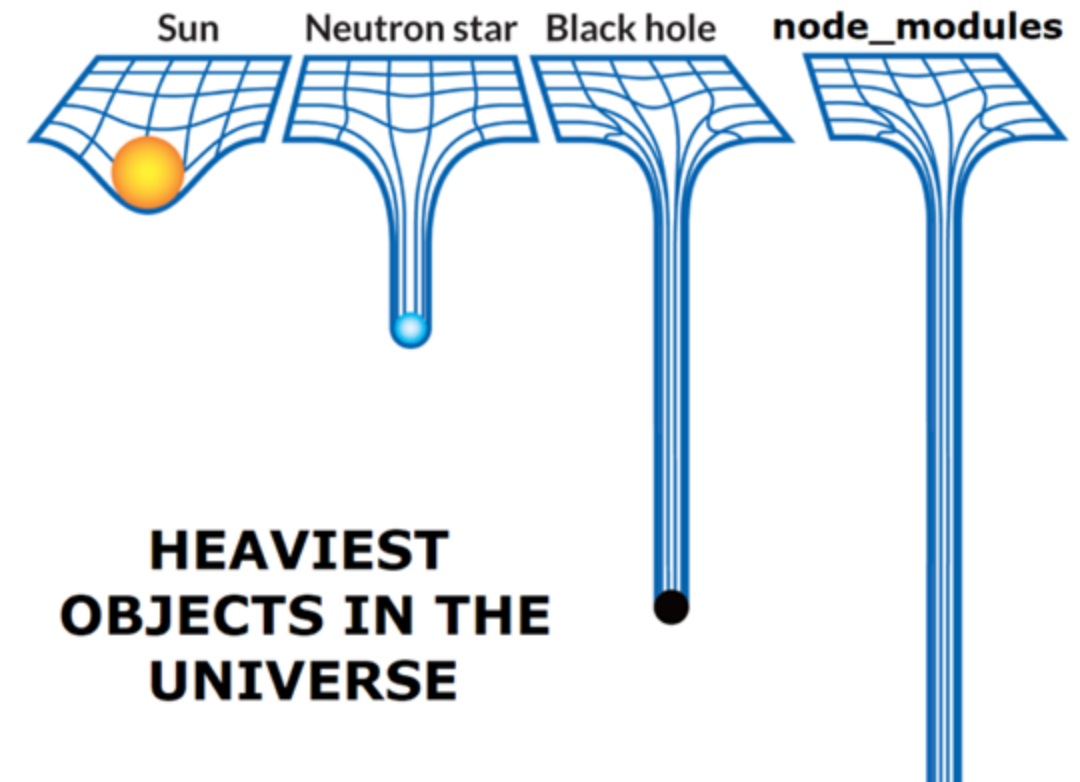


A group of diverse, smiling people, overlaid with a purple gradient and the text 'NODE_MODULES'. The image features a group of six people of various ethnicities and ages, all smiling and looking towards the camera. The image is overlaid with a semi-transparent purple gradient. The text 'NODE_MODULES' is written in a bold, white, sans-serif font across the center of the image.

NODE_MODULES

NODE_MODULES

- Quando declaramos as dependências e instalamos pacotes com **npm install**, para onde eles vão?
- Para a pasta **node_modules** na raiz do projeto.
- Contém o **código de todos os pacotes** (e as dependências deles!).
- Por isso ela fica GIGANTE! 🦉
- Nunca suba a pasta node_modules para o Git! Use o **.gitignore**.
- Basta rodar **npm install** no projeto para baixar tudo que está listado no **package.json**.



A group of diverse, smiling people, overlaid with a purple gradient. The text "COMANDOS DO NPM" is centered over the image in a bold, white, sans-serif font.

COMANDOS DO NPM

COMANDOS ESSENCIAIS

Inicializando um projeto

```
npm init # Cria package.json (faz perguntas)
npm init -y # Cria package.json com valores padrão
```

Instalando Pacotes

```
npm install <package-name> # Instala como dependency
npm i <package-name> # Atalho para install

npm install <package-name> --save-dev # Instala como devDependency
npm i -D <package-name> # Atalho para install --save-dev

npm install # Instala TODAS as dependências do package.json
```

COMANDOS ESSENCIAIS

Removendo Pacotes

```
npm uninstall <package-name>  
npm remove <package-name> # Atalho  
npm un <package-name> # Atalho
```

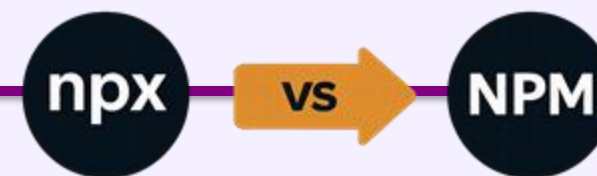
Executando Scripts

```
npm run <script-name> # Ex: npm run dev  
npm start # Atalho para 'npm run start'  
npm test # Atalho para 'npm run test'
```



NPX

NPX



- O **NPX** (Node Package Execute) é uma ferramenta incluída no NPM
- Permite executar um binário (programa cli) de um pacote NPM.
- VANTAGEM: Executa sem precisar instalar o pacote globalmente (npm install -g).

NPX vs NPM

- **npm install** <pacote>: Instala o pacote no node_modules do projeto.
- **npm install -g** <pacote>: Instala o pacote globalmente no seu sistema.
- **npm run <script>**: Executa um script definido no package.json (que geralmente usa comandos de pacotes locais).
- **npx <comando>**: Executa o comando binário de um pacote (localmente do projeto ou baixando temporariamente)

NPX

AGORA, abra seu terminal e digite o seguinte comando:

```
npmx cowsay "Olá amiguinhos!"
```



PRÁTICA 01. EXPLORANDO UM PROJETO NODE

PRÁTICA



Objetivo: Criar um pequeno script em Node.js que usa um pacote externo do NPM (crypto-js) para criptografar e descriptografar uma mensagem definida no próprio código.

Foco: Praticar npm init, npm install, entender package.json e node_modules, e rodar um script Node.js.v

PRÁTICA



Dicas:

- Use o pacote **crypto-js**
- Use o algoritmo **AES** para criptografar e descriptografar.
- Você deverá definir a **mensagem** a ser criptografada e a **chave secreta** da criptografia.

✓ Está tudo bem se você não conseguir fazer tudo! A ideia aqui é começar um projeto node!



VERSIONAMENTO SEMÂNTICO

SEMVER

Pelo número da versão conseguimos saber se uma nova versão de um pacote vai quebrar nosso código!

SemVer: Um padrão (**major.minor.patch**) para versionamento de software:

MAJOR.MINOR.PATCH

MAJOR: Mudanças incompatíveis (quebram código antigo). Ex: 1.0.0 -> 2.0.0

MINOR: Novas funcionalidades compatíveis com versões anteriores. Ex: 1.0.0 -> 1.1.0

PATCH: Correções de bugs compatíveis. Ex: 1.0.0 -> 1.0.1

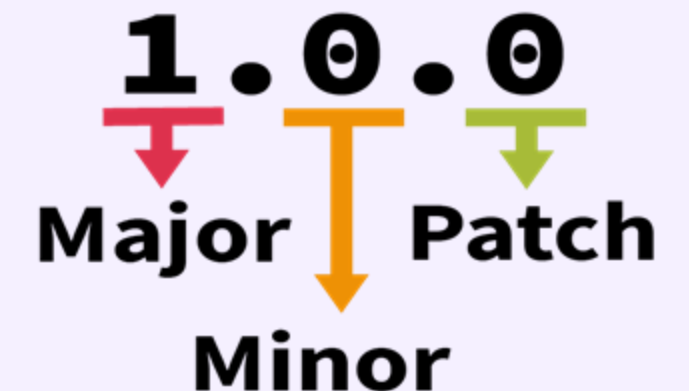
Símbolos no package.json:

^1.2.3: Permite atualizações **MINOR** e **PATCH** (até a próxima MAJOR - v2.0.0). Recomendado!

~1.2.3: Permite apenas atualizações **PATCH** (até a próxima MINOR - v1.3.0). Mais restritivo.

*****: Qualquer versão (evitar!).

1.2.3: Versão exata.





MÓDULOS NO JS

POR QUE MODULARIZAR?



- Imagine ter todo o código da sua aplicação em um único arquivo script.js. 🤪
- Fica difícil de:
 - Ler e entender.
 - Reutilizar partes do código.
 - Evitar conflitos de nomes de variáveis/funções.
 - Manter e dar manutenção.

Solução: Dividir o código em arquivos menores (módulos), cada um com uma responsabilidade específica.

COMMONJS

Historicamente, Javascript não tinha um sistema de módulos nativo.

Hoje o ecossistema está fugindo do CommonJS e migrando para o ESM.

- **CommonJS:**
 - Padrão popularizado pelo Node.js.
 - Sintaxe:
 - **require**('módulo') para importar,
 - **module.exports** = valor para exportar

ES MODULES (PADRÃO MODERNO)

ES Modules (ESM):

- Sistema de módulos padrão do Javascript definido pela especificação ECMAScript.
- Padrão para o frontend (navegadores) e o padrão recomendado para o backend (Node.js moderno).
- Sintaxe mais limpa e declarativa.
- Permite análise estática (otimizações por bundlers como Webpack/Vite).

ES MODULES (PADRÃO MODERNO)

Exportando Módulos

- Ao exportar tornamos variáveis, funções e classes disponíveis para outros módulos

```
// utils.js
export const PI = 3.14159;

export function sum(a, b) {
  return a + b;
}

export const subtract = (a, b) => a - b;

// Outra forma de exportar tudo no final
// export { PI, soma, subtrai };
```

ES MODULES (PADRÃO MODERNO)

Exportando Módulos com export default

- Exportação Padrão (Default): Exporta um único item principal do módulo.
- Você só pode ter um export default por arquivo.
- Ao importar, você pode dar qualquer nome a esse item.

```
// calculos.js
const multiply = (a, b) => a * b;
const divide = (a, b) => a / b;

export default multiply; // Exporta APENAS a função multiplica como padrão
// export { divide }; // Pode ter exportações nomeadas junto
```


ES MODULES (PADRÃO MODERNO)

Importando Módulos

- Acessar variáveis, funções, classes, etc., exportadas por outros módulos.
- **Importação Nomeada e Importação Padrão**

```
import { PI, sum, subtract } from './utils.js'; // Caminho relativo
// ou de um pacote instalado:
// import { camelCase } from 'lodash';

// Importa o default export
import multiply from './calculos.js'; // Importa o default export
// ou:
import multiply from './calculos.js'; // Nome comum é usar o nome original
```

ES MODULES (PADRÃO MODERNO)

ESM no Node.js

- Por padrão, versões mais antigas do Node.js usavam CommonJS (require).
- Para dizer ao Node.js que seu projeto usa ES Modules (import/export), adicione "type": "module" ao seu package.json.

```
{  
  "name": "meu-projeto-js",  
  "version": "1.0.0",  
  "type": "module", // <-- Adicione esta linha!  
  "main": "index.js"  
}
```



PRÁTICA 02. **MÓDULOS NO JS**

PRÁTICA

- Vamos aplicar tudo que vimos!
- Vamos criar um projeto Node.js do zero.
- Inicializar com npm init.
- Instalar um pacote externo (lodash).
- Configurar type: "module".
- Usar o pacote importando-o (import).
- Criar nossos próprios módulos (utils.js, calculos.js).
- Exportar itens (export, export default).
- Importar nossos módulos no arquivo principal (index.js).
- Executar scripts customizados (npm run).
- Usar npx.

A group of diverse young people, including men and women of various ethnicities, are smiling and waving. The image is overlaid with a purple gradient. The word "RESUMO" is written in large, white, bold, sans-serif capital letters on the right side of the image.

RESUMO

RESUMO

- **NPM:** Gerencia pacotes externos (node_modules).
- **package.json:** Manifesto do projeto, lista dependências, scripts.
- **package-lock.json:** Garante versões exatas e consistentes das dependências.
- **SemVer:** MAJOR.MINOR.PATCH indica compatibilidade das versões.
- **Módulos:** Organizam código em arquivos menores.
- **ES Modules (import/export):** Padrão moderno para reuso e organização.
- Use "**type**": "**module**" no package.json para usar ES Modules no Node.js.



Vamos
avaliar o
encontro?

KORU

