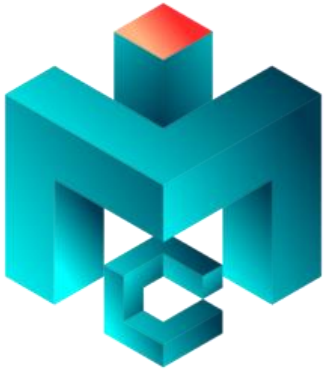


Aula – 10
Introdução ao Node

Disciplina: XDES03 – Programação Web

Prof: Phyllipe Lima Francisco
phyllipe@unifei.edu.br

Universidade Federal de Itajubá – UNIFEI
IMC – Instituto de Matemática e Computação

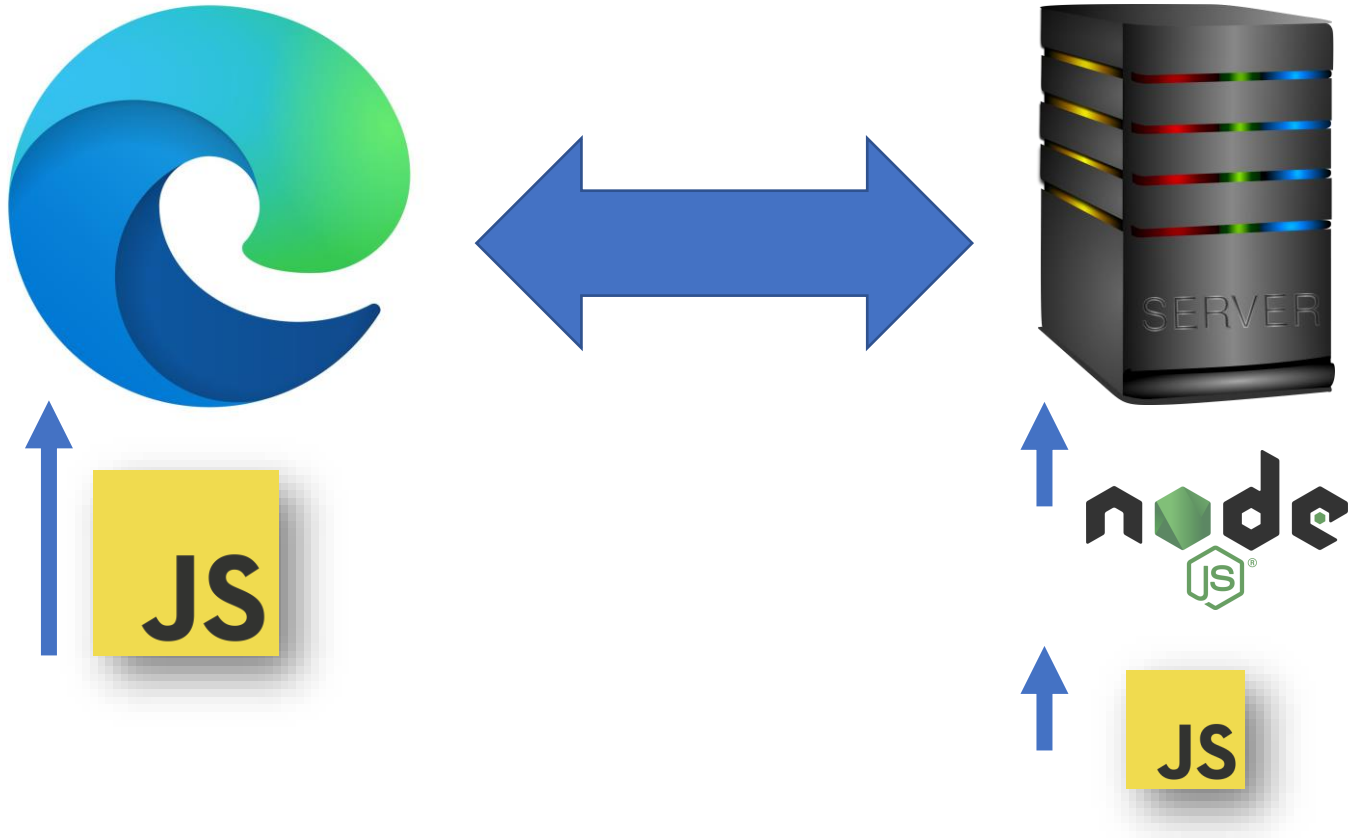
Agenda

- Introdução
- Instalação
- Execução de Projetos Node
- Sistema de Arquivos
- NPM
  - Importar pacotes

Introdução ao NodeJS.

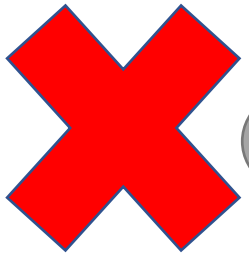
- NodeJS é um ambiente de execução de código JavaScript
- Com NodeJS é possível executar código JavaScript fora do navegador.
- Com NodeJS podemos criar código do lado do servidor utilizando JavaScript

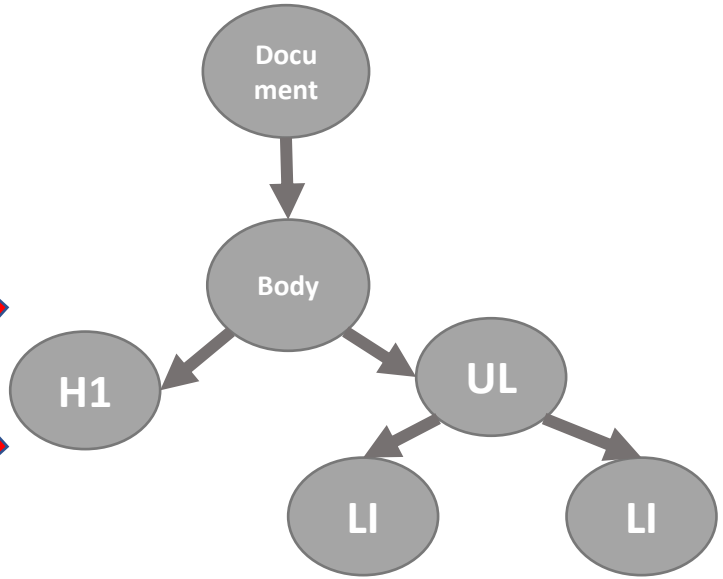
Código JavaScript no servidor.



Qual a diferença?

- Mesmo sendo JavaScript, existem diferenças entre o código de servidor e o código que executa em um navegador.
- Exemplo: No servidor, não temos o DOM.





Qual a vantagem?

- Uma das vantagens mais interessantes é permitir utilizar a linguagem JavaScript em código sendo executado do lado do servidor.
- Existe toda uma cultura relacionado ao uso de qualquer linguagem de programação. E permitir utilizar a mesma linguagem em diferentes locais pode ser muito vantajoso para equipes.

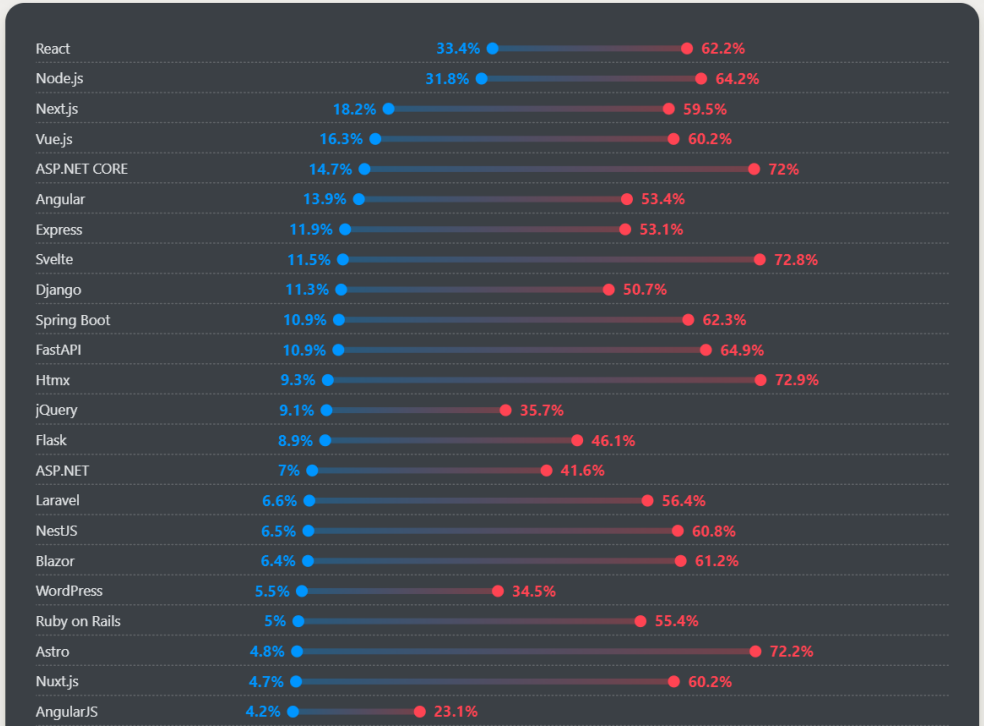
Popularidade do NodeJS

StackOverflow 2024

Web frameworks and technologies

73% of developers that used it want to keep working with Svelte. Fun fact: Our team at Stack Overflow used Svelte for the first time in building our 2024 Developer Survey results site. We could go on and on about Svelte, listen to us do just that in a interview with one of our own.

Which **web frameworks and web technologies** have you done extensive development work in over the past year, and which do you want to work in over the next year? (If you both worked with the framework and want to continue to do so, please check both boxes in that row.)

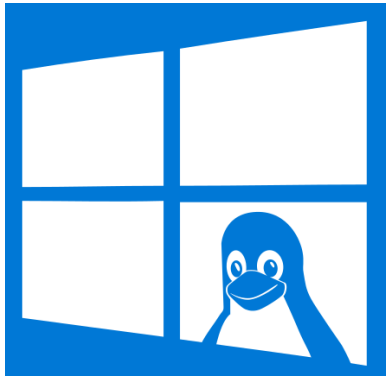


O que se constrói com NodeJS?

- Servidores Web (Especialmente com ExpressJS)
- Frontend, devido aos módulos tais como:
  - Converter JSX em JS.
- Aplicações de Linha de Comando.
- Aplicações Desktop
  - VSCode é feito com HTML/CSS/JS

Ambiente de Instalação

- Os passos a seguir consideram que umas das seguintes máquinas estão sendo utilizadas:
  - Mac OSX
  - Windows com WSL
  - Linux



Verificando se NodeJS está instalado

- No terminal digite node :~$ node
  - Se obtiver uma resposta como:
    - Welcome to Node.js v21.7.1.
    - Está instalado

```
Welcome to Node.js v21.7.1.
Type ".help" for more information.
>
```

Verificando se NodeJS está instalado

- No terminal digite node `:~$ node`
  - Se a resposta for algo como: *command not found*
  - O NodeJS não está instalado e precisamos iniciar o processo de instalação

Instalando o NVM (Node Version Manager)

- A forma recomendada de instalar o NodeJS é através de um gerenciador.
- O que vamos utilizar é o NVM (Node Version Manager)
- Acesse a página: https://github.com/nvm-sh/nvm
- Observe no README as instruções na subseção “Instaling and Updating”
- https://github.com/nvm-sh/nvm#installing-and-updating

Executando scripts de instalação

- É possível utilizar tanto um Script cURL ou Wget
- Basta copiar o código do README, colar no terminal e executar pressionando *enter*.
- Exemplo com Wget:
  - wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
  - Execute o comando acima no terminal

E se minha máquina não possuir Wget?

- Utilize o cURL ou instale o Wget:
  - Windos (WSL) e Linux: apt-get install wget
  - Mac OSX: brew install wget

Verificando que NVM foi instalado

- :~$ command -v nvm
- A saída devera sér
- :~$ nvm

Instalando NodeJS com NVM

- Para instalar a versão mais recente
  - :~$ nvm install node
- No exemplo a mais recente é a v20.2.0

```
Downloading and installing node v22.1.0...
Downloading https://nodejs.org/dist/v22.1.0/node-v22.1.0-linux-x64.tar.xz...
######################################################################
################################################## 100.0%Computing checksum
with sha256sum
Checksums matched!
Now using node v22.1.0 (npm v10.7.0)
```

Verificando que NodeJS foi instalado

- :~$ node -v
- O terminal deverá responder com a versão que acabou de ser instalada (v22.1.0 no exemplo)
- Definindo a versão padrão
  - :~$ nvm alias default 22.1.0

Outros usos do NVM

- Instalando versão específica
  - :~$ nvm install 14.7.0 (vai instalar a 14.7.0)
- Listando versões instaladas
  - :~$ nvm ls

```
        v18.20.2
         v19.9.0
         v21.7.1
->       v22.1.0
default -> node (-> v22.1.0)
```

- Trocando de versão de NodeJS
  - :~$ nvm use <versão desejada>

Resumindo Node e NVM

- NodeJS é o ambiente de execução de código JavaScript.
- NVM é o gerenciar de versões do NodeJS.
- O NodeJS está em constante evolução e existem projetos que executam em diferentes versões do NodeJS.
- Gerenciar as versões manualmente seria caótico.
- Além disso o NVM facilita escolher uma versão anterior devido a problemas de compatibilidade.

Executando código Node

- Com NodeJS é bem simples executar um código JS
- Basta executar o comando “node” e passar o caminho de onde se encontra o arquivo “.js”
- :aula-node$ node exemplo1.js

```
console.log('Meu Primeiro Script NodeJS');
```

```
Meu Primeiro Script NodeJS
```

Executando código Node

- Se houver erro o próprio interpretador irá informar.
- Exemplo tentando acessar o DOM (exemplo2)

```
console.log('Não tem DOM no servidor!!!!');
const btn = document.querySelector('button');
```

```
const btn = document.querySelector('button');
            ^

ReferenceError: document is not defined
    at Object.<anonymous> (/mnt/c/Users/phyll/Desktop/aula-node/scriptComErro.js:
2:13)
```

Process e Argv

- "process" é um objeto global que permite controle e manipulação do processo atual onde o NodeJS está sendo executado.
- Argv (argumentos)
  - Argumentos que podem ser passados para o processo NodeJs em execução.

Exemplo 3 – Usando Process

```
const version = process.version;
const cwd = process.cwd();
const release = process.release; //retorna objeto JSON
const env = process.env; //retorna objeto JSON
/*
Quando temos um objeto como retorno podemos transformar
em JSON "string" usando a função JSON.stringify().
Se informado um terceiro parâmetro numérico, cada item ficará na sua linha
com um espaço inicial "x", onde x é o terceiro parâmetro.
*/

console.log(`Versão: ${version}`);
console.log(`Diretório de Trabalho: ${cwd}`);
console.log(`Release: ${JSON.stringify(release, null, 4)}`); //Cada item
possui 4 espaços e ocupará sua própria linha
//console.log(`Ambiente (env) ${JSON.stringify(env, null,5)}`);
```

Exemplo 4 – Usando Argv

```
//argv é passado durante a execução do script.
//O primeiro parâmetro é a versão do node
//O segundo parâmetro é o CWD (Current Working Directory)
//Os demais são passados no próprio terminal.
//Considere a execução
//node exemplo4.js 40 20
console.log(process.argv); //Será impresso quatro valores em formato de array
//Pegando apenas os valores (posição 2 em diante)
const valores = process.argv.slice(2);
for(let valor of valores)
    console.log(valor); //40 e 20
```

Exemplo 5 – Criando Calculadora com Argv

- Para praticarmos o uso de Argv e parâmetros no terminal, crie uma calculadora de acordo com os seguintes requisitos:
- Parâmetro ‘-op’ informa a operação que pode ser
  - + (soma), - (subtração), / (divisão) ou x (multiplicação)
- Parâmetro ‘-opn’ informa os operandos´.

Exemplo 5 – Criando Calculadora com Argv

- Requisitos:
  - Se não houver parâmetro “-op” o programa encerra e informa que é necessário o parâmetro “-op”
  - Se houver parâmetro “-op” mas o usuário não passou um operador (+,-,x,/), o programa encerra e informa que é necessário uma operação
  - Se não houver parâmetro “-opn”), o programa encerra e informa que é necessário o parâmetro “-opn”
  - Se houver parâmetro “-opn” mas o usuário não passou dois números na sequencia o programa encerra e informa que é necessário operandos.

Exemplo 5 – Criando Calculadora com Argv

- Os parâmetros podem ser informados em qualquer ordem.
- Exemplos:

```
:$ node exemplo5.js -opn 25 34 -op +
Resultado da operação: 25 + 34 = 59

:$ node exemplo5.js -op / -opn 67.89 56.32
Resultado da operação: 67.89 / 56.32 =
1.2054332386363635
```

Exemplo 5 – Criando Calculadora com Argv

- Exemplos: O programa se chama “exemplo5.js”

$ node exemplo5.js
Necessário informar um operador com o parâmetro –op

$ node exemplo5.js -op
Operador inválido

$ node exemplo5.js -op + -opn 2
Operandos não informados.

$ node exemplo5.js -op + -opn 2 42.67
O Resultado de 2 + 42.67 = 44.67

Exemplo 5 – Criando Calculadora com Argv

- Dicas:
  - process.argv é um array e os elementos podem ser acessados pelo operador colchetes [].
    - process.argv[3] => Exemplo do 4º elemento
  - Para verificar a existência de um elemento em um array utilize *includes('elemento');*
  - Para verificar o índice de um elemento em um array utilize indexOf('elemento');
  - Lembre-se de converter 'string' para 'number': Number(string)
  - Caso queira encerrar o programa antecipadamente utilize 'process.exit()';

Leitura e Escrita de Arquivos com Node

- Para a leitura e escrita de arquivos utilizamos o módulo ‘fs’ (file system)
- É necessário importar o módulo (require).
- Com uma referência para ‘fs’ conseguimos acessar funções como:
  - mkdirSync()//cria diretório
  - open(Sync) //abre e/ou cria arquivo
  - existsSync()//Verifica se o diretório existe

Exemplo 6 – Criando Arquivos

- Elabore um programa JS que recebe um parâmetro para ser o nome de um diretório fictício e em seguida, nesse diretório, crie os três arquivos básicos para desenvolvimento web:
  - index.html
  - styles.css
  - main.js

mkdirSync – criando diretórios

- mkdirSync('nome do diretório');
- Exemplo: mkdirSync('teste') irá criar um diretório chamado *teste*.
- Caso já exista esse diretório, será lançado um erro.
- É recomendado verificar se o diretório já existe com existsSync('nome do diretório') antes de criar.

existsSync – verificando diretórios

- existsSync('nome do diretório');
- Exemplo: existsSync('teste') irá verificar se um diretório chamado *teste* existe. Caso exista a função retorna *true*.

```
if(fs.existsSync(nomeDir))
{
    console.log('Diretório já existe')
}
else
{
    fs.mkdirSync(nomeDir);
    console.log('Diretório criado')
}
```

openSync – abrindo/criando arquivos

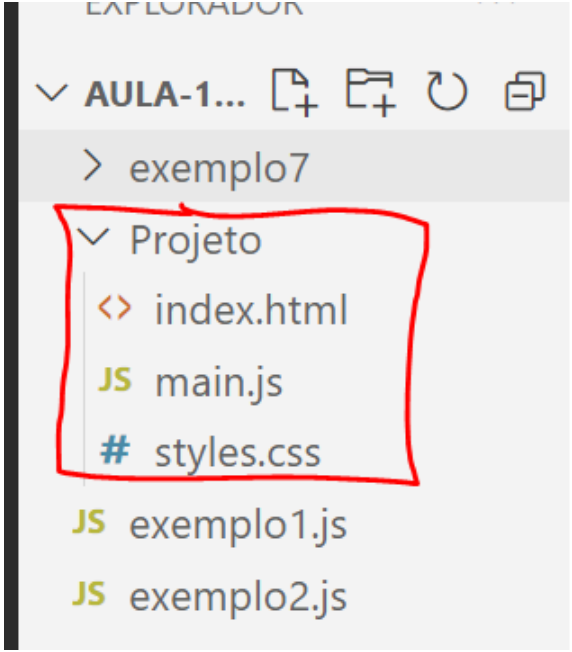
- openSync('caminho do arquivo', 'modo de acesso');
- Exemplo 1: Criar arquivo para escrita

```
//Utilize path.join para criar caminhos independente do
//sistema operacional
//Necessário fazer o require
//const path = require('path');
const nomeArquivo = path.join(nomeDir,'index.html');
//'w+' modo escrita. Caso arquivo exista, irá
sobrescrever
fs.openSync(nomeArquivo,'w+');
//
fs.writeFileSync(nomeArquivo,'<h1>Meu H1</h1>');
```

Exemplo 6 – Criando Arquivos

❑ Exemplos de execução e resultado esperado:

$ node exemplo6.js Projeto



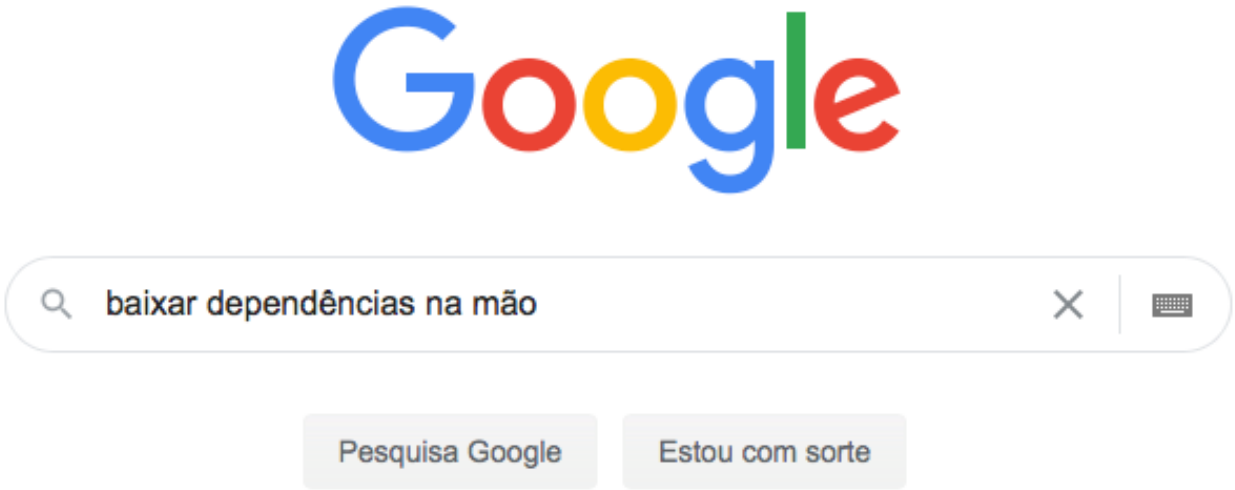
NPM – Node Package Manager

- Para entendermos NPM, precisamos primeiro compreender o que é “gerenciamento de dependências” e “automatização da build”.

Considere o desenvolvimento de um jogo !


DIABLO IV

- Você vai fazer o download dessas bibliotecas uma por uma?
- E se alguma é atualizada? Uma nova versão com melhorias. Será necessário fazer o download novamente.



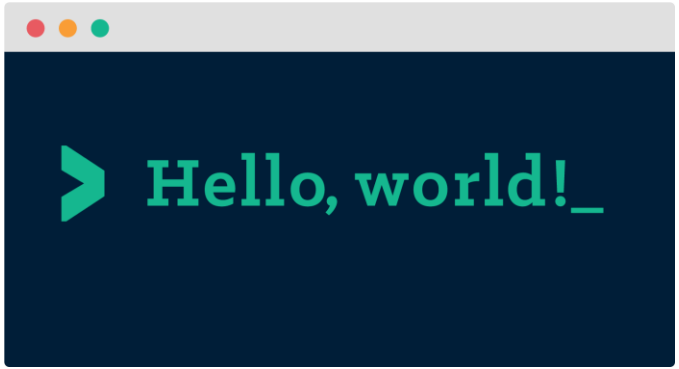
- E sua equipe? Alguém vai baixar a dependência nova e enviar por e-mail para os demais?

- Ou pior.....vai usar pen-drive?

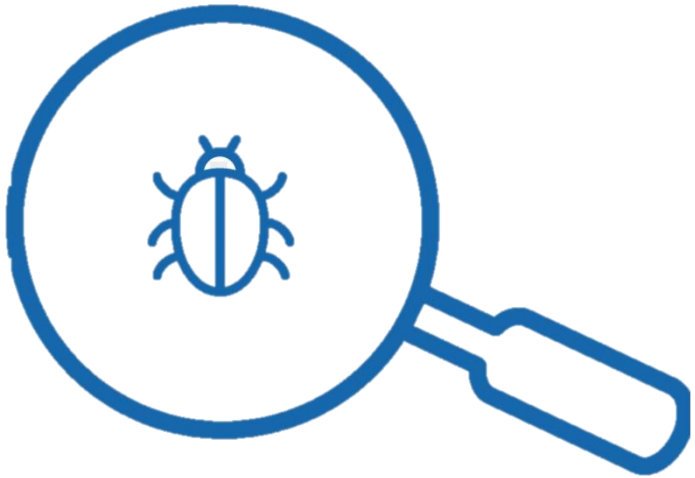
- E pra fazer o *build?* Com todas essas dependências?

- Build é o processo de geração do software como produto (compilação?). Envolve, geralmente, as etapas de: teste automatizado, compilação e empacotamento (jar , por exemplo)

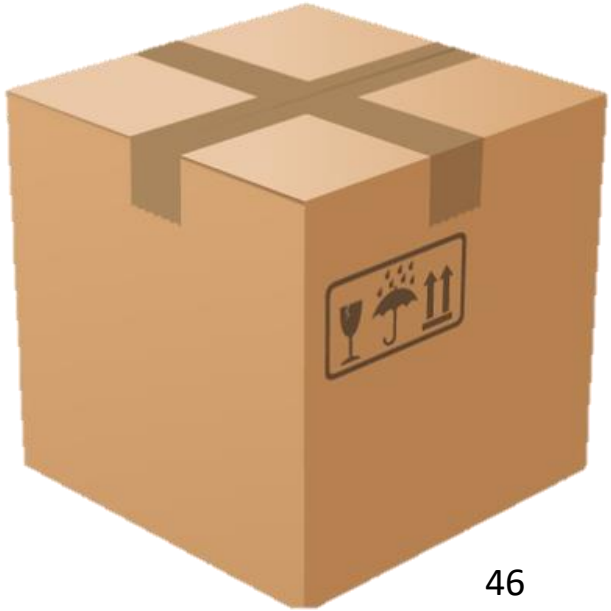
- Build – Fases para a construção do software
  - Programas pequenos, apenas compilar é o suficiente!
  - Exemplo: “Hello World”



- Software real requer mais etapas para “construir” com qualidade!
- Testes!



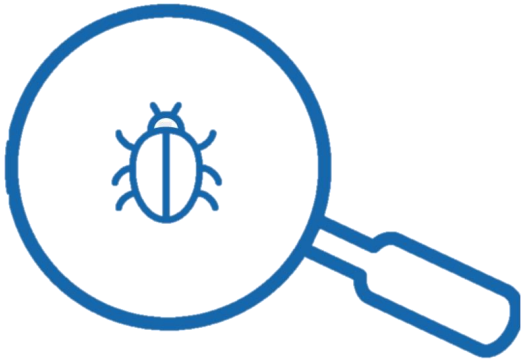
- Empacotar!
  - Gerar um pacote com tudo necessário para instalação do software.



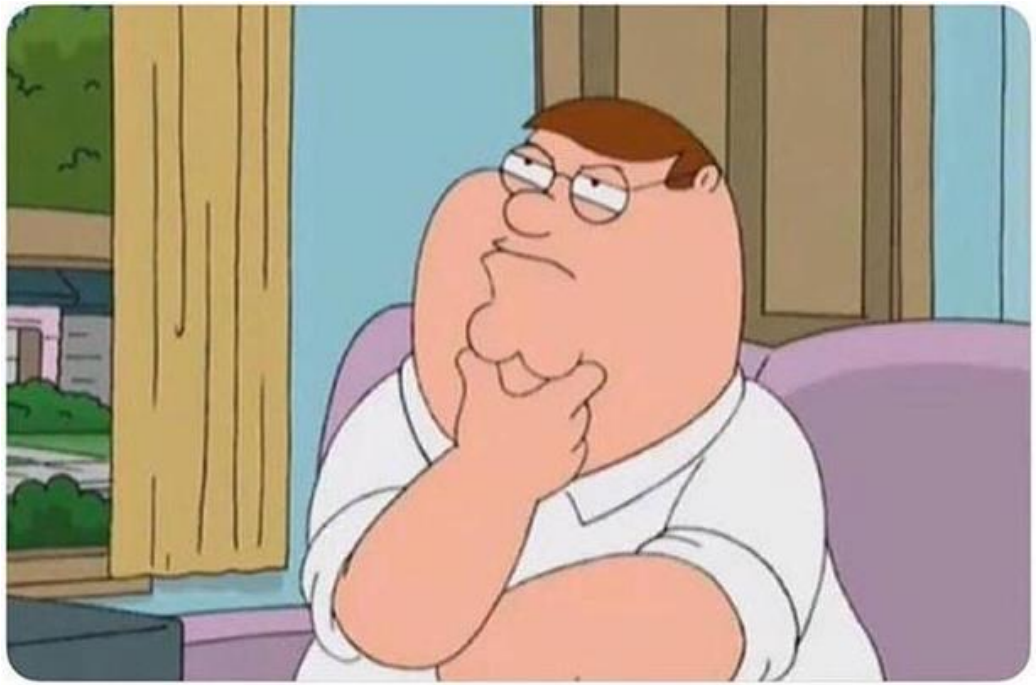

BUILDER


node
JS

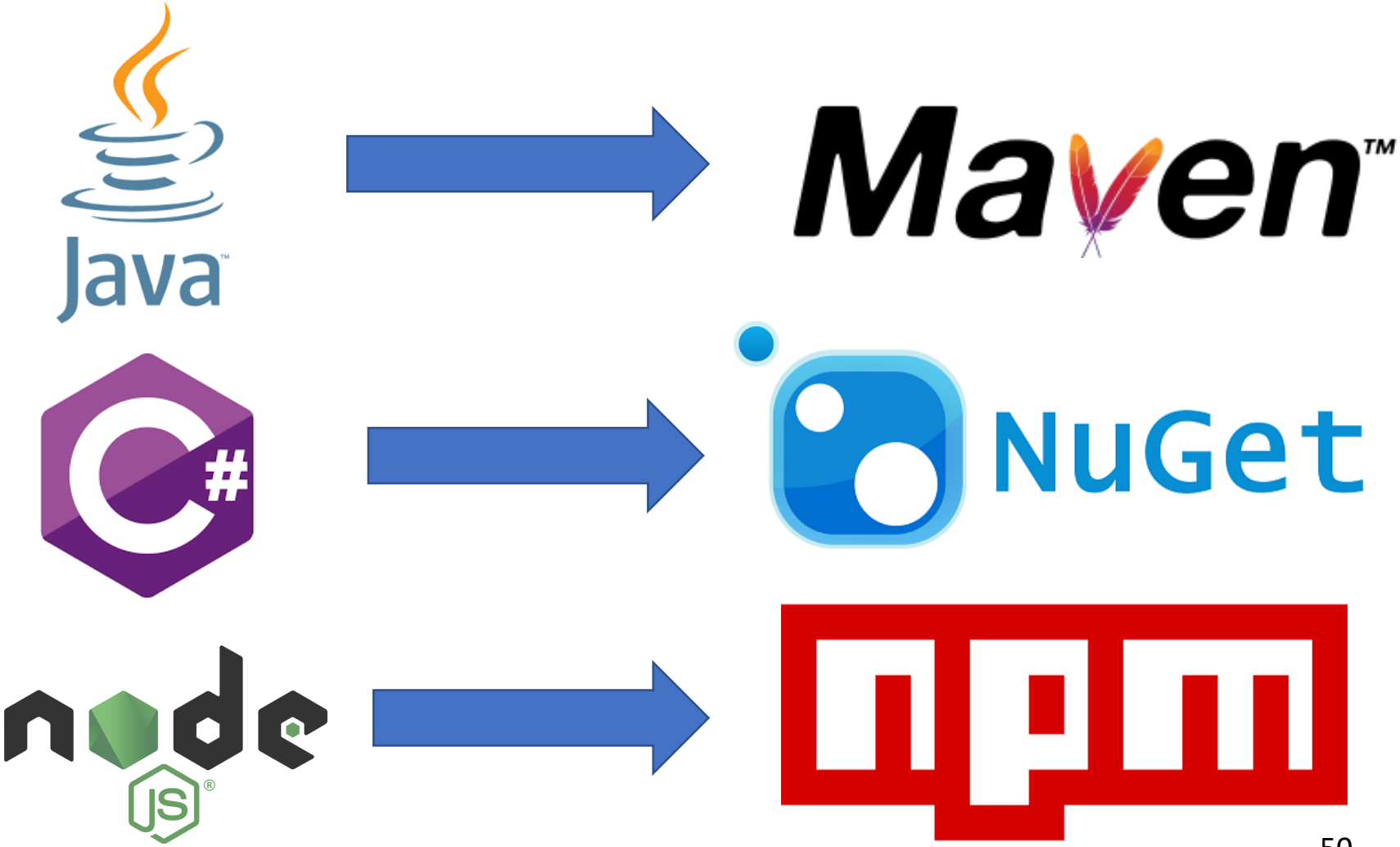

JDK



- Tem como automatizar esse processo?



- Automatização da Build


Java
Maven™
C#
NuGet
node
JS
npm

Unity's Package Manager

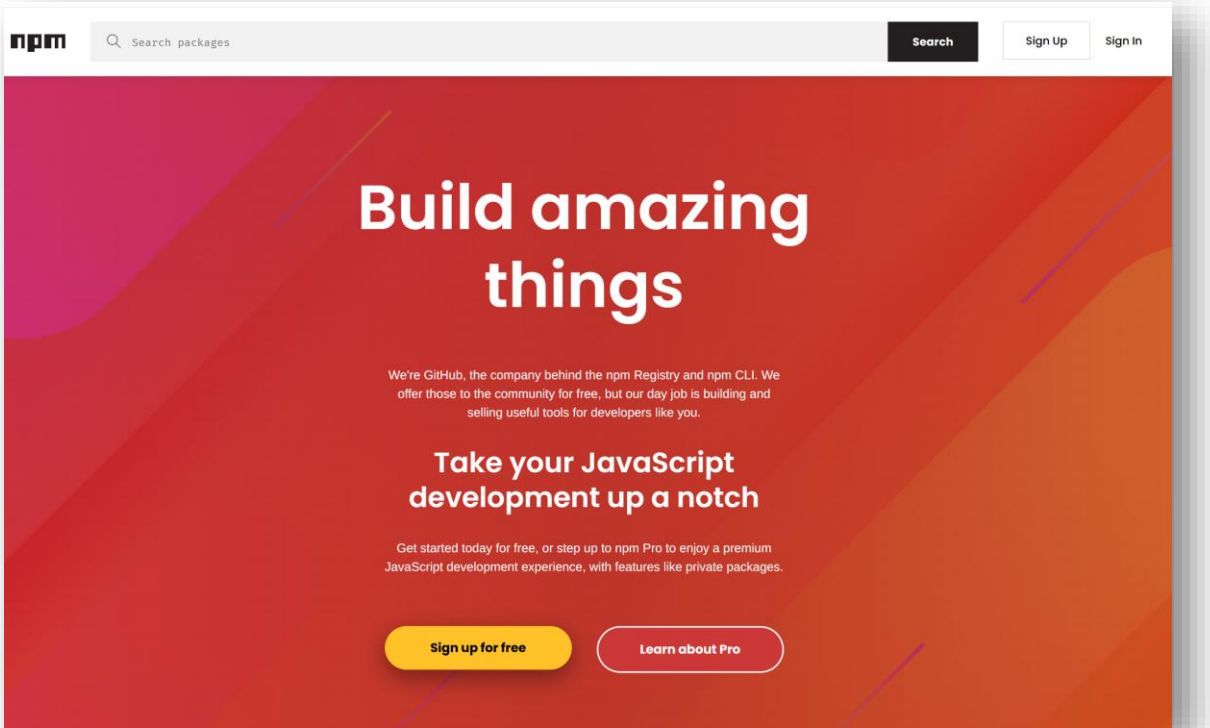
- O Gradle é *cross-platform* e pode ser usado com diversas linguagens, inclusive Java.
- Ele é totalmente baseado no Maven, e compartilham várias características!
- É o padrão em projetos Android

NPM - Dependências!

- Onde elas ficam?
- O que fazem?
- Como achamos?
- Como usamos?

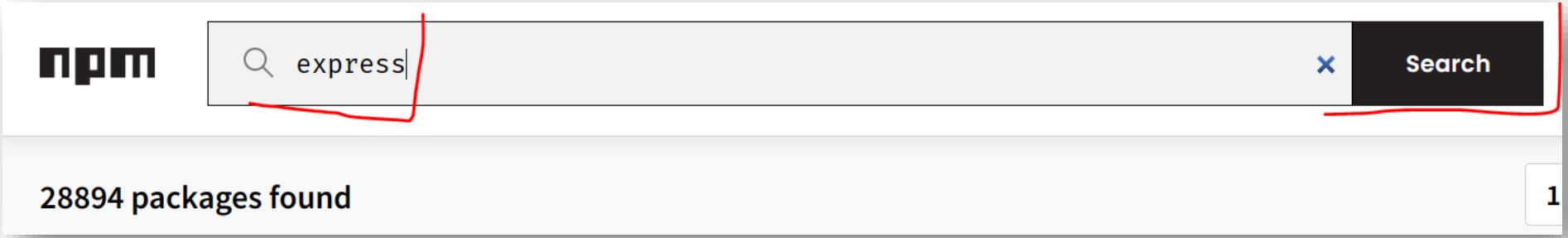
NPM – Repositório Central!

- https://www.npmjs.com/



NPM – Repositório Central!

- É onde se encontram as bibliotecas, frameworks, ferramentas e qualquer outro software que desejamos utilizar em nossas próprias soluções.
- Basta procurar por uma dependência



NPM – Inicialização

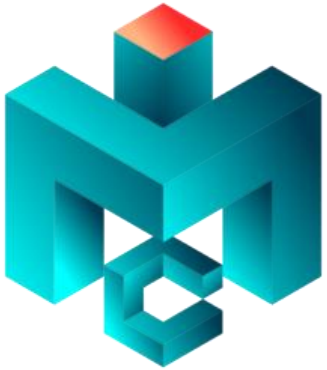
- Crie um diretório onde deseja criar o projeto
- Execute “npm init” no diretório escolhido e pressione “Enter” em todas as perguntas para ficar a opção padrão.
- Após encontrar as bibliotecas remotas que deseja utilizar execute “npm install <nome lib>” ou “npm i <nome lib>
- No código JS lembre-se de importar com ‘require’.

Exemplo 7 - NPM

- Utilizando as dependências listadas crie um programa que imprima a lista de países que falam inglês com uma cor aleatória.
  - @colors/colors
  - country-data
- É necessário estudar a documentação dessas duas APIs.
- Dica: Pesquisa como usar a função “filter” do JS

Executando Projetos NPM

- Caso esteja acessando um projeto que já possua o arquivo “package.json”, execute o comando “npm install”.
- Esse comando irá baixar todas as dependências listadas no arquivo “package.json”.
- Caso queira compartilhar esse projeto lembre-se de remover (ou ignorar) a pasta ‘node_modules’.
- Esta pasta deve ser gerada localmente por “npm install”



Aula – 10
Introdução ao Node

Disciplina: XDES03 – Programação Web

Prof: Phyllipe Lima Francisco
phyllipe@unifei.edu.br

Universidade Federal de Itajubá – UNIFEI
IMC – Instituto de Matemática e Computação