

**INSTITUTO FEDERAL**

Paraná

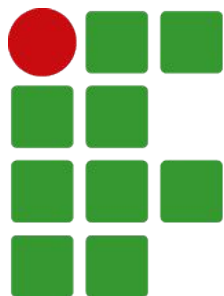
Campus Paranaguá

# Node.js

## Servidores Web

Desenvolvimento Web

Prof. Diego Stiehl



**INSTITUTO FEDERAL**

Paraná

Campus Paranaguá

Primeiramente, vamos relembrar.

## **Revisão**

# Revisão

---

- 1º Bimestre
  - HTML
  - CSS
- 2º Bimestre
  - JavaScript
- Somente **Front-end**
  - Tudo é executado e renderizado no browser

# Páginas Locais

---

- A forma como trabalhávamos não permitia a **publicação** das nossa páginas
  - Como o usuário chegará até nós?
  - De onde ele irá baixar as páginas?
  - Qual o nome de domínio?
    - Qual o endereço IP?
- Repare nas nossas URLs
  - **file:///home/aluno/meusite/index.html**
    - Local ← Somente meu computador tem acesso

# Servidor Web

---

- Porque usar um servidor
  - Páginas servidas via protocolo HTTP
    - É o padrão da web
  - Acessar bancos de dados
  - Páginas customizadas
  - Processamento pesado
  - Esconde código-fonte e dados do usuário
    - Problema → JavaScript, HTML e CSS podem ser lidos e manipulados diretamente no browser
  - ...

# Servidor Web

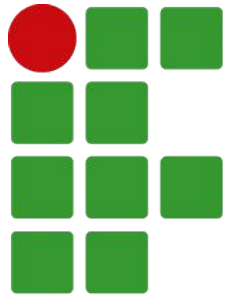
---

- Um servidor web fornece recursos via **protocolo HTTP**
  - Usuários podem acessá-lo
- O servidor costuma ser de dois tipos
  - Páginas estáticas
    - Fornece apenas HTML, CSS e JS puros
  - Páginas dinâmicas
    - Os recursos fornecidos são **customizados no servidor** antes de serem fornecidos ao usuário

# Ferramentas

- Precisaremos de ferramentas e uma linguagem de programação





**INSTITUTO FEDERAL**

Paraná

Campus Paranaguá

Executando JavaScript fora do browser

**Node.js**



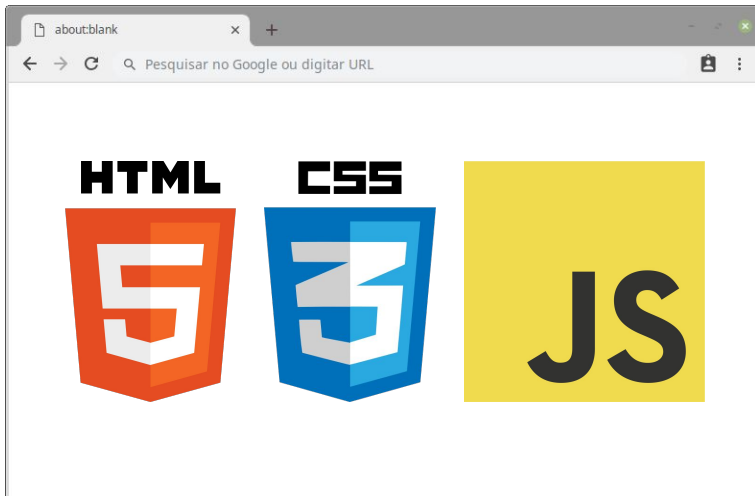
- Definição:
  - Node.js® é um runtime JavaScript desenvolvido com o Chrome's V8 JavaScript engine.



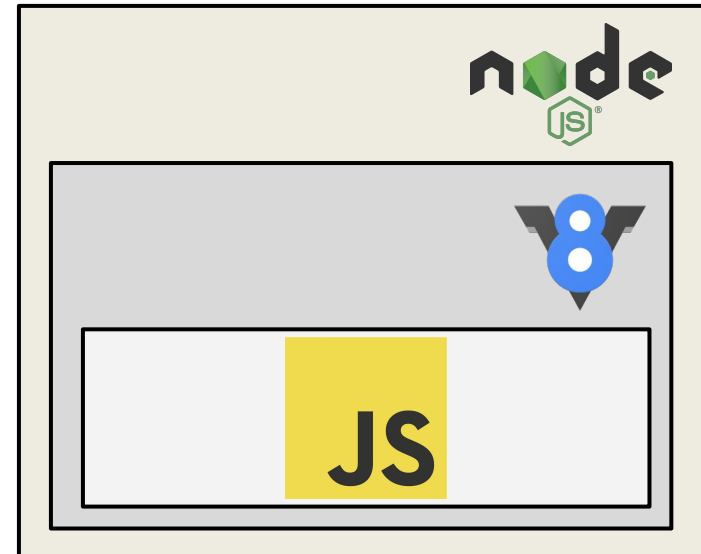
- Site: <https://nodejs.org>
- Criado em 2009

# JavaScript fora do Browser

Browser do usuário



Qualquer computador  
(ex: meu servidor)



# JavaScript no computador

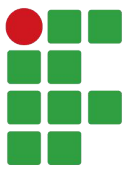
---

- Com Node.js podemos executar JavaScript em qualquer computador
  - Antes ele era restrito ao browser
- Node.js estende as capacidades do JavaScript
  - Acessar sistema de arquivos
  - Melhoria nas capacidades de rede
  - Detalhes do sistema operacional
  - Módulos
  - ...

# Possibilidades

- Node.js === novas possibilidades
- Principais:
  - Desenvolvimento para dispositivos móveis
    - React Native
  - Desenvolvimento para desktop
    - Electron
  - **Desenvolvimento para a web (server-side)**





# Por quê?

- Mas por que JavaScript no servidor?
  - Porque não uma linguagem mais clássica?
    - PHP, Java, Python, ASP.NET, ...
- Motivo:
  - JavaScript em todo o ciclo de desenvolvimento
  - Você já é obrigado a usar JavaScript no **front-end**
    - Por que aprender uma nova linguagem?





# Por quê?

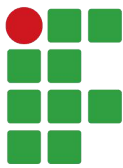
---

- Comunidade de desenvolvedores muito ativa
  - Muitas bibliotecas (NPM)
- Muitas empresas importantes usando
  - Netflix
  - Trello
  - Paypal
  - LinkedIn
  - Walmart
  - Uber
  - NASA

# Instalação

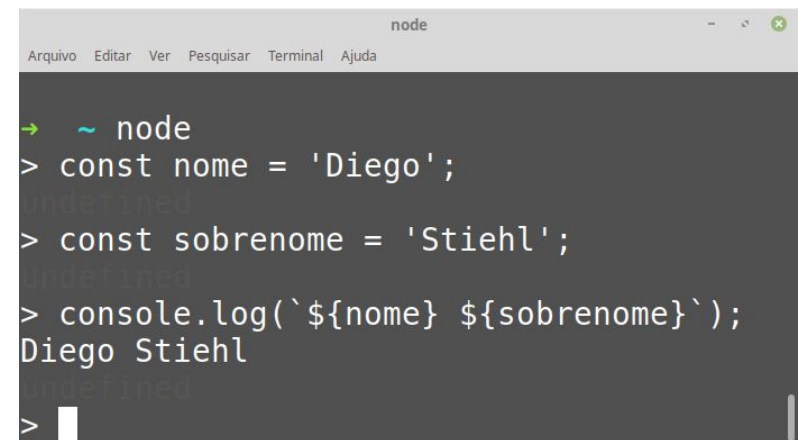
---

- Acessar <https://nodejs.org>
- Baixar a última versão estável
  - Hoje 10.16.1 LTS
  - Cuidar com sistema operacional e 32/64 bits
- Instalar no seu sistema
  - Talvez precise reiniciar
- Observação
  - No laboratório já está instalado



# REPL

- O Node.js vem com um REPL
  - *Read-eval-print loop*
- Semelhante ao Console do browser
- Podemos executar código JavaScript
- Para abrir:
  - `node` ← No terminal
- Para sair:
  - Ctrl + D



```
node
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
> ~ node
> const nome = 'Diego';
> const sobrenome = 'Stiehl';
> console.log(`${nome} ${sobrenome}`);
Diego Stiehl
>
```



# Arquivo .js

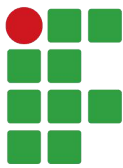
---

- O comando **node** também é usado para executar nossos arquivos **.js**
- Executar:
  - `node arquivo.js`

# Prática

---

- Crie um arquivo **index.js**
- Crie 3 variáveis constantes
  - nome, sobrenome, idade
- Usando `template strings` crie uma variável `nomeCompleto`
- Se a idade for maior ou igual a 18:
  - Imprimir → Seja bem vindo [NOME COMPLETO].
- Se não:
  - Imprimir → Olá amiguinho [NOME COMPLETO]!
- Executar com o **node**



# Módulos

---

- Node.js provê muita modularidade de código
  - Bibliotecas
- Módulo é uma forma de exportar objetos JavaScript para outros usarem
- Podemos usar módulos
  - Do próprio Node.js
  - Criados por nós
  - De terceiros

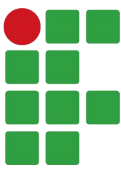
# Usando Módulos

---

- Para usar um módulo usamos a sintaxe:

```
const modulo = require( 'nome-modulo' );
```

- O módulo da string precisa existir no meu ambiente
- A variável criada dá acesso a tudo que o módulo exporta
  - Funções, propriedades, objetos, ...



# Módulo fs

- Módulo que dá acesso ao sistema de arquivos
  - Já vem com o Node.js
- Lendo um arquivo de texto:

```
const fs = require('fs');  
fs.readFile('./public/arquivo.txt', 'utf-8', function(err, data) {  
  if (err) {  
    console.log(err);  
  } else {  
    console.log(data);  
  }  
});
```

# Arrow Functions

- Versões mais novas do JavaScript permitem uma “simplificação” na escrita de funções
- Considere a função:

```
const funcao = function(param1, param2) {  
    console.log(param1, param2);  
};
```

- Ela também pode ser escrita assim:

```
const funcao = (param1, param2) => {  
    console.log(param1, param2);  
};
```

Arrow (seta)

# Arrow Functions - Exemplos

```
const funcao = function(param1, param2) {  
  console.log(param1, param2);  
};
```



```
const funcao = (param1, param2) => {  
  console.log(param1, param2);  
};
```

```
const funcaoUmParametro = function(unicoParam) {  
  console.log(unicoParam);  
};
```



```
const funcaoUmParametro = unicoParam => {  
  console.log(unicoParam);  
};
```

```
const funcaoQueRetornaSemParam = function() {  
  const mensagem = 'Olá amigo';  
  return mensagem;  
};
```



```
const funcaoQueRetornaSemParam = () => {  
  const mensagem = 'Olá amigo';  
  return mensagem;  
};
```

```
const funcaoRetornoUmaLinha = function(n1, n2) {  
  return n1 + n2;  
};
```



```
const funcaoRetornoUmaLinha = (n1, n2) => n1 + n2;
```

# Arrow Functions

---

- Acostume-se com a sintaxe
- Vamos usar Arrow Functions sempre
- Vantagens
  - Melhor legibilidade do código
  - Código mais enxuto
  - Facilita a criação de callbacks:

```
objeto.fazAlgunaCoisa('parâmetro', 10, (dados, erro) => {  
  console.log(dados);  
  // Parâmetro "erro" poderia ser oculto, pois não foi usado  
});  
objeto.fazOutraCoisa('parâmetro', valor => valor + 50);
```



# Prática

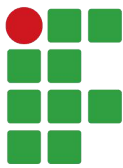
---

- Reescreva o script que lê um arquivo de texto
  - Utilize uma arrow function para o callback
- Crie um script que escreva algo neste arquivo
  - Dica:

`fs.writeFile`



- **npm** (*Node Package Manager*)
- É um serviço de registro e distribuição de pacotes JavaScript
- Consiste em três componentes
  - Site (<https://www.npmjs.com>)
  - Command Line Interface (CLI)
  - Repositório de pacotes
- Mais informações:
  - <https://docs.npmjs.com/about-npm>



# npm

---

- O **npm** permite o uso de módulos criados por terceiros
  - Busca, verifica versão, faz download, vincula ao nosso projeto, atualiza, ...
- Facilidade para instalar pacotes
- Gestão das dependências do nosso projeto
  - Facilita trabalho em equipe
- Possibilita a configuração e execução de scripts

# Usando o npm

---

- Na pasta do projeto, digitar no terminal:
  - `npm init`
- Irá solicitar algumas configurações
  - Apenas confirme
- Ao final irá gerar um arquivo
  - `package.json`
    - Será utilizado para todas as configurações de dependências do projeto

# package.json

---

Arquivo gerado:

```
{  
  "name": "projeto",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}
```

# Instalando um pacote do npm

---

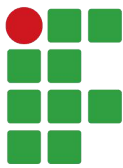
- Para instalar um pacote no nosso projeto:
  - 1. Busque no site: <https://www.npmjs.com>
  - 2. Execute o comando na pasta do projeto:
    - `npm install nome-do-pacote --save`
- O npm irá verificar, baixar e vincular o pacote
  - Ele irá modificar o arquivo package.json

```
"dependencies": {  
  "nome-do-pacote": "versão",  
  "outro-pacote": "versão",  
}
```

# Pacote **cpf**

---

- Vamos instalar o pacote **cpf**
  - <https://www.npmjs.com/package/cpf>
  - Valida, gera e formata CPFs
- Executar
  - `npm install cpf --save`
- Ver package.json
- Ver diretório criado → `node_modules`
  - Todas nossas dependências ficam nele



# Dicas

---

- npm install pode ser abreviado
  - npm i nome-do-pacote --save
- Ao baixar um projeto feito por outra pessoa
  - Executar:
    - npm install
  - Irá ler o package.json e baixar todos pacotes



# Usando pacote no projeto

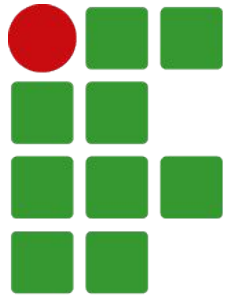
---

- Pacotes normalmente exportam módulos
- Então:

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

```
const cpfGerado = cpf.generate(true);  
console.log(cpfGerado);
```

- Sempre leia a documentação do pacote



**INSTITUTO FEDERAL**

Paraná

Campus Paranaguá

Finalmente

# **Criando um Servidor Web**

# Servidor Web

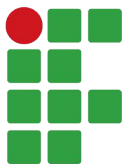
---

- Servidor web (ou **servidor HTTP**) é uma aplicação infinitamente ativa
  - Nunca termina
- Ela fica esperando por novas conexões via **protocolo HTTP** (*HyperText Transfer Protocol*)
  - As chamamos de requisições (*requests*)
- Ao receber uma requisição, ele gera uma resposta (*response*) equivalente
  - Devolve esta resposta ao usuário (browser)

# Módulo http

---

- O Node.js tem um módulo http
- Com ele podemos criar um server
- O server pode ouvir (listen) uma porta
- O servidor fica ativo “para sempre”
  - Sempre que receber uma requisição, devolverá uma resposta



# Nosso Servidor

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

// req: objeto com dados da requisição
// res: objeto com dados da resposta
const server = http.createServer((req, res) => {
  // Imprime no console do servidor
  console.log(`REQUISIÇÃO: ${req.url}`);

  // O código de status 200 significa OK (sucesso)
  res.writeHead(200, { 'Content-type': 'text/html' });

  // Esta string define o que será retornado para o browser
  res.end('<h1>Bem vindo ao nosso site!</h1>');
});

server.listen(8000, '127.0.0.1', () => {
  console.log('Ouvindo na porta 8000...');
  console.log('Aceitando somente requisições locais (127.0.0.1 ou localhost).');
  console.log('Para acessar o servidor, digite: http://localhost:8000 no browser.');
```

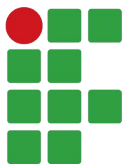


# Rotas

- Neste ponto, nosso servidor só sabe devolver uma coisa

```
<h1>Bem vindo ao nosso site!</h1>
```

- A propriedade req.url pode ser usada
  - Ela é uma **string** e pode ser comparada
  - Assim começamos a montar as rotas internas da nossa aplicação
    - Rotas === URLs válidas



# Rotas

```
const server = http.createServer((req, res) => {  
  res.writeHead(200, { 'Content-type': 'text/html' });  
  if (req.url === '/') {  
    res.end('<h1>Bem vindo ao nosso site!</h1>');  
  } else if (req.url === '/sobre') {  
    res.end('<h1>Somos uma empresa bacana.</h1>');  
  }  
  // SEMPRE deve haver um "res.end()" ao final da comunicação  
  // A falta de "else" pode gerar problemas  
  // Teste no seu browser com uma URL inválida  
});
```

# Prática

---

- Conserte nosso servidor para que sempre haja uma resposta
  - A requisição nunca pode entrar em loop infinito
- Em caso de erro (URL inválida)
  - Utilizar status 404 ao invés de 200
    - 404 === “not found”
  - Devolver mensagem → “Recurso inexistente”



# Liberar servidor na rede

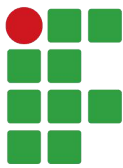
---

- Faça seu servidor “ouvir” no IP **0.0.0.0**
- Qualquer computador na rede poderá acessar
  - Precisar  saber o seu IP na rede local
    - Ver no comando ifconfig no terminal

```
server.listen(8000, '0.0.0.0', () => {  
  console.log('Ouvindo na porta 8000...');  
  console.log('Aceitando requisições de qualquer local.');
```

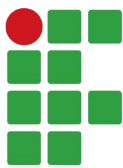
```
  console.log('Para acessar o servidor, digite: http://SEU-IP:8000 no browser.');
```

```
});
```



# nodemon

- Ferramenta que monitora alterações no projeto e reinicia o servidor
- Para instalar, execute:
  - `npm i nodemon -g`
    - O `-g` instala globalmente
- Depois execute
  - `npm i nodemon --save-dev`
    - Para incluir como dependência de desenvolvimento do nosso projeto
      - Ver package.json



# Atividade

- Baixe do Moodle o Jogo da Força completo
  - Coloque dentro da pasta public no seu projeto
- Faça um servidor web que forneça o conteúdo da pasta public
  - URL → <http://localhost:5555>
    - Ao acessar a URL acima (**sem mudar nada**), o usuário deve ver o jogo da força completo
  - Lembre que o servidor deve fornecer HTML, CSS, JavaScripts, imagens, ...
    - O servidor é indiferente com relação ao que estiver na pasta public: tudo deve ser fornecido

# Atividade - Dicas

- Saber se um arquivo existe

```
fs.exists(fileName, callback);
```

- Módulo que retorna o tipo de um arquivo
  - text/html, text/css, ...
  - <https://www.npmjs.com/package/mime-types>

```
const mime = require('mime-types');  
const contentType = mime.lookup(fileName);
```

- Ler arquivos sem informar “utf-8”

```
fs.readFile(fileName, (err, data) => {  
    // Útil para fornecer as imagens, por exemplo  
});
```