

INTRODUÇÃO AO WEBPACK



Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifrroma@gmail.com

Slides: <https://git.io/vAd6S>

AGENDA

- Node JS
- Node Package Manager (NPM)
- Instalação
- Primeiros passos com Node JS e NPM
- Acessando módulos externos
- Introdução ao webpack
- Instalação e uso básico
- Configuração
 - Básico
 - Loaders
 - Plugins
- Desenvolvimento
- Produção

JAVASCRIPT

JAVASCRIPT



NODE JS

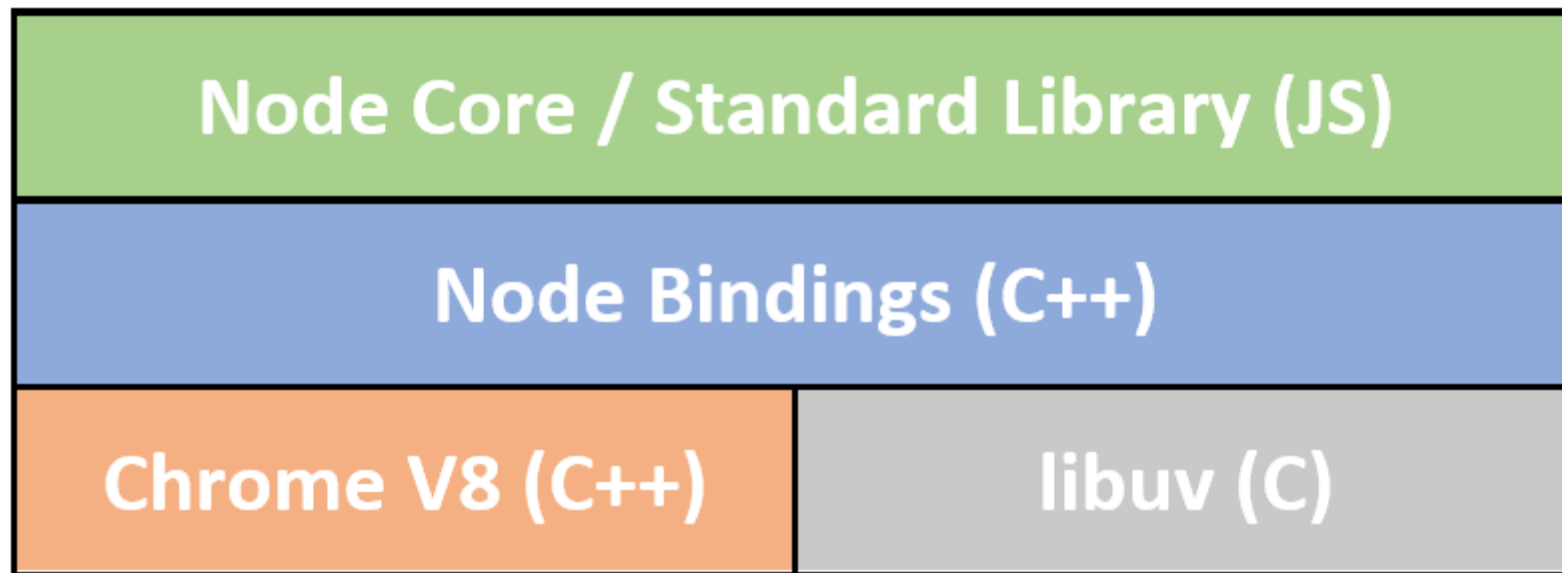
O QUE É?

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, **npm**, is the largest ecosystem of open source libraries in the world.

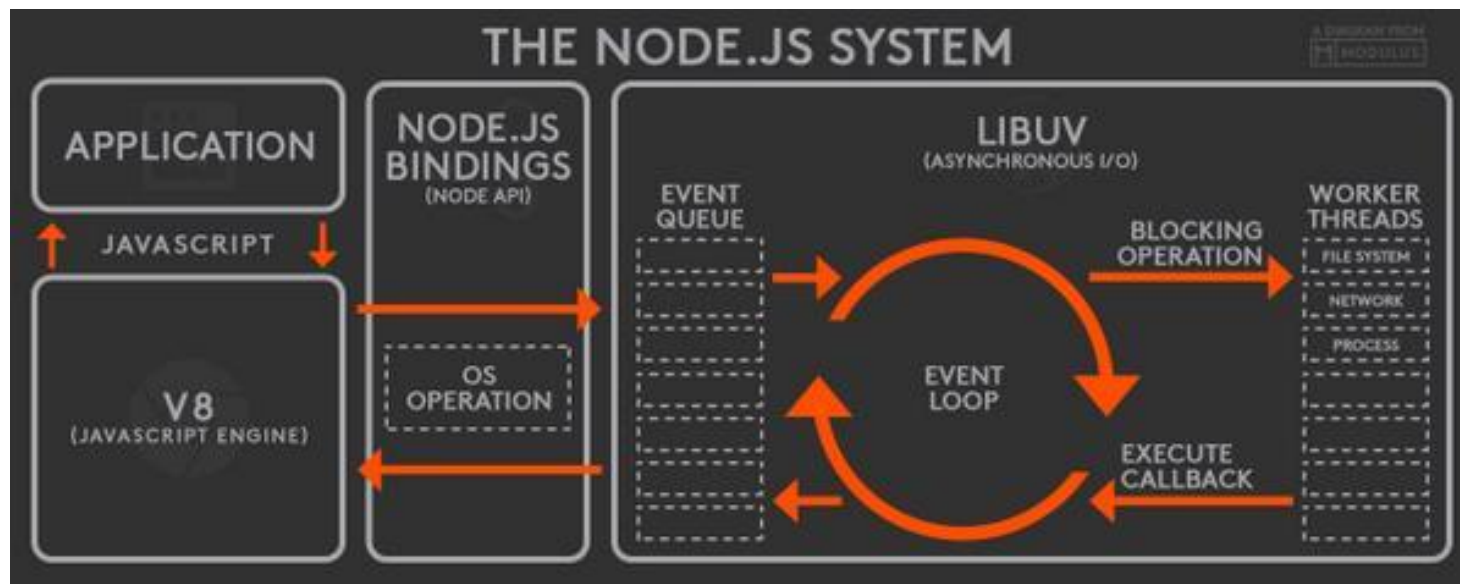
<https://nodejs.org/>

- Runtime JavaScript desvinculado do navegador
- Orientado à eventos
- Gratuito e de código aberto
- Criado por Ryan Dahl

ARQUITETURA



SISTEMA



ONDE USAR?

- APIs
- Backend para Jogos, IoT e Apps
- Aplicações em tempo real
- Automatizar tarefas



QUEM UTILIZA?



U B E R



NETFLIX

NODE PACKAGE MANAGER (NPM)

NODE PACKAGE MANAGER (NPM)



INSTALANDO NO WINDOWS

1. Instale o Git com GitBash (Recomendado)

<https://git-scm.com/download>

2. Faça o download do NodeJS de acordo com sua arquitetura

<https://nodejs.org/en/download/>

3. Execute o instalador

Next, Next, Next and Finish.

INSTALANDO EM DEBIAN LIKE

1. Adicione o repositório de pacotes

```
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
```

2. Execute a instalação

```
$ sudo apt-get install -y nodejs
```

INSTALANDO EM RED HAT LIKE

1. Adicione o repositório de pacotes

```
$ curl --silent --location https://rpm.nodesource.com/setup_8.x | sudo bash -
```

2. Execute a instalação

```
$ sudo yum -y install nodejs
```



INSTALANDO NO MAC OSX

1. Instale o Homebrew (caso ainda não tenha)

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. Execute a instalação

```
$ brew install node@8
```


INSTALAÇÃO

- Verificando a versão do Node

```
$ node --version
```

- Verificando a versão do NPM

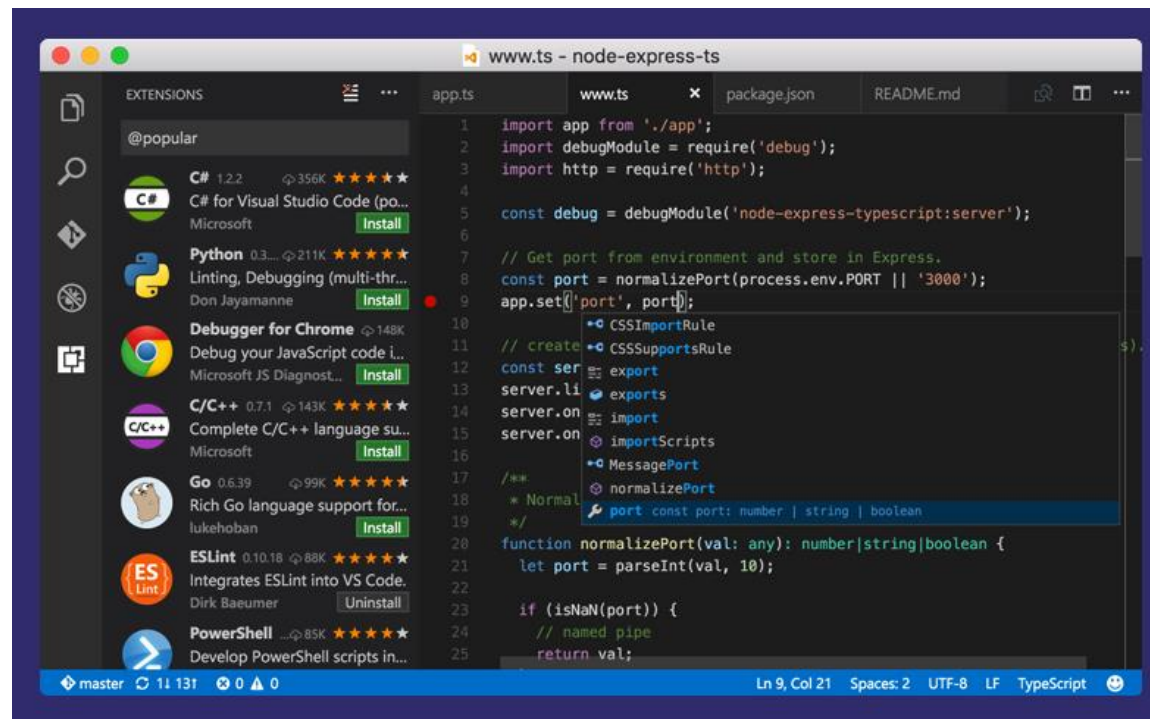
```
$ npm --version
```

- Atualizando o NPM

```
$ npm install -g npm
```

BÔNUS

- Visual Studio Code (Recomendado)
 - <https://code.visualstudio.com/>
 - Debug integrado
 - Auto-completar
 - Identificação automática
 - Terminal integrado
 - Variedade de extensões



PRIMEIROS PASSOS COM NODE JS

- Crie uma pasta para o projeto. Ex: MeuPrimeiroProjeto
- Crie um arquivo `index.js` que imprima uma mensagem no console.

```
console.log('Olá Node JS!');
```

- Execute o arquivo com o Node JS.

```
$ node index.js
```

PRIMEIROS PASSOS COM NPM

- Dentro do diretório `MeuPrimeiroProjeto` inicie o arquivo `package.json`.

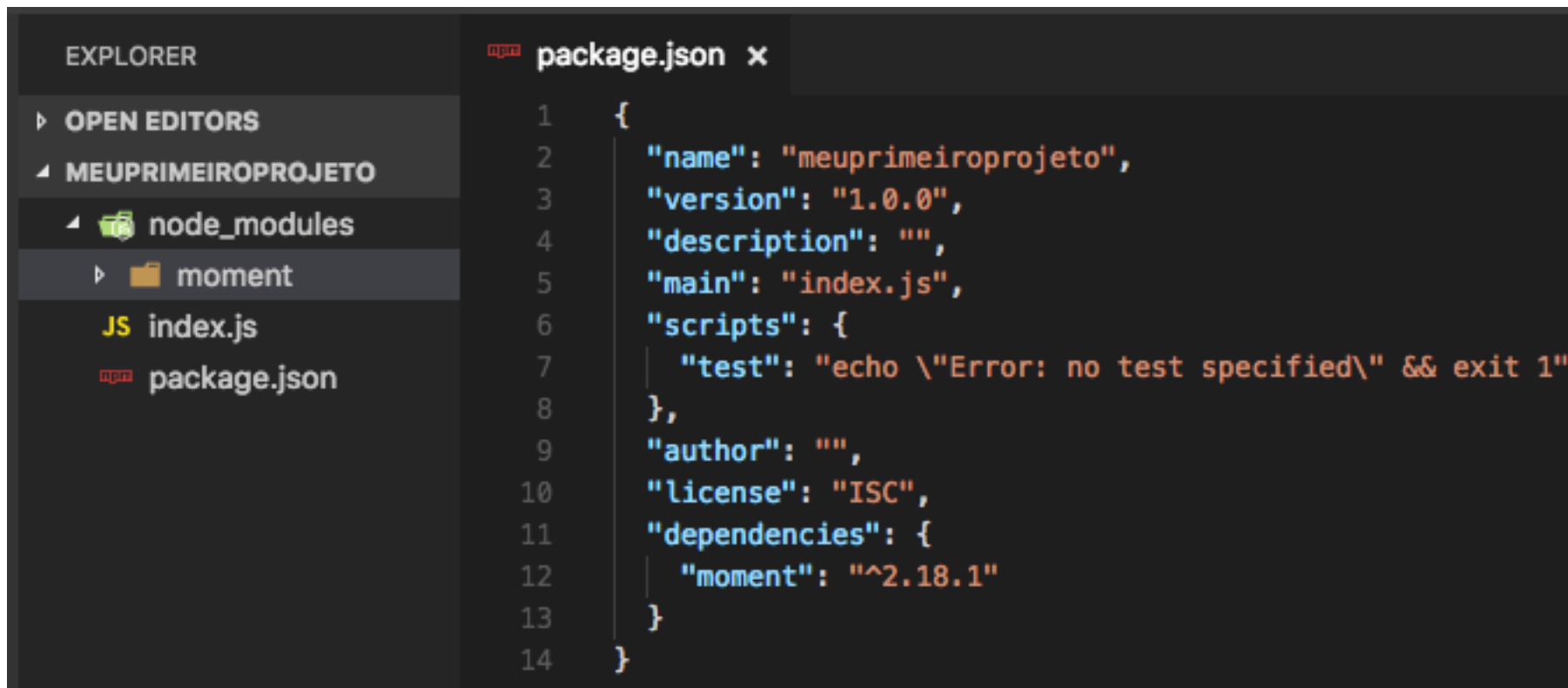
```
$ npm init
```

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

PRIMEIROS PASSOS COM NPM

- Instale a dependência `moment` para manipulação de datas.

```
$ npm install --save moment
```



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure: 'MEUPRIMEIROPROJETO' containing a 'node_modules' directory (expanded) with a 'moment' folder, and files 'index.js' and 'package.json'. The main editor area shows the 'package.json' file with the following content:

```
1  {
2    "name": "meuprimeiroprojeto",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "moment": "^2.18.1"
13   }
14 }
```

ACESSANDO MÓDULOS EXTERNOS

- Importe o módulo `moment` e imprima a data atual, formatada, no console.

`index.js`

```
const moment = require('moment');  
  
const dataAtual = new Date();  
  
const dataFormatada =  
moment(dataAtual).format('DD/MM/YYYY HH:mm:ss');  
  
console.log(dataFormatada);
```

- Execute o arquivo com o Node JS

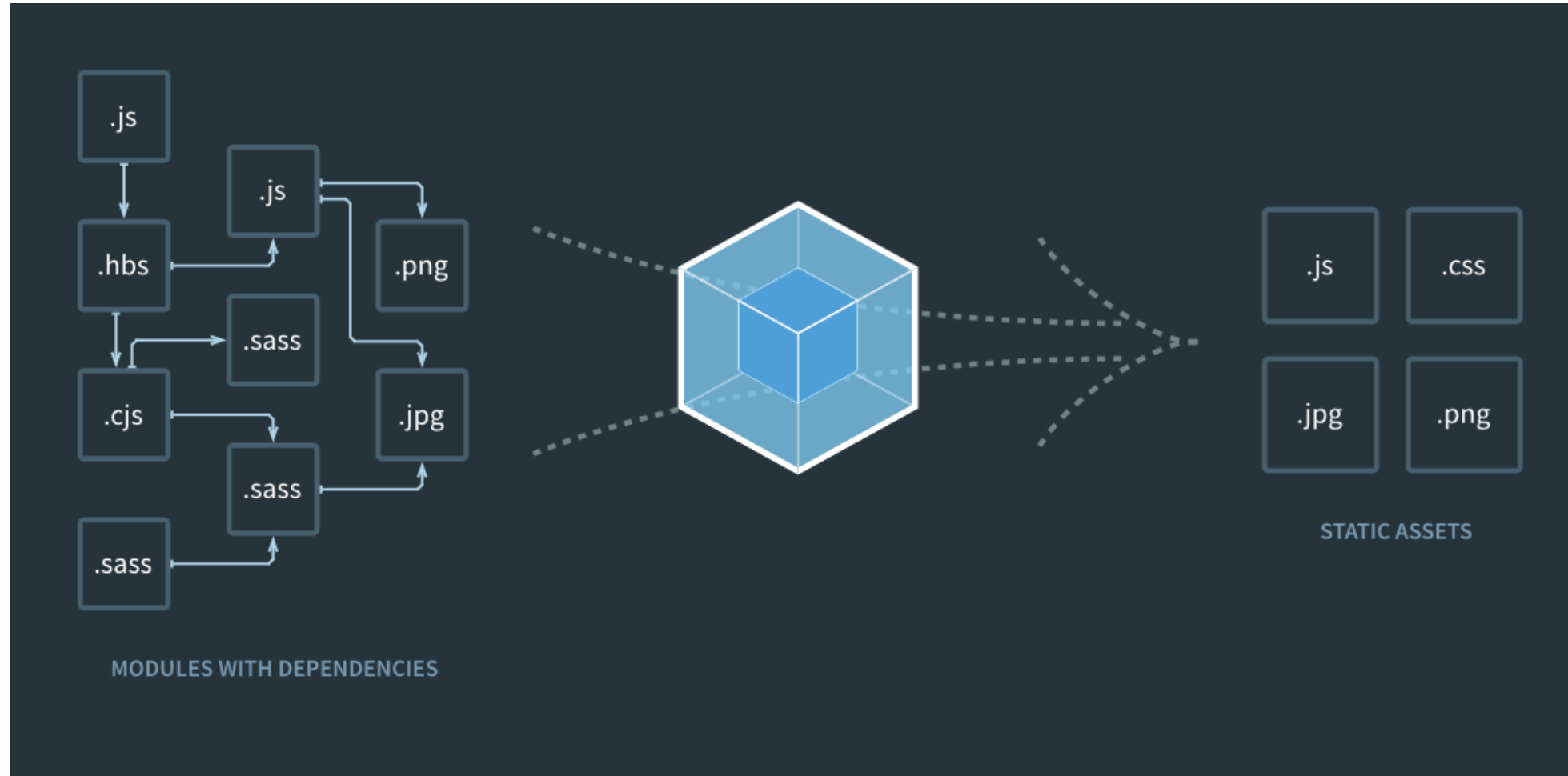
```
$ node index.js
```

WEBPACK

INTRODUÇÃO AO WEBPACK

- `webpack` (com “w” minúsculo) é um empacotador de código para projetos web, assim como o [browserify](#).
- O objetivo do `webpack` não é apenas unir todos os arquivos JS e CSS em um único pacote, mas também possibilitar a divisão do projeto em módulos reaproveitáveis e com isso auxiliando no trabalho em equipe.
- Entretanto, o `webpack` não é recomendado apenas para projetos grandes, tornando-se uma ferramenta muito útil também para projetos pequenos.

INTRODUÇÃO AO WEBPACK



INSTALANDO

- Essencialmente o `webpack` deve ser instalado como uma dependência de desenvolvimento do seu projeto.

```
$ npm install -D webpack webpack-cli
```

- Porém, pode ser interessante também instalar o `webpack` globalmente, para facilitar a execução de scripts e testes rápidos.

```
$ npm install -g webpack webpack-cli
```

USO BÁSICO

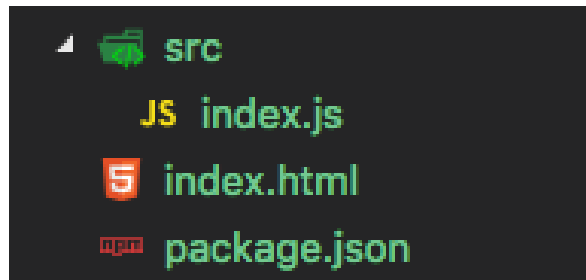
- Para entender o real papel do webpack, primeiro vamos criar uma estrutura tradicional de um projeto web.

```
$ mkdir meu-projeto-webpack
```

```
$ cd meu-projeto-webpack
```

```
$ npm init -y
```

- Em seguida crie a seguinte estrutura de arquivos:



USO BÁSICO

- index.html

```
<html>

<head>
  <meta charset="utf-8" />
  <title>Introdução ao webpack</title>
  <script src="https://unpkg.com/lodash@4.16.6"></script>
</head>

<body>
  <script src="./src/index.js"></script>
</body>

</html>
```

USO BÁSICO

- src/index.js

```
function component() {  
  var element = document.createElement('div');  
  // Lodash é utilizado como variável global por meio da declaração  
do <script> no index.html  
  element.innerHTML = _.join(['Olá', 'webpack', '!'], ' ');  
  return element;  
}  
  
document.body.appendChild(component());
```

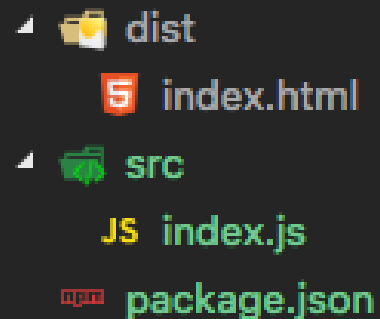
USO BÁSICO

- Para converter este projeto para utilizar o webpack, é preciso instalar as dependências necessárias:

```
$ npm install -D webpack webpack-cli
```

```
$ npm install lodash
```

- Atualizar a estrutura do projeto para:



```
└─ dist
  └─ index.html
└─ src
  └─ index.js
package.json
```

USO BÁSICO

- dist/index.html (novo)

```
<html>

<head>
  <meta charset="utf-8" />
  <title>Introdução ao webpack</title>
</head>

<body>
  <script src="bundle.js"></script>
</body>

</html>
```

USO BÁSICO

- src/index.js (novo)

```
import _ from 'lodash';

function component() {
  var element = document.createElement('div');

  // Lodash agora é importado do node_modules
  element.innerHTML = _.join(['Olá', 'webpack', '!'], ' ');

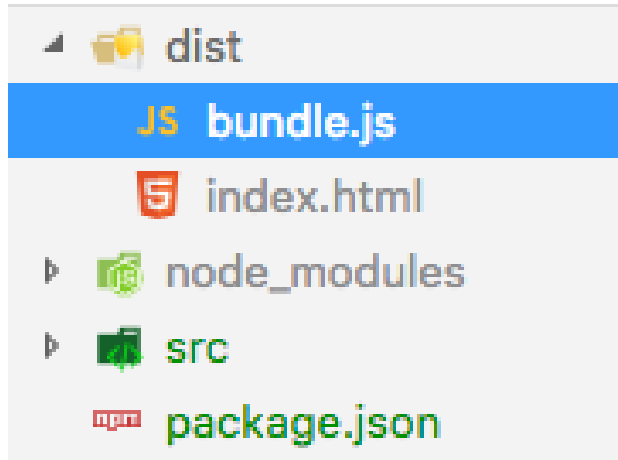
  return element;
}

document.body.appendChild(component());
```


USO BÁSICO

- Por fim, para empacotar o projeto execute o comando:

```
$ npx webpack -o dist/bundle.js
```



CONFIGURAÇÃO

- A maioria dos projetos pode precisar de uma configuração mais complexa, para isso o `webpack` suporta a criação de arquivos de configuração.
- Para utilizar o arquivo de configuração, basta criar um arquivo chamado `webpack.config.js` na raiz do projeto e acrescentar o seguinte conteúdo.

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

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  }
};
```

```
$ npx webpack --config webpack.config.js
```

LOADERS

- Loaders são auxiliares que permitem que o webpack saiba trabalhar com outros tipos de arquivos além do JavaScript. Uma vez que o loader é configurado, você está dizendo ao webpack o que deve ser feito quando um arquivo daquele formato for encontrado.
- Por exemplo, para possibilitar que o webpack saiba carregar arquivos do tipo CSS, é preciso instalar e configurar os seguintes módulos:

```
$ npm install -D style-loader css-loader
```

LOADERS

- webpack.config.js

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

module.exports = {
  // entry, output,
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          'style-loader',
          'css-loader'
        ]
      }
    ]
  }
};
```

LOADERS

- Uma vez que o loader de CSS foi configurado, é possível importar os arquivos de estilo utilizando o `import`.

src/index.js

```
import _ from 'lodash';
import './styles.css';

function component() {
  var element = document.createElement('div');

  element.innerHTML = _.join(['Olá', 'webpack', '!'], ' ');
  element.classList.add('hello');

  return element;
}

document.body.appendChild(component());
```

src/styles.js

```
.hello {
  color: red;
}
```

LOADERS

- Loaders também podem ser utilizados para carregar imagens.

```
$ npm install -D file-loader
```

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

module.exports = {
  // entry, output,
  module: {
    rules: [
      ... ,
      {
        test: /\. (png|svg|jpg|gif) $/,
        use: [
          'file-loader'
        ]
      }
    ]
  }
};
```

LOADERS

- Uma vez que o arquivo de imagem for importado, será retornada uma `String` contendo o caminho para o arquivo.

```
...  
import Icon from './icon.svg';  
  
function component() {  
  ...  
  
  var myIcon = new Image();  
  myIcon.src = Icon;  
  
  element.appendChild(myIcon);  
  
  return element;  
}  
  
document.body.appendChild(component());
```

PLUGINS

- Outro recurso interessante do webpack é a possibilidade de adição de plugins. Isso permite que você possa adicionar funcionalidades extras que o webpack não atende por si só.
- Por exemplo, podemos adicionar o `html-webpack-plugin` para gerenciar a criação do `index.html` à partir de um template.

```
$ npm install -D html-webpack-plugin
```


PLUGINS

- Configurando o `html-webpack-plugin`:

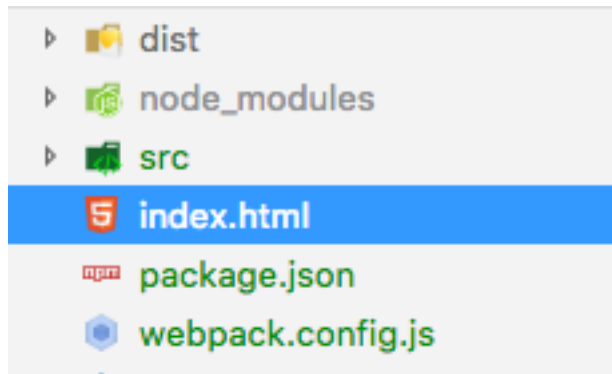
`webpack.config.js`

```
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  ...,
  plugins: [
    new HtmlWebpackPlugin({
      template: './index.html'
    })
  ],
  ...
};
```

PLUGINS

- Então, o arquivo de template `index.html` agora deve ficar na raiz do projeto.



```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Projeto webpack!</title>
</head>

<body>
</body>

</html>
```

DESENVOLVIMENTO

- Durante o desenvolvimento do projeto pode ser interessante configurar alguns recursos úteis, como reload automático da página e criação dos source-maps.
- Para criação do source-maps basta adicionar a seguinte propriedade ao `webpack.config.js`:

```
module.exports = {  
  ... ,  
  mode: 'development' ,  
  devtool: 'inline-source-map' ,  
  ...  
};
```

DESENVOLVIMENTO

- Para o reload automático, podemos utilizar o webpack-dev-server:

```
$ npm install -D webpack-dev-server
```

- Configurar o `webpack.config.js`:

```
module.exports = {  
  ... ,  
  devServer: {  
    contentBase: './dist'  
  },  
  ...  
};
```

- E adicionar o script ao `package.json`:

```
"scripts": {  
  "start": "webpack-dev-server --open",  
  "build": "webpack"  
},
```

PRODUÇÃO

- Ao pensar em colocar um projeto Web em produção, existem alguns cuidados que devem ser tomados, como remoção dos comentários e minificação do código.
- Para facilitar este processo, vamos quebrar o arquivo e configuração do `webpack` em três partes:
 - `webpack.common.js`
 - `webpack.dev.js`
 - `webpack.prod.js`
- Instalar o `webpack-merge`:

```
$ npm install -D webpack-merge
```

PRODUÇÃO

webpack.common.js

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const CleanWebpackPlugin = require('clean-webpack-plugin');

module.exports = {
  entry: './src/index.js',
  plugins: [
    new CleanWebpackPlugin(['dist']),
    new HtmlWebpackPlugin({
      template: './index.html'
    })
  ],
  ...
```

```
...

module: {
  rules: [
    {
      test: /\.css$/,
      use: ['style-loader', 'css-loader']
    }, {
      test: /\. (png|svg|jpg|gif) $/,
      use: ['file-loader']
    }
  ]
}
};
```

PRODUÇÃO

webpack.dev.js

```
const merge = require('webpack-merge');
const common = require('./webpack.common.js');

module.exports = merge(common, {
  mode: "development",
  output: {
    filename: 'bundle.js'
  },
  devtool: 'inline-source-map',
  devServer: {
    contentBase: './dist',
    port: 8080,
  }
});
```

PRODUÇÃO

webpack.prod.js

```
const path = require('path');
const merge = require('webpack-merge');
const UglifyJSPlugin = require('uglifyjs-webpack-plugin');
const common = require('./webpack.common.js');

module.exports = merge(common, {
  mode: "production",
  output: {
    filename: '[hash].js',
    path: path.resolve(__dirname, 'dist')
  },
  plugins: [
    new UglifyJSPlugin()
  ]
});
```


PRODUÇÃO

package.json

```
...  
  
"scripts": {  
  "start": "webpack-dev-server --open --config webpack.dev.js",  
  "build": "webpack -p --config webpack.prod.js"  
},  
  
...
```

REFERÊNCIAS

- Node JS - <https://nodejs.org/>
- Lib UV - <http://libuv.org/>
- Node Package Manager (NPM) - <https://www.npmjs.com/>
- Git e GitBash - <https://git-scm.com/downloads>
- Visual Studio Code - <https://code.visualstudio.com/>
- Moment JS - <https://momentjs.com/>
- webpack - <https://webpack.js.org/>
- webpack guides - <https://webpack.js.org/guides/>
- Lodash - <https://lodash.com/>

DÚVIDAS?



Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifroma@gmail.com

Slides: <https://git.io/vAd6S>