

# Teoria do empacotador de módulos javascript webpack ↩️

---

## 1. INDEX

---

### 1. Introdução

1. [Objetivo.](#)
2. [Pre-requisitos.](#)
3. [Benefícios.](#)
4. [Descrição](#)
5. [Como usar webpack com a linguagem typescript](#)
  1. Vídeos
    1. [Webpack & TypeScript;](#)
    2. .
6. [Vídeo Aula de como usar webpack com typescript](#)

### 2. Instalação e configuração local

1. [Instalar a versão mais recente do webpack.](#)
2. [Instalar plugins html-webpack-plugin](#)
3. [Instalar a versão mais recente do webpack-cli.](#)
  1. [Instalar webpack-cli init](#)
  2. [Instalar webpack-cli generators](#)
  3. [Instalar nanoid.](#)
4. [Configuração do webpack.](#)
  1. [Configuração padrão.](#)
  2. [Configuração personalizada](#)
5. [Instalar a versão mais recente do webpack-dev-server.](#)
6. .
7. .

### 3. Exemplos.

1. [Exemplo básico](#)
2. [Exemplo de webpack com a linguagem javascript](#)
3. [Exemplo de webpack com a linguagem typescript.](#)
4. [Exemplo completo de uso do TypeScript e uma vídeo aula no youtube.com](#)
5. Exemplo do manual [webpack](#):
  1. [getting-started/#basic-setup.](#)

### 4. Referências.


### 5. Histórico.

## 2. CONTEÚDO


---

### 1. Introdução


#### 1. Objetivo:

1. O [webpack](#) é um compactador javascript e empacotador de módulo estático para aplicativos JavaScript. Quando o [webpack](#) processa seu aplicativo ele cria uma versão de distribuição na pasta `./dist` e copia para ela todos os arquivos `.js` compactado e todas as dependências tais como arquivos `.html`, `.css` e as imagens do projeto. [Veja mais no vídeo...](#) ou [Veja mais no site...](#)
2. []

#### 2. Pre-requisitos:

1. Assistir o vídeo [Webpack - Curso rápido para iniciantes](#)
2. Ultima versão do programa [nodejs](#) de preferência a versão LTS. Veja as versões do nodejs [aqui...](#);
3. []

#### 3. Benefícios:

1. Gerar uma pasta pronta para publicação na web.
2. Compactar todo código javascript e html e css.
3. Remove todo código não usado no projeto atual.
4. Gerar versão compatível com browser antigos usando o babel.
5. ...
6. []

#### 4. Descrição:

1. O [webpack](#) é um empacotador de módulo estático para aplicativos JavaScript modernos. Quando o [webpack](#) processa um projeto, ele constrói internamente um gráfico de dependência que mapeia todos os módulos de que o projeto precisa e gera um ou mais pacotes na pasta `./dist`. Para mais informações sobre o conceito do webpack veja [aqui...](#) e para acessar a documentação oficial completa clique [aqui...](#)
2. O webpack deve ser executado usando o comando **npx** para que ele execute a versão local do projeto do webpack.

1. Código ShellScript:

```
npx webpack
```

3. A configuração padrão do webpack não exige que você use um arquivo de configuração, no entanto, ele assumirá que o [ponto de](#)

/

**entrada** do projeto seja o arquivo **./src/index.js** e arquivo de saída seja **./dist/main.js**. O arquivo de saída será reduzido e otimizado para produção na pasta **./dist**.

4. Caso deseje mudar o comportamento padrão é possível criando o arquivo **webpack.config.js** na pasta do arquivo **package.js** porque ele espera que exista o arquivo **./index.html** na pasta do arquivo **package.json**. Veja [mais...](#)

1. As propriedades do arquivo **webpack.config.js** que pode ser personalizadas são:

1. **Entry**

1. Um ponto de entrada indica qual módulo webpack deve usar para começar a construir seu gráfico de dependência interna.

1. Exemplo de arquivo **webpack.config.js**:

```
module.exports = {  
  entry: './foo.js',  
},  
};
```

2. **Output**

1. A propriedade **output** informa ao webpack onde gerar os pacotes que ele cria e como nomear esses arquivos. O padrão é **./dist/main.js**.

1. Exemplo de arquivo **webpack.config.js**:

```
/**  
 * path é uma biblioteca NodeJS  
 padrão que está globalmente  
 disponível quando você instala o  
 NodeJS.  
 * A linha abaixo importa o  
 módulo path que tem informações  
 sobre a localização das pastas do  
 projeto.  
 * [Veja mais sobre o módulo  
 path...:]  
 (https://nodejs.org/api/path.html).  
 */  
const path = require('path');  
  
/**  
 * O objeto module.exports é um  
 objeto de nodejs usado para exportar  
 objetos,  
 * funções, variáveis e  
 constantes do módulo onde ele for  
 declarado.
```

```

/
  * [Vaja mais sobre
module.exports]
(https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/export)
  * [Veja mais sobre configuração
do arquivo webpack-config.js]
(https://webpack.js.org/configuration/)
  */
  module.exports = {
    /**
     * Propriedade entry é usada
para informa o nome do arquivo
principal
     * de entrada do pacote
webpack e pode ter mais um entrada
para o mesmo projeto.
     * O arquivo de entrada deve
importar todos os arquivos de
recursos dependentes do projeto.
     */
    entry: './src/file.js',

    /**
     * O objeto module é usado
para definir as regras usada na
compilação ele
     * contém o conjunto de plug-
ins ou módulos usados pelo webpack.
     */
    output: {
      path:
path.resolve(__dirname, 'dist'),
      filename: 'my-first-
webpack.bundle.js',
    },
  };

```

2. Essa configuração criará a pasta ./dist na pasta do pacote package.json e o arquivo './src/file.js' e suas dependências será compactado no arquivo './dist/my-first-webpack.bundle.js'..

### 3. Loaders

1. Por padrão o webpack só entende arquivos JavaScript, para que ele possa entender outros tipos de arquivos devemos usar os Loaders que são módulos que podem ser instalados separadamente possibilitando que o webpack converta esses arquivos em módulos válidos e os adicione ao gráfico de dependência. Os Loaders também são utilizados para converter JavaScript de uma versão para outra.

/

Para incluirmos os Loaders criamos uma nova seção module no arquivo de configuração, nessa seção podemos definir uma ou mais [rules](#):

1. Código javascript do arquivo

#### **webpack.config.js**

```
const path = require('path')
const config = {
  output: {
    filename: 'my-first-
webpack.bundle.js'
  },
  /**
   * O objeto module é usado
   para definir as regras usada na
   compilação ele
   * contém o conjunto de plug-
   ins ou módulos usados pelo webpack.
   */
  modules: {
    // O array rules são usado
    para passagem de regras na
    transpilação dos módulos:
    rules: [

      // rules for modules
      (configure loaders, parser options,
      etc.)
      {test: /\.css$/, use:
'raw-loader'}

    ]
  }
}
```

1. Essa [regra](#) (rules) diz ao compilador do webpack para quando for encontrado um arquivo '.css' dentro de uma declaração [require](#) ou [import](#), deve se usar o [css-loader](#) para transformá-lo antes de adicioná-lo ao pacote.

1. Na instrução [rules : \[\]](#) temos duas propriedades obrigatórias quais sejam:

1. [test](#) : Usa-se a expressão regular para encontrar todos os arquivo **'css'** da pasta.
2. [use](#) : Usar o pacote css-loader para carregá-los.

#### 4. [Plugins](#)

/

1. Os **plugins** servem para executar uma variedade de tarefas como otimização de pacotes, gerenciamento de assets e injeção de variáveis de ambiente. Plugins são definidos em duas partes no arquivo de configuração, no topo do arquivo para plugins externos e na seção plugins dentro da seção module:

1. Código javascript do arquivo

#### **webpack.config.js**

```
// instalado via npm site:
https://webpack.js.org/plugins/html-
webpack-plugin/
const HtmlWebpackPlugin =
require('html-webpack-plugin');
//plugins internos
const webpack =
require('webpack');
const config = {
  module: {
    rules: [
      {test: /\.css$/, use:
'css-loader'}
    ]
  },
  plugins: [
    new
HtmlWebpackPlugin({template:
'./src/index.html'})
  ]
};
module.exports = config;
```

1. No código acima, o plugin **html-webpack** gera um arquivo html e faz a injeção automática do pacote gerado. O webpack possui diversos plugins internos, [confira a lista](#).

#### 5. Mode

1. No atributo mode é definido o mode de execução do webpack como <production>, <development> ou <none>. De acordo com a opção definida o webpack ativa otimizações específicas. O ambiente padrão é produção:

1. Código javascript do arquivo

#### **webpack.config.js**

```
module.exports = {
  mode: 'production'
}
```

## 6. Browser Compatibility

1. webpack é compatível com todos os navegadores compatíveis com ES5 (IE8 e versões anteriores não são compatíveis). webpack precisa [Promise](#) para [import\(\)](#) e [require.ensure\(\)](#) . Se quiser oferecer suporte a navegadores mais antigos, você precisará carregar um [polyfill](#) antes de usar essas expressões.

### 5. O webpack só entende arquivos JavaScript e JSON.

1. Os [carregadores \(loaders\)](#) permitem que o webpack processe outros tipos de arquivos e os converta em módulos válidos que possam ser consumidos pelo projeto e adicionados ao gráfico de dependência do projeto.
2. Os carregadores podem transformar arquivos de uma linguagem diferente (como [TypeScript](#) ) em JavaScript ou carregar imagens embutidas como URLs de dados. Veja [conceitos dos carregadores...](#) .

### 3. Usando carregadores:

1. Existem três maneiras de usar carregadores em seu aplicativo:
  1. **Configuração** (recomendado): O ideal é usar o arquivo de configuração [webpack.config.js](#) para customizar webpack porque é mais flexível pelo fato de ser um código javascript e usa a variável global [module.exports](#) para exportar o que for necessário quando a configuração é diferente do padrão webpack.

1. Código JavaScript - exemplo de [configuração simples](#):

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

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

2. Código JavaScript - exemplo de configuração com [múltiplas configurações](#):

```

/
module.exports = [
  {
    output: {
      filename: './dist-
amd.js',
      libraryTarget: 'amd',
    },
    name: 'amd',
    entry: './app.js',
    mode: 'production',
  },
  {
    output: {
      filename: './dist-
commonjs.js',
      libraryTarget:
'commonjs',
    },
    name: 'commonjs',
    entry: './app.js',
    mode: 'production',
  },
];

```

### 3. Código JavaScript - exemplo de configuração **paralelismo**:

```

module.exports = [
  {
    //config-1
  },
  {
    //config-2
  },
];
module.exports.parallelism = 1;

```

## 2. **Inline** : especifique-os explicitamente em cada instrução do comando **import**.

### 1. Código JavaScript exemplo:

```

// Adds the `jquery` to the global
object under the names `$` and
`jQuery`
import $ from "expose-loader?
exposes=$,jQuery!jquery";

// Adds the `lodash/concat` to the
global object under the name

```



```

/
`_.concat`
import { concat } from "expose-
loader?
exposes=_.concat!lodash/concat";

// Adds the `map` and `reduce`
method from `underscore` to the
global object under the name `_.map`
and `_.reduce`
import { 'map,
        reduce,
} from "expose-loader?
exposes=_.map|map,_.reduce|reduce!un
derscore";

```

3. **webpack-cli** : especifique-os em um comando shellscript.

1. Código ShellScript exemplo:

```

webpack --module-bind pug-loader --
module-bind 'css=style-loader!css-
loader'

```

1. Nota: O comando acima usa **pug-loader** para arquivos **.jade** ou **.pug** e **style-loader** e **css-loader** para arquivos **.css**.

6. Os **plug-ins** podem ser aproveitados para realizar uma ampla gama de tarefas, como otimização de pacote, gerenciamento de ativos e injeção de variáveis de ambiente etc... Veja [mais...](#) .

7. Quando estamos criando um projeto é necessário customizarmos o arquivo de configuração **webpack.config.js** para o **modo desenvolvimento**. Veja o tópico [usando webpack-dev-server](#).

8. O aplicativo **webpack-cli** fornece um conjunto flexível de comandos para que os desenvolvedores aumentem a velocidade ao configurar um projeto webpack personalizado. A partir do webpack v4, o webpack não espera um arquivo de configuração, mas frequentemente os desenvolvedores desejam criar uma configuração do webpack mais personalizada com base em seus casos de uso e necessidades. O webpack-cli atende a essas necessidades, fornecendo um conjunto de ferramentas para melhorar a configuração do webpack personalizado.

1. Veja o [guia de instalação](#) para saber como instalar.

2. Para conhecer os comandos do programa **webpack-cli** veja o [documento Interface da Linha de comando](#).

5. [Como usar webpack com a linguagem typescript](#).

/

1. A forma mais prática de criar ambiente de desenvolvimento typescript é executar a sequencia abaixo. Dica: a ultima linha executa o comando **npx tsc-init**. O programa **tsc-init** instala o webpack e cria uma configuração básica. [Veja mais...](#)

1. Código ShellScript

```
# Instala a linguagem typescript globalmente se
não tiver instalado.
sudo npm -g install typescript

# para saber se foi instalado checar a versão
tsc -v

# Instala pacote tsc-init globalmente
npm install -g tsc-init

# Cria arquivo package.json
npm init

# Instala e configura tudo que é necessário na
pasta local.
npx tsc-init
```

2. O comando **npx tsc-init** instala os seguinte pacotes:

1. "webpack": "^5.37.0",

1. O webpack é um compactador javascript e empacotador de módulo estático para aplicativos JavaScript. [Veja mais...](#)

2. "webpack-dev-server": "^3.11.2"

1. O webpack-dev-server fornece um servidor web simples e a capacidade de recarregar ao vivo para evitar que a cada alteração do código original o programador tenha que publicar para ver se funcionou as alterações.

3. "@types/jasmine": "^3.7.2",

1. [Este pacote contém definições de tipo para Jasmine.](#)

4. "jasmine-core": "^3.7.1",

1. [Jasmine](#) é uma estrutura de desenvolvimento orientada por comportamento para testar o código JavaScript independente de navegadores.

5. "karma": "^6.3.2",

1. [Karma](#) é um test runner feito para o AngularJs. O principal objetivo do Karma é automatizar os testes

/

em diversos navegadores web com um único comando.

6. "karma-chrome-launcher": "^3.1.0",

1. Pacote karma para o navegador chrome. [Veja mais...](#)

7. "karma-jasmine": "^4.0.1",

1. Adaptador para a estrutura de teste Jasmine. [Veja mais...](#)

8. "karma-webpack": "^5.0.0",

1. Use o webpack para pré-processar arquivos no karma. [Veja mais...](#)

9. "ts-loader": "^9.1.2", 2. Este é o carregador TypeScript para webpack. [Veja mais...](#)

3. Para que o projeto seja editado e executado no vscode adicione no registro "**compilerOptions**" do arquivo **tsconfig.json** as propriedades **outDir** (diretório de saída para arquivos **.js**) e a propriedade **sourceMap** (Permite compilar e depurar com vscode ):

1. Código json do arquivo **tsconfig.json**:

```
{
  "compilerOptions" :
  {
    "outDir": "./src/js", //Pasta de saída
    dos arquivos .js
    "sourceMap": true, //Permite compilar e
    depurar com vscode
                                //Gera o arquivo
    '.map' correspondente
  }
}
```

## 2. Instalação e configuração local

### 1. Instalar versão mais recente do **webpack**:

1. Criar a pasta e o arquivo de configuração **package.json** do projeto.

1. Código ShellScript:

```
mkdir webpack-demo
npm init -y
```

1. O comando `npm init -y` criou o arquivo de configuração **package.json**.

```
{
  "name": "webpack-demo",
  "version": "1.0.0",
  "description": "",
```

2021-05-03

```

/
    "main": "index.js",
    "scripts": {
      "test": "echo \"Error: no test specified\" && exit 1"
    },
    "author": "",
    "license": "ISC"
  }

```

2. Instalar **webpack** na pasta criada no passo 1:

1. Código ShellScript para instalação:

```
npm install --save-dev webpack
```

1. O comando **npm install --save-dev webpack** adicionou no arquivo **package.json** a chaves: "devDependencies": {"webpack": "^5.36.1"}.

```

{
  "name": "webpack-demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^5.36.1"
  }
}

```

2. [ BACK]

3. Instalar plugins html-webpack-plugin;

1. O plugins **"html-webpack-plugin"** para webpack simplifica a criação de arquivos HTML para servir seus pacotes webpack. Isso é especialmente útil para pacotes webpack que incluem um hash no nome do arquivo, que muda a cada compilação. Você pode deixar o plugin gerar um arquivo HTML para você, fornecer seu próprio modelo usando lodash modelos ou usar seu próprio carregador.

2. Código shellscript

```

# Instala plugin a ser usado por
alterWebpackConfig()
npm install npm install --save-dev html-

```

2021-05-03

3. [ BACK]

4. .

## 2. Instalar a versão mais recente do [webpack-cli](#)

1. Para a versão 4 ou posterior do webpack é preciso instalar o aplicativo **webpack-cli** para executar o webpack partir da linha de comando.

1. Código ShellScript para instalação:

```
npm install --save-dev webpack-cli
```

1. O comando **npm install --save-dev webpack-cli** adicionou no arquivo **package.json** a chaves: "webpack-cli": "^4.6.0".

```
{
  "name": "webpack-demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^5.36.1",
    "webpack-cli": "^4.6.0"
  }
}
```

2. [ BACK]

## 3. Instalar **webpack-cli init**

1. Este pacote contém a lógica para criar uma nova configuração do webpack. Veja [mais...](#)

1. Código ShellScript

```
npm i -D webpack-cli @webpack-cli/init
```

2. [ BACK]

## 4. Instalar **webpack-cli generators**

/

1. Este pacote contém todos os [geradores yeoman](#) relacionados ao webpack-cli..
2. Veja dica do site [medium.com](#) para entender do que se trata.
3. Código ShellScript

```
npm i -D webpack-cli @webpack-cli/generators
```

4. [ BACK]

### 3. **Configuração do webpack:**

#### 1. **Configuração padrão:**

1. O webpack não exige que você use um arquivo de configuração. No entanto, ele assumirá que o ponto de entrada de seu projeto seja **src/index.js** e produzirá o resultado **dist/main.js** reduzido e otimizado para produção.
2. Para testar se seu projeto está funcionando execute a linha de comando abaixo:

1. Código ShellScript

```
./node_modules/.bin/webpack
```

2. Este comando produzirá a seguinte mensagem:

1. asset main.js 25 bytes [compared for emit] [minimized] (name: main):
2. ./src/index.js 25 bytes [built] [code generated]
3. AVISO na configuração A opção 'mode' não foi definida, o webpack irá retornar para 'production' para este valor. Defina a opção 'mode' como 'development' ou 'production' para habilitar os padrões para cada ambiente. Você também pode defini-lo como 'none' para desativar qualquer comportamento padrão. Saiba mais: <https://webpack.js.org/configuration/mode/>

3. Se a opção 'mode' não for definido, webpack define 'mode' para 'production' como o valor padrão.

3. [ BACK]

#### 2. **Configuração Personalizada.**

1. Quando você precisar alterar a configuração padrão você pode criar o arquivo **webpack.config.js** na pasta raiz e o webpack irá usá-lo automaticamente para satisfazer sua preferência.
2. Caso seja necessário o nome do arquivo de configuração diferente de **webpack.config.js** é possível informar no arquivo

/  
**package.json** na propriedade **script{}** como no exemplo abaixo:

1. Código json:

```
"scripts": {  
  "build": "webpack --config  
production.config.js"  
}
```

3. A maneira mais prática para criar o arquivo de configuração **webpack.config.js** personalizada é executar o comando abaixo:

1. Código ShellScript:

```
npx webpack-cli init
```

1. Este comando faz as seguintes tarefas:

1. Cria o arquivo **webpack.config.js**
2. Criar um arquivo `./index.html`
3. Cria a pasta `./src`
4. Cria um arquivo `./src/index.js`
5. ..

4. [ BACK]

## 2. Instalar **nanoid**

1. Um gerador de ID de string minúsculo, seguro, compatível com URL e exclusivo para JavaScript.
2. Obs: Se não instalar **nanoid** o comando **npx webpack init** gerar um erro.
3. Código ShellScript

```
npm install --save-dev nanoid  
npm install --save-dev @types/nanoid
```

4. [ BACK]

## 4. Instalar a versão mais recente do **webpack-dev-server**:

1. O **webpack-dev-server** fornece um servidor web simples e a capacidade de recarregar ao vivo para evitar que a cada alteração do código original o programador tenha que publicar para ver se funcionou as alterações.
2. Código ShellScript para instalação:

```
npm install --save-dev webpack-dev-server
```

/

1. O comando **npm install --save-dev webpack-dev-server** adicionou no arquivo **package.json** a chaves: "webpack-dev-server": "^3.11.2"

```
{
  "name": "webpack-demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^5.36.1",
    "webpack-cli": "^4.6.0",
    "webpack-dev-server": "^3.11.2"
  }
}
```

2. [

3. ..

5. item 02.

6. [

### 3. Exemplos.

1. webpack Getting Started.

1. [Basic Setup](#).

1. Primeiro, vamos criar um diretório, inicializar o npm, instalar o webpack localmente e instalar o webpack-cli(a ferramenta usada para executar o webpack na linha de comando):

1. Código ShellScript

```
mkdir basic_setup
cd basic_setup
npm init -y
npm install webpack webpack-cli lodash --save-dev
```

2. Segundo vamos criar o arquivo **./index.html**, a pasta **./src** e o arquivo **./src/index.js**.

3. Adicionar código abaixo no arquivo **./src/index.js**

1. Código javascript do arquivo **./src/index.js** :



```

/
import _ from 'lodash';
function component() {
  const element =
document.createElement('div');
  /**
   * Lodash (https://lodash.com/),
   * atualmente incluído por meio de um script,
   * é necessário para que esta linha
   * funcione porque tem a function "_.join()".
   *
   * Lodash é uma biblioteca de utilitários
   * JavaScript
   * moderna que oferece modularidade,
   * desempenho e extras.
   */
  element.innerHTML = _.join(['Alo',
'Mundo'], ' ');
  return element;
}

document.body.appendChild(component());

```

4. Adicionar código abaixo no arquivo ./index.html

1. Código html do arquivo **./index.html**:

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Getting Started</title>
  <!--
    Lodash (https://lodash.com/) é uma
    biblioteca de utilitários JavaScript
    moderna que oferece modularidade,
    desempenho e extras.
  -->
  <!-- <script
src="https://unpkg.com/lodash@4.17.20">
</script> -->
</head>
<body>
  <script src="./src/index.js"></script>
</body>
</html>

```

5. Agora é preciso ajustar o arquivo **package.json** adicionando a propriedade **"private" : true** e removendo a propriedade **"main": "index.js"**. Isso evita uma publicação acidental do seu código.

1. Código json do arquivo **./package.json**:

```

/
{
  "name": "basic_setup",
  "version": "1.0.0",
  "description": "",
  "private": true,
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
    "build" : "webpack --mode='production'",
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^5.36.2",
    "webpack-cli": "^4.6.0"
  }
}

```

6. Último passo é preciso seguir os seguintes passos:

1. Executar comando **npx webpack** que tomará nosso script `./src/index.js` como ponto de entrada e gerará `./dist/main.js` como saída. O npx executa o binário webpack (`./node_modules/.bin/webpack`) do pacote webpack que instalamos no início:

#### 1. Código ShellScript

```

/**
  O comando abaixo criar a pasta
  ./dist e
  compacta o código javascript
  ./src/index.js
  para a pasta ./dist
  */
npx webpack

```

2. Mova o arquivo `index.html` para a pasta `./dist`;
3. No arquivo `./dist/index.html` troque a linha `<script src="./src/index.js"> </script>` para `<script src="main.js"></script>`.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Getting Started</title>
  <!--
  Lodash é uma biblioteca de utilitários
  2021-05-03

```

```

/
JavaScript
  moderna que oferece modularidade,
  desempenho e extras.
  -->
  <!-- <script
src="https://unpkg.com/lodash@4.17.20">
</script> -->
</head>
<body>
  <!-- <script src="./src/index.js"> -->
  <script src="main.js"></script>
</body>
</html>

```

4. Para testar execute o arquivo **./dist/index.html** no browser.

5. []

## 2. Exemplo de webpack com a linguagem javascript

1. Criar o projeto na pasta selecionada:

1. Código ShellScript

```

npm init -y

npm install --save-dev @babel/core @babel/cli
npm install --save-dev @babel/plugin-transform-
block-scoping

npm install --save-dev webpack webpack-cli
npm i -D webpack-cli @webpack-cli/generators
npm install --save-dev webpack-dev-server

npm install --save-dev nanoid
npm install --save-dev @types/nanoid
npm install --save lodash

// Veja nota ...2.1 abaixo para detalhes do
comando a seguir.
npx webpack-cli init

```

2. Notas:

1. O comando **npx webpack-cli init** acima informa que existe um conflito. Motivo: O arquivo **package.json** foi criado no primeiro comando. Diga **não** para que mantenha o **package.json** atual.
2. Após criar o projeto siga os passos abaixo para construção do exemplo 01:
  1. Selecionar o arquivo **./src/index.js** e edite o seguinte código:
  1. Código javascript:

/

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

  // Lodash, currently included via a
  // script, is required for this line to work
  element.innerHTML = _.join(['Hello',
'webpack'], ' ');

  return element;
}

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

2. Selecionar o arquivo **./index.html** e edite o seguinte código:

1. Código html:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Getting Started</title>
  <script
src="https://unpkg.com/lodash@4.17.20">
</script>
</head>
<body>
  <script src="./src/index.js"></script>
</body>
</html>
```

3. Também precisamos ajustar nosso **package.json** arquivo para ter certeza de marcar nosso pacote como private, bem como remover a entrada **main**. Isso evita uma publicação acidental do seu código.

3. [ BACK]

### 3. Exemplo de webpack com a linguagem typescript:

1. Criar o projeto na pasta selecionada:

1. Código ShellScript

1. [Shellscript para criar projetos typescript/my-tsc-init-ver-0.2.0.sh](#)

2. [Veja mais...](#)

2. .

2. [ BACK]

## 4. REFERÊNCIAS

1. [Guia de instalação oficial](#)
2. [Enhance main window](#)
3. [webpack](#)
4. [Vídeo para iniciante sobre webpack](#)
5. [webpack-2-para-iniciantes-o-que-e-porque-usar-e-como-iniciar.](#)
6. [postcss aumentar a legibilidade do código css](#)
7. [nanoid](#)
8. [configuration/mode](#)
9. [O que é babel js](#)
10. [Letra da música do programa babel js](#)
11. [Música do programa bebel js](#)
12. [Conceitos do webpack](#)
13. [Generate Webpack Configuration file \(webpack.config.js\) using webpack-cli](#)
14. [dev-server-configuration](#)
15. [Webpack & TypeScript](#)
16. [html-webpack-plugin](#)
17. [Babel e Webpack que roda em seus Projetos JavaScript?](#)
18. [webpack // Dicionário do Programador](#)
19. [html-loader](#)
20. [html-webpack-plugin](#)
21. [html-webpack-template](#)
22. <#>
23. [\[#\]\(link](#)
24. [\[#\]\(link](#)
25. [\[← BACK\]](#)

## 5. HISTÓRICO

1. 27/04/2021
  - ☒ Criar este documento baseado no [modelo02.md](#) ;
    - [\[← BACK\]](#)
2. 28/04/2021
  - ☒ Escrever tópico Objetivos;
  - ☒ Escrever tópico Pre-requisitos
  - ☒ Escrever tópico Benefícios

- []

### 3. 29/04/2021

- ✓ Criar tópico instalar a versão mais recente do webpack.
- ✓ Criar tópico Instalar a versão mais recente do webpack-cli.
- ✓ Criar tópico Instalar a versão mais recente do webpack-dev-server.

- []

### 4. 30/04/2021

- ✓ Escrever tópico configuração do webpack.
- ✓ Escrever tópico Exemplos - falta executar javascript dentro do arquivo ./index.html

### 5. 05/05/2021

- ✓ Escrever tópico Exemplo básico de uso do webpack.

### 6. 06/05/2021

- ✓ Escrever tópico descrição do webpack parte 01.

### 7. 08/05 a 11/05/2021

- ✓ Escrever tópico descrição do webpack.

### 8. 27/05/2021

- ✓ Criar documento: Como usar webpack com a linguagem typescript.
- ✓ Escrever tópico Referências
- ✓ Atualizar o histórico deste documento.

- [ ]

### 9. 15/06/2021

- ✓ Depois de tanta luta para entender webpack, achei o projeto webpack-demo com 15 exemplo. Meu erro foi achar que ele pudesse pegar meu projeto atual e transformar em site de distribuição na web. O objetivo do webpack é empacotar javascript apenas.

### 10. xx/06/2021

- ☐ Testar este documento depois que eu esquecer dele.

- []

