Prática da Video aula Webpack & TypeScript ↵℘

0. Play Lista da Video aula Webpack & TypeScript

1. Aula 01 - Setup - Introduction:

    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
```

2. Aula 02 - Setup - Webpack Installation:

    1. Código shellscript

```
    # Criar pasta para o projeto:
        mkdir ./webpack-typescript

    # Move-se para a pasta ./webpack-typescript
        cd ./webpack-typescript

    # Cria o arquivo tsconfig.json na pasta ./webpack-typescript
        tsc --init

    # Entre no editor de texto sua preferência e edit o arquivo
tsconfig.json:
        xed ./tsconfig.json

    # Troque as propriedades:
        # "target" : "es6" - Trocar a propriedade "target" de
"es5" para "es6".
        # "module" : "es2015" - Trocar a propriedade "module" de
"commonjs" para "es2015"

        # Salve as alterações;
        # Saia do editor de texto.

    # Cria o arquivo package.json
        #  - O comando abaixo cria o arquivo package.json para
salvar as configurações do projeto tais como dependências, nome
do projeto, nome dos script de execução do projeto,  versão do
projeto, descrição do projeto, autor, e licença;
```

```
    npm init -y # O parâmetro -y diz para confirmar tudo.

    # Instalar pacotes: webpack, webpack-cli e ts-loader com a
opção -D (-D é equivalente a --save-dev = Salve em:
devDependencies": {} do arquivo package.json)

        #   1 - Cria a pasta node_modules para os modulos locais
do projeto e cria;

        #   2 - Cria o arquivo package-lock.json usado para
bloqueio dos pacotes quando precisar;
    npm install webpack webpack-cli ts-loader -D

    # Instalar typescript na pasta local para que seja uma
dependência de desenvolvimento:
    npm install typescript -D

    # Criar a pasta ./src para salvarmos os fontes do projeto.
    mkdir ./src

    # Criar pasta ./public para salvar os arquivos de saida do
webpack.
        # Nesta pasta conterá os arquivo .css, .html, .js após a
execução de webpack.
        # Esta é a pasta que deve ser publicada no servidor web.
    mkdir ./public

    # Criar o arquivo index.html
    touch ./public/index.html
```

2. Com editor de sua preferência adicione o código abaixo no arquivo ./public/index.html:

    1. Código html

```
    <!DOCTYPE html>
    <html dir="ltr" lang="pt-br">
        <head>
            <meta http-equiv="content-type"
 content="text/html; charset=utf-8" />

            <meta name="viewport" content="width=device-
 width, initial-scale=1.0" />

            <title>Exemplo de webpack & typescript</title>

            <meta name="createDate" content="20/05/2021" />
            <meta name="createDateUpdate"
 content="20/05/2021" />
            <meta name="description" content="Exemplo
 prático de um pacote gerador pelo webpack" />
```

```
            <meta name="keywords" content="webpack,
typescript, javascript" />

            <link type="text/css"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css"
                rel="stylesheet" />
        </head>
        <body>
            <p>Formulário teste webpack</p>

            <form action="endpoint" method="get">
                <fieldset>
                <label for="nome">Nome:? </label> <input
type="text" id="nome" required>
                <label for="idade">Idade:? </label> <input
type="number" id="idade" required>
                </fieldset>
                <fieldset>
                <label for="email">Email:? </label> <input
type="text" id="email" required>
                </fieldset>
                <fieldset>
                <button type="submit">Enviar formulário?
</button>
                </fieldset>

            </form>

        </body>
    </html>
```

3. Criar o arquivo ./src/index.ts:

    1. Código shellscript

```
    # Cria o arquivo ./src/index.ts
    touch ./src/index.ts
```

3. Aula 03 - Webpack & TypeScript Setup - Webpack Config File

    1. Criar o arquivo javascript: **webpack-config.js**:

        1. Código shellscript

```
   touch ./webpack-config.js
```

2. No arquivo **webpack-config.js** adicione o código javascript abaixo para customizar a compilação do webpack segundo nossa necessidade:

    1. Os parâmetros que o webpack espera são objetos do nodejs passado pelo objeto **module.exports**.

        1. Código javascript

```
    /**
      * path é uma biblioteca NodeJs padrão que está
globalmente disponível quando você instala o NodeJs.
      * A linhas a baixo 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/)
     * Em module é a opções que determina como os
diferentes tipos de módulos em um projeto serão
tratados.
     * [Veja mais sobre module...]
(https://webpack.js.org/configuration/module/)
     */
    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/index.ts',
        /**
          * A propriedade output é um objeto com as
seguintes informações:
            * filename = Nome do arquivo do pacote a ser
gerado.
            * path = Nome absolute da pasta onde o
pacote será gerado. Obs: Em caso
            * de omissão do mesmo o webpack criar a
```

```
pasta ./dist.
          */
        output : {
            /**
              * Nome do arquivo para o pacote a ser
gerado na pasta ./public
              */
            filename : 'bundle.js',

            /**
              * Caminho absoluto da pasta em que será
gerado o pacote que no nosso
              * caso é a pasta ./public.
              * [Veja mais sobre path.resolve ...)
(https://nodejs.org/api/path).html#path_path_resolve_pat
hs
              */
            path: path.resolve(__dirname,'public'),
            /**
              * __dirname : - Esta é uma variável global
NodeJs que fornece o caminho
              * do arquivo atualmente em execução. Nesse
caso, ele fornece o caminho
              * do webpack.config.js . (
webpack.config.js é o arquivo atualmente em execução)
              *
            */
        },

        /**
         * resolve:{}
         * Essas opções mudam a forma como os módulos
são resolvidos. webpack fornece
         * padrões razoáveis, mas é possível alterar a
resolução em detalhes. Dê uma
         * olhada em Resolução do Módulo para obter mais
explicações sobre como o
         * resolvedor funciona.
         * veja mais:
https://webpack.js.org/configuration/resolve/
         *
         * resolve.extensions
         * [string] = ['.ts', '.tsx','js', '.wasm']
         * Tente resolver essas extensões em ordem. Se
vários arquivos compartilham o
         * mesmo nome, mas têm extensões diferentes, o
webpack resolverá aquele com a
         * extensão listada primeiro na matriz e
ignorará o resto.
         */
        resolve: {
          extensions: ['.ts', '.tsx', '.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.
          */
         module: {
          // O array rules são usado para passagem de
regras na transpilação dos módulos:
          rules: [
            // rules for modules (configure loaders,
parser options, etc.)
            {

              /**
               * A expressão na lista de test: informa
ao webpack que pegue todos os arquivos
               * com extensão é ts e o $ indica fim da
expressão.
               */
              test : /.tsx?$/,

              /**
               * O comando ts-loader receber arquivos
\.ts (TypeScript) e gera saída arquivos .js
(javascript).
               *
               */
              use : 'ts-loader', //loader ou use faz o
mesmo efeito .

              /**
               * A propriedade include informa a pasta
no qual devemos pegar os arquivo \.ts
              */
              include : [path.resolve(__dirname,'src')]
            }
          ]
        },

    }
```

2. .

2. Com editor de sua preferência adicionar no arquivo ./src/index.ts o seguinte código:

    1. Código javascript

```
   console.log('Alo mundo');
```

3. Com editor de texto de sua preferência adicione o propriedade **"build" : "webpack"** no objecto **scripts{}** do arquivo package.json

    1. Código json.

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

      "scripts": {

        "build" : "webpack",

        "test": "echo \"Error: no test specified\" &&
exit 1"
      },
      "keywords": [],
      "author": "Paulo Pacheco",
      "license": "ISC",
      "devDependencies": {
        "ts-loader": "^9.2.1",
        "typescript": "^4.2.4",
        "webpack": "^5.37.1",
        "webpack-cli": "^4.7.0"
      }
    }
```

3. Para gerar o pacote na pasta ./public entrar na pasta raiz do projeto **./webpack-typescript** e execute o seguinte comando:

    1. Código ShellScript.

```
npm run build
```

4. ..

4. Aula 04 - Webpack & TypeScript Setup - Webpack Dev Server

    1. Instalar o servidor de desenvolvimento local com a opção -D = --save-dev.

        1. Código shellscript

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

2. Com editor de texto de sua preferência adicione o propriedade **"dev": "webpack serve --mode development --env development --hot --port 3000"** no objecto **scripts{}** do arquivo package.json

    1. Código json.

```
    {
       "name": "webpack-typescript",
       "version": "1.0.0",
       "description": "",
       "scripts": {
         "test": "karma start",

         "dev": "webpack serve --mode development --env
development --hot --port 3000",

         "build": "webpack",
       },
       "author": "",
       "license": "ISC",
       "devDependencies": {
         "@types/jasmine": "^3.7.4",
         "jasmine-core": "^3.7.1",
         "karma": "^6.3.2",
         "karma-chrome-launcher": "^3.1.0",
         "karma-jasmine": "^4.0.1",
         "karma-webpack": "^5.0.0",
         "ts-loader": "^9.2.2",
         "typescript": "^4.2.4",
         "webpack": "^5.37.1",
         "webpack-cli": "^4.7.0",
         "webpack-dev-server": "^3.11.2"
       }
    }
```

3. Adicionar o script **<script src='bundle.js'> </script>** no arquivo **./public/index.html** logo após a tag </form>.

    1. Código html

```
    <!DOCTYPE html>
    <html dir="ltr" lang="pt-br">
        <head>
            <meta http-equiv="content-type"
content="text/html; charset=utf-8" />
```

```
            <meta name="viewport" content="width=device-
width, initial-scale=1.0" />

            <title>Exemplo de webpack &
typescript</title>

            <meta name="createDate" content="20/05/2021"
/>
            <meta name="createDateUpdate"
content="20/05/2021" />
            <meta name="description" content="Exemplo
prático de um pacote gerador pelo webpack" />
            <meta name="keywords" content="webpack,
typescript, javascript" />

            <link type="text/css"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css"
                rel="stylesheet" />
        </head>
        <body>
            <p>Formulário teste webpack</p>

            <form action="endpoint" method="get">
                <fieldset>
                <label for="nome">Nome:? </label> <input
type="text" id="nome" required>
                <label for="idade">Idade:? </label>
<input type="number" id="idade" required>
                </fieldset>
                <fieldset>
                <label for="email">Email:? </label>
<input type="text" id="email" required>
                </fieldset>
                <fieldset>
                <button type="submit">Enviar formulário?
</button>
                </fieldset>

            </form>

            <script src='bundle.js'> </script>
        </body>
    </html>
```

2. Para testar execute o comando:

    1. Código shellScript

```
    npm run dev
```

2. ...

2. Para que o **webpack-dev-server** compile automáticamente toda vês que se faz um alteração nos códigos, é necessário adiciona no objeto **output:{}** do arquivo **webpack.config.js** a propriedade **publicPath : 'public'**.

    1. Código javascript

```
   /**
    * path é uma biblioteca NodeJs padrão que está
globalmente disponível quando você instala o NodeJs.
    * A linhas a baixo 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/)
   * Em module é a opções que determina como os diferentes
tipos de módulos em um projeto serão tratados.
   * [Veja mais sobre module...]
(https://webpack.js.org/configuration/module/)
   */
   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/index.ts',
       /**
        * A propriedade output é um objeto com as seguintes
informações:
          * filename = Nome do arquivo do pacote a ser
gerado.
          * path = Nome absolute da pasta onde o pacote
```

```
será gerado. Obs: Em caso
           * de omissão do mesmo o webpack criar a pasta
./dist.
         */
       output : {
           /**
            * Informa ao webpack-dev-server para compilar
para a pasta ./public toda vez que
            * um código for alterado.
           */
           publicPath : 'public',

           /**
            * Nome do arquivo para o pacote a ser gerado na
pasta ./public
             */
           filename : 'bundle.js',

           /**
            * Caminho absoluto da pasta em que será gerado o
pacote que no nosso
             * caso é a pasta ./public.
             * [Veja mais sobre path.resolve ...)
(https://nodejs.org/api/path).html#path_path_resolve_paths
             */
           path: path.resolve(__dirname,'public'),
           /**
            * __dirname : - Esta é uma variável global
NodeJs que fornece o caminho
             * do arquivo atualmente em execução. Nesse
caso, ele fornece o caminho
             * do webpack.config.js . ( webpack.config.js é
o arquivo atualmente em execução)
             *
           */
       },

       /**
       * resolve:{}
       * Essas opções mudam a forma como os módulos são
resolvidos. webpack fornece
       * padrões razoáveis, mas é possível alterar a
resolução em detalhes. Dê uma
       * olhada em Resolução do Módulo para obter mais
explicações sobre como o
       * resolvedor funciona.
       * veja mais:
https://webpack.js.org/configuration/resolve/
       *
       * resolve.extensions
       * [string] = ['.ts', '.tsx','js', '.wasm']
       * Tente resolver essas extensões em ordem. Se vários
arquivos compartilham o
       * mesmo nome, mas têm extensões diferentes, o webpack
```

```
resolverá aquele com a
       * extensão listada primeiro na matriz e ignorará o
resto.
       */
       resolve: {
         extensions: ['.ts', '.tsx', '.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.
         */
       module: {
         // O array rules são usado para passagem de regras
na transpilação dos módulos:
         rules: [
           // rules for modules (configure loaders, parser
options, etc.)
           {

             /**
             * A expressão na lista de test: informa ao
webpack que pegue todos os arquivos
             * com extensão é ts e o $ indica fim da
expressão.
             */
             test : /.tsx?$/,

             /**
             * O comando ts-loader receber arquivos \.ts
(TypeScript) e gera saída arquivos .js (javascript).
             *
             */
             use : 'ts-loader', //loader ou use faz o mesmo
efeito .

             /**
             * A propriedade include informa a pasta no qual
devemos pegar os arquivo \.ts
             */
             include : [path.resolve(__dirname,'src')]
           }
         ]
       },

   }
```

2. ...

3. .

5. Aula 5 - Webpack & TypeScript Setup - Using ES6 Modules

    1. Criar arquivo forms.ts para lidar com envio dos formulários do projeto.

        1. Código shellscript para criar o arquivo ./src/forms.ts

```
touch ./src/forms.ts
```

        2. Execute o editor de textos de sua preferência e adicione o código abaixo em ./src/forms.ts.

            1. Código TypeScript

```
/**
 * Vaja a Documentação de [HTMLFormElement]
(https://developer.mozilla.org/en-
US/docs/Web/API/HTMLFormElement) para mais informações.
 *
 */
export const formData = (form:HTMLFormElement) => {
   const inputs = form.querySelectAll('input');
   let values : {[prop:string]:string} = {};

   inputs.forEach(input => {
      values[input.id] = input.value;
   });
   return values;
};
```

            2. .

        3. .

    2. Execute o editor de textos de sua preferência e adicione o código abaixo em ./src/index.ts..

        1. Código TypeScript

```
console.log('Alo mundo');

import {formData} from './forms';

const form = document.querySelector('form')!;

form.addEventListener(
   'submit',(e) => {  e.preventDefault();
```

```
                        const data = formData(form);
                        console.log(data);
                    }
    );
```

1. OBS: O código acima não funciona.

6. ...