**WEB-PACK 20.2020**

Андлер (статичний модульний складальник) коду, призначений для поєднання (інтеграції) різних модулів коду в єдину систему (Бандл). При цьому можлива додаткова обробка, модифікація та перевірка на помилки.

**Entry -** модуль, який використовується для побудови внутрішнього графа залежностей. З його допомогою webpack визначає, від яких модулів і бібліотек залежить точка входу (безпосередньо і не безпосередньо). Потім включає їх в граф, поки не залишиться жодної залежності. За замовчуванням для властивості entry встановлено значення ./src/index.js. Але можна вказати інший модуль у файлі конфігурації збирача.

**Output** - це властивість вказує, де webpack повинен зберігати бандл і як називати його файл (або файли). Заводський параметр ./dist/main.js для основного бандла і ./dist для інших згенерованих файлів.

**Завантажувачі -** за замовчуванням webpack розуміє тільки файли JavaScript і JSON. Щоб обробити інші типи файлів і конвертувати їх в модулі, збирач використовує завантажувачі. Наприклад, завантажувач може трансформувати файли з мови CoffeeScript в JavaScript або вбудовані зображення в URL-адреси. За допомогою загрузчиков можна навіть імпортувати CSS-файли прямо з модулів JavaScript.

**Плагіни** Використовуються для задач, які не можуть виконувати завантажувачі.

**Режими** Webpack дозволяє налаштувати режим на development, production або none. Завдяки цьому він може використовувати вбудовані оптимізації для кожного середовища. За умовчанням встановлено значення Режим none означає, що всі опції оптимізації за замовчуванням будуть відключені. Щоб дізнатися більше про опції, які webpack використовує в development і production, відвідайте [сторінку конфігурації режимів](https://webpack.js.org/configuration/mode/).

Робочий процес webpack включає в себе настройку файлу index.html з відповідними посиланнями на CSS, JavaScript і необхідні ресурси.

При виконанні завдань webpack спирається на конфігурацію. Вони прописані у файлі **webpack.config.js**. У ньому зазначено, як файли і ресурси слід трансформувати.

Встановити останню рекомендовану версію Node.js

Створити структуру проекту:

Webpack –test-project

**npm init –y**

прапор ***–y*** означає створення дефолтного файлу package.json

**npm install --save-dev webpack webpack-cli**

У файлі **package.json** модифікуємо об’єкт **scripts:**

"scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "dev": "webpack --mode development",

    "build": "webpack --mode production"

  },

Це дозволить звертатись до локально встановленного WebPack через скорочені команди:

**webpack** ***--mode*** **development; означає:** запустити пакет **webpack** у режимі **розробки**

Стають доступними команди: ***npm run dev, npm run build, npm run test***

**Можна зробити по іншому:**

У файлі: **webpack.config.js** визначити режим роботи WEBpack:

  mode: "development",

і більше не декларувати при запускі режим:

**npx webpack**

**Кінцевий варіант налаштування запуску:**

1. До **package.json** => **scripts** додаємо**:**

"start": "npx webpack"

1. Додається команда запуску **npm start === npx webpack === *npm run dev***
2. Режим спостереження:

"watch": "webpack --mode development --watch"

Запуск через npm run watch

В /src створюємо файл точку входу: index.js

Запускаємо тестове збирання проекту: **npm run dev (npx webpack)**

Тестуємо режим продакшену:

Створено папку **/dist**  і в неї збандлився проект, а саме файл **main.js**. Створюємо додатково в цій папці файл **index.html**

Встановлюємо додатковий плагін: **npm run build**

**npm install html-webpack-plugin --save-dev**

*Автоматично підставляє потрібний бандл в html – файл, та здатний модифікувати його.*

*Створюємо файл налаштувань WEBPACK:* **webpack.config.js** та вводимо наступне:

const HtmlWebpackPlugin = require("html-webpack-plugin");

const path = require('path');

module.exports = {

  plugins: [

    new HtmlWebpackPlugin({

      title: "Webpack practice",

    }),

  ],

};

Активуємо плагін додаючи його в масив плагінів, задаємо додаткові налаштування

Перевіряємо роботу плагіну через: **npm run build**

У HTML файлі відбудеться заміна тайтлу та автопідстановка потрібного JS бандлу.

Модифікуємо **input** та **output**

Розширюємо налаштування:

module.exports = {

   entry: {

    main: './src/js/jscomponents/app.js',

    libs: []

},

  plugins: [

    new HtmlWebpackPlugin({

      title: "Webpack practice",

    }),

  ],

};

Об’єкт параметрів **entry** приймає основний файл JS з імпортами модулів та окремо бібліотеки. В результаті все буде поєднано в один файл.

Конфігуруємо параметри вихідних данних

  output: {

  filename: '[name].bundle-[hash].js',

  path: path.resolve(\_\_dirname, 'dist')

  },

Додатковий пакет **const path = require('path');** дозволяє коректно працювати зі шляхами до файлів у різних операційних системах.

Аргументи:

**\_\_dirname -**  папка на комп’ютері де здійснюється виклик коду

**‘dist’** - дуга папка для створення повного шляху призначення

У структурі проекту створюємо відповідні файли: **./src/js/jscomponents/app.js**  та **./src/js/jscomponents/component\_one.js**

**component\_one.js**

*export* *default* (text = "Hello webpack") => {

  const element = document.createElement("p");

  element.innerHTML = text;

*return* element;

};

**app.js**

*import* componentOne *from* "./component\_one";

document.body.appendChild(componentOne());

перевіряємо роботу: **npm run build**

Запуск через утиліту npx webpack (npx – утиліта node.js для запуску локальних модулів)

Для ідеального побудування шляхів треба визначити контекст та змінити **entry:**

context: path.resolve(\_\_dirname, 'src'),

entry: {

    main: './js/jscomponents/app.js',

    libs: []

},

**Транспіляція ES6 => ES5**

Використовуємо лоадер **babel-loader**

Встановлюємо: **npm install -D babel-loader @babel/core @babel/preset-env**

Додаємо налаштування лодеру до **webpack.config.js** :

module: {

    rules: [

        {

            test: /\.js*$*/,

            exclude: /node\_modules/,

            use: {

                loader: 'babel-loader',

                options: {

                    presets: ['@babel/preset-env']

                }

            }

        }

    ]

  },

До файлу **package.json** додажмо параметр що визначає цільові браузери

  "browserslist": "> 0.25%, not dead",

npm install --save @babel/polyfill

та модифікуємо запис точки входу для файлів що містять експериментальний функціонал JS

entry: {

    main: ['@babel/polyfill', './js/app.js'],

    libs: ['./js/libs/lib\_1.js']

  },

Якщо вникають додаткові проблеми з транспіляцією, можна розширити бабель за рахунок плагінів наприклад:

**npm i -D @babel/plugin-proposal-class-properties**

та додати в **webpack.config.js:**

{

        test: /\.js*$*/,

        exclude: /node\_modules/,

        use: {

          loader: 'babel-loader',

          options: {

            presets: ['@babel/preset-env'],

**plugins: ['@babel/plugin-proposal-class-properties']**

          }

        }

      },

**Робота з JSON**

1. Створити потрібний файл у форматі json,
2. У головному файлі JS прописати

*import* json *from* "../JSON/test"

**Робота із зображеннями**

1. Створити файл зображення
2. npm install file-loader --save-dev
3. У головному JS:

*import* SomeImage *from* "../../assets/images/DesktopWallpaper.jpg"

1. Тепер можна сміливо підключати картинки через CSS

**Робота із шрифтами**

1. Скачати потрібний файл шрифтів
2. Створюємо окремий файл css для підключення шрифтів fonts.css

*@font-face* {

  font-family: "Roboto";

  src: url("../assets/fonts/roboto/Roboto.ttf") format('truetype');

}

1. У осовному файлі style.css:

*/\* Import of fonts \*/*

*@import* "./fonts.css";

**npm run watch**

**Підключення CSS бібліотек**

1. <https://necolas.github.io/normalize.css/> npm install normalize.css
2. У **styles.css:**

**Плагіни**

<https://webpack.js.org/plugins/>

**UglifyjsWebpackPlugin** мініфікація JS

**npm install uglifyjs-webpack-plugin --save-dev**

const UglifyJsPlugin = require('uglifyjs-webpack-plugin');

optimization: {

    minimizer: [new UglifyJsPlugin({

      test: /\.js*$*/,

      parallel: true,

    })],

  },

**HTML Webpack Plugin (** <https://github.com/jantimon/html-webpack-plugin#options> **)**

**npm i --save-dev html-webpack-plugin**

plugins: [

    new HtmlWebpackPlugin({

      filename: 'index.html',

      template: './index.html',

      title: 'Home'

    }),

]

У розмітці потрібної сторінки HTML потрібно використати спеціальний синтаксис вставки даних про заголовок з WEBPACK

Підключення стилів через WEBPACK:

**npm install -D css-loader style-loader**

**module => rules :**

{

test: /\.css*$*/,

use: [

{ loader: 'style-loader'},

            { loader: 'css-loader'}

          ]

}

У головному файлі скриптів прописати імпорт стилів:

require('./../css/style.css');

або

*import* css *from* './../css/style.css';

Перевірити запустивши збірку проекту: **npm start**

**./dist/index.html** при відкритті в браузері стилі з вихідної папки будуть підключені через тег <style></style>

Для виокремлення стилів в окремий файл потрібен

**MiniCssExtractPlugin (**<https://webpack.js.org/plugins/mini-css-extract-plugin/>**)**

**npm install -D mini-css-extract-plugin**

plugins: [

new MiniCssExtractPlugin( { filename: 'style.css'} ),

 ],

**Hot Module Reloading (HMR)**

Треба встановити додатково **npm i -D cross-env**

**Та стврити глобальну змінну**

const isDevelopment = process.env.NODE\_ENV === 'development';

{

        test: /\.css*$*/,

        use: [

            {loader: MiniCssExtractPlugin.loader,

            options: {

              hmr: isDevelopment,

              reloadAll: true

              }

            },

            'css-loader'

          ],

      },

Змінити запуски на:

"scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "dev": "cross-env NODE\_ENV=development webpack --mode development",

    "build": "cross-env NODE\_ENV=production webpack --mode production",

    "start": "npx webpack",

    "watch": "cross-env NODE\_ENV=development webpack --mode development --watch",

    "serve": "cross-env NODE\_ENV=development webpack-dev-server --compress --mode development --open"

  },

Очистка папки ./dist перед бандлом

**npm install --save-dev clean-webpack-plugin**

const { CleanWebpackPlugin } = require('clean-webpack-plugin');

plugins: [

    new CleanWebpackPlugin(),

**Розширення форматів що розуміє WEBPACK**

Формати прописуються через **extensions**

resolve: {

    extensions: ['.js', '.json', '.jpg', '.png',],

    alias: {

      '@components': path.resolve(\_\_dirname, './src/js/jscomponents'),

      '@': path.resolve(\_\_dirname, './src')

    }

  }

**Аліаси**  дозволяють створити всередині проекту власні шляхи (від відносних до абсолютних) до окремих папок, щоб не писати довгий шлях до файлу. Тому тепер можна отримати доступ до папки з JS компонентами через ‘**@components’**

**Підключення сторонніх JS бібліотек**

1. npm install -S jquery
2. Імпортувати до основного JS потрібнубібліотеку

*// Import JS libs*

*import* \* *as* $ *from* 'jquery';

3) Перевірка: створити тег <pre> та прописати скрипт

$('pre').html("Jquery works!");

!!!Але якщо jquery потрібен в багатьох модулях (у двох чи більше точках входу) варто уникнути дублювання коду при бандлі через функцію оптимізації

optimization: {

**splitChunks: {**

**chunks: "all"**

**},**

    minimizer: [

      new UglifyJsPlugin({

        test: /\.js*$*/,

        exclude: /node\_modules/,

        parallel: true,

      })

    ],

  },

**Сервер для девелопменту**

<https://webpack.js.org/configuration/dev-server/>

**npm install –D webpack-dev-server**

**In webpack.config.js**

  devServer: {

    contentBase: './dist',

  },

**In package.json** додативаріант запуску на локальному сервері з спостереженням за змінами вихідних даних

"scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "dev": "webpack --mode development",

    "build": "webpack --mode production",

    "start": "npx webpack",

    "watch": "webpack --mode development --watch",

    "serve": "webpack-dev-server --mode development --open"

  },

**npm run serve –** запуск веб паку через вбудований сервер з відкриттям сторінки в браузері

При цьому готовий бандл завантажується одразу до оперативної пам’яті, для білду в продакшн треба використовувати: **npm run build**

**webpack-dev-server --public 192.168.0.101:8080**

**Копіювання файлів з ./src до папки ./dist**

1) **npm i -D copy-webpack-plugin**

2)У розмітку **index.html** додаємо:

  <link rel="icon" href="./assets/icons/favicon.ico" type="image/ico">

3) webpack.config.js

const CopyPlugin = require('copy-webpack-plugin');

plugins: [

new CopyPlugin([

      { from: path.resolve(\_\_dirname, './src/assets/icons'), to: path.resolve(\_\_dirname, './dist/assets/icons') },

*// { from: 'other', to: 'public' },*

    ]),

]

**Оптимізація збірки в продакшн**

1) Оптимізація **HTML** через **HtmlWebpackPlugin**

new HtmlWebpackPlugin({

      filename: 'index.html',

      template: './index.html',

      title: 'Webpack + scss',

**minify: {**

**collapseWhitespace: !isDevelopment**

**}**

    }),

2) Оптимізація **JS** через **TerserWebpackPlugin**

**npm install terser-webpack-plugin --save-dev**

3) Оптимізація CSS **optimize-css-assets-webpack-plugin**

**npm install --save-dev optimize-css-assets-webpack-plugin**

В **webpack.config.js:**

const TerserJSPlugin = require('terser-webpack-plugin');

const OptimizeCSSAssetsPlugin = require('optimize-css-assets-webpack-plugin');

4) Відключаємо UglifyJsPlugin з **webpack.config.js**  та додаємо функцію управління типу збірки з параметрами застосування оптимізації **CSS & JS:**

function optimization() {

  const optimizationOptions = {

    splitChunks: {

      chunks: "all"

    }

  };

*if* (!isDevelopment) {

    optimizationOptions.minimizer = [

      new TerserJSPlugin({}),

      new OptimizeCSSAssetsPlugin({})

    ]

  }

*return* optimizationOptions

}

Заповнюємо поле оптимізації модулю WEBpack:

optimization: optimization(),

Тепер при запускі бандлу на продакшн всі файли буде оптимізовано що скоротить час завантаження сайту.

**Препроцесинг SCSS**

1) Встановити **npm install sass-loader node-sass**

2) Додати правило **webpack.config.js**

{

        test: /\.scss*$*/,

        use: [

          {

            loader: MiniCssExtractPlugin.loader,

            options: {

              hmr: isDevelopment,

              reloadAll: true

            }

          },

          'css-loader',

          'scss-loader'

        ],

      },

3) Підключити стилі в основний JS файл

*import* '../scss/style.scss';

4) Створити функцію оптимізації імен

function filenameHashConfig(ext) {

*return* isDevelopment ? `[name].${ext}` : `[name]-[hash].${ext}`;

}

Замінити шаблони імен збандленних файлів **webpack.config.js**:

output: {

**filename: filenameHashConfig("js"),**

*// filename: '[name].bundle-[hash].js',*

    path: path.resolve(\_\_dirname, 'dist')

  },

Та

new MiniCssExtractPlugin({ filename: filenameHashConfig("css") }),

**Оптимізація коду конфігурацій**

**webpack.config.js**

function stylesLoadersConfig(mainLoader) {

  const loaders = [

    {

      loader: MiniCssExtractPlugin.loader,

      options: {

        hmr: isDevelopment,

        reloadAll: true

      }

    },

    'css-loader'

  ]

*if*(mainLoader) {

    loaders.push(mainLoader);

  }

*return* loaders;

}

та задаємо параметри в правилах через цю функцію **webpack.config.js**

{

        test: /\.css*$*/,

        use: stylesLoadersConfig();

      },

      {

        test: /\.s[ac]ss*$*/i,

        use: stylesLoadersConfig('sass-loader'),

      },

**Робота з TS**

<https://babeljs.io/docs/en/babel-preset-typescript>

1) Копіюємо правило для JS та модифікуємо під TS **webpack.config.js**

{

        test: /\.ts*$*/,

        exclude: /node\_modules/,

        use: {

          loader: 'babel-loader',

          options: {

            presets: ['@babel/preset-env', '@babel/preset-typescript'],

            plugins: ['@babel/plugin-proposal-class-properties']

          }

        }

      },

2) **npm install --save-dev @babel/preset-typescript**

**Додаткові утиліти розробки**

Source-map у режимі розробки **webpack.config.js**:

devtool: isDevelopment ? 'source-map' : '',

es-lint

npm install eslint-loader --save-dev

….

**Аналіз бандлу веебпака**

webpack-bundle-analyzer

**npm install --save-dev webpack-bundle-analyzer**

**const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin;**

**plugins: [**

**new BundleAnalyzerPlugin()**

**]**