Webpack 4

1. Co to jest webpack

Jest to tak zwany bundler (narzędzie), które czyta wskazany plik JavaScript, a następnie wykonuje na nim zadane czynności.

W wyniku takich akcji nasze źródłowe skrypty są zamieniane na jeden wynikowy, zminimalizowany i zoptymalizowany plik.

2. Pierwsze kroki

Upewnij się, że na swoim komputerze masz zainstalowaną najnowszą, stabilną wersję Node'a - https://nodejs.org/en/download/

2.1 Inicjalizacja projektu

Stwórz folder o nazwie np. `kurs_webpack` (nazwa jest dowolna, ale nie można folderu nazwać po prostu `webpack`), a w nim kolejny folder o nazwie `src` i `dist` (src – pliki źródłowe, dist – folder z plikami docelowymi).

Otwórz terminal, a następnie przejdź do folderu „kurs webpack" który utworzyłeś. W oknie terminala wpisz:

```
npm init -y
```

aby utwórzyć plik `package.json`. Aby zainstalować webpacka w terminalu wpisz

```
npm install webpack@^4.29.6 --save
```

oraz:

```
npm install webpack-cli@^3.3.0 --save
```

Otwórz plik `package.json` i sprawdź, czy wymienione wyżej komponenty zostały zainstalowane.

2.2 Tworzenie webpack.config.js

W folderze `src` utwórz plik `index.js` , a w nim wpisz:
`console.log("Hello World");`

Następnie, w głównym folderze utwórz plik `webpack.config.js` . W pliku tym wpisz:

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

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

A w pliku package.json dopisz do istniejącego już polecenia `scripts` :

```
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
```

Następnie w oknie terminala wpisz:

```
npm run build
```

W folderze `dist` powinien pojawić się plik `bundle.js` ! Gdy go otworzysz to zwróć uwagę, że jest on zminifikowany!

2.3 Import wielu plików *.js

W folderze `src` stwórz plik `sum.js` i wpisz do niego:

```
function sum(a,b){
    return a+b;
}

module.exports = {
    sum
}
```

A do pliku `index.js` dopisz:

```
import { sum } from "./sum";
console.log("Hello World");
console.log(sum(2,3));
```

Aby sprawdzić, czy plik `index.js` wykorzysta funkcję z pliku `sum.js`, w oknie terminala wpisz:

```
npm run build
node dist/bundle.js
```

Jeżeli zobaczyłeś wynik działania funkcji (czyli liczbę 5), to usuń teraz cały folder `dist`.

2.4 Tryb Production - development

Zmodyfikuj plik `package.json`:

```
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack --mode=production",
    "build:dev": "webpack --mode=development"
```

Uruchom poleceniem `npm run build:dev`

Zauważ, że plik bundle.js został utworzony, ale jak zajrzysz do niego, to on nie został zminifikowany! To dzieje się w wersji produkcyjnej!

Podzieliliśmy go również ze względu na szybsze wykonanie w trybie produkcyjnym.

Zobacz więcej: https://webpack.js.org/concepts/mode/

2.5 Prosty projekt

W folderze `dist` utwórz plik `index.html`, a przed tagiem zamykającym `body` wpisz:

```
<h1 id="demo"></h1>
<script src="bundle.js"></script>
```

A w pliku `index.js`:

```
let heading = document.querySelector("#demo"),
    sumValue = sum(10,5);

heading.innerHTML = `10 + 5 = ${sumValue}`;
```

ponownie w oknie termianala wpisz `npm run build:dev`, po czym otwórz plik `index.html`. Na stronie internetowej owinieneś zobaczyć zmiany.

2.6 DevServer

W powyższym przykładzie wadą jest fakt, że zmiany zmiennej nie powodują automatycznej zmiany na stronie internetowej. Aby nastąpiło automatyczne odświeżanie, musimy zainstalować plugin:

```
npm install webpack-dev-server@^3.7.1 --save
```

Musimy wskazać, które pliki będą uruchamiane:

```
module.exports = {
    entry: "./src/index.js",
    output: {
        path: path.resolve(__dirname, "dist"),
        filename: "bundle.js"
    },
    devServer:{
        contentBase: path.join(__dirname,"dist"),
        port: 9000
    }
}
```

A do pliku `package.json` (pod `build:dev`) dopisz:

```
"start:dev": "webpack-dev-server --mode-development"
```

W okni terminala wpisz:

```
npm run build:dev
npm run start:dev
```

Spróbuj dokonać zmian w pliku `index.js` – automatycznie zobaczysz je na stronie! Problem w tym, że nie można tego dokonać w pliku html. Aby to zmienić, dopisz jedną linijkę:

```
  devServer:{
        contentBase: path.join(__dirname,"dist"),
        port: 9000,
        watchContentBase: true
    }
```

A następnie ponownie wykonaj polecenia: `npm run build:dev`, `npm run start:dev` i dokonaj zmian w pliku index.html. Pamiętaj, aby od czasu do czasu jednak uruchomić skrypt `npm run build: dev` (szczególnie wtedy, gdy kończysz

pracę).

3. Loaders

Loadery takie to zewnętrzne biblioteki, które rozszerzają możliwości webpacka o umiejętność, między innymi, ładowania plików innych typów niż JavaScript. Dostarczają też możliwości pre-procesowania plików podczas ich ładowania w module (oraz przeprowadzania w zasadzie dowolnych operacji na ładowanych modułach). Dzięki temu możemy ładować pliki napisane w TypeScript/ES6, które "w locie" zostaną przetłumaczone na JavaScript (zrozumiały dla przeglądarek). To samo z plikami LESS/Sass - odpowiednie loadery potrafią "w locie" przetłumaczyć je na czysty CSS.

Więcej informacji: https://webpack.js.org/concepts/loaders/

3.1 CSS - cz. I

Zainstaluj:

```
npm install css-loader@^2.1.1 --save
npm install style-loader@^0.23.1  --save
```

W folderze `src` utwórz folder `css`, w nim plik `index.css` i zmień kolor tła na czerwony! W pliku `index.js` dodaj linię:

```
import style from "./css/index.css"
```

a w `webpack.config.js` dopisz:

```
module: {
        rules: [{
            test: /\.css$/, //what loaders should be applied
            use: ["style-loader","css-loader"] //what loaders we want to
        }]
    }
```

Wykonaj instrukcję `npm run build:dev`, a następnie uruchom server. Strona powinna mieć czerwone tło!

3.2 CSS - cz. II

Aby jednak w folderze `dist` pojawił się nowy plik css, musimy go stworzyć!

Zainstaluj:

```
npm install file-loader@^3.0.1 --save
```

Do ostatnich zmian w configu podmień linijkę na

```
use: ["style-loader/url","file-loader"] //what loaders we want to use
```

Następnie zbuduj aplikację. Jak widzisz, pojawił się plik, jednak o dziwnej nazwie. Aby nazwać go tak, jak w folderze src, edytuj moduł, aby wyglądał tak:

```
module: {
    rules: [{
        test: /\.css$/,
        use: [
            {loader: "style-loader/url"},
            {loader: "file-loader",
            options: {name: "[name].[ext]"}}
        ]
    }]
}
```

Ponownie zbuduj aplikację! Plik index.css powinien pojawić się w folderze dist. Otwórz plik index.html i zauważ, że w inspektorze, w sekcji head, jest odniesienie do pliku css – mimo że go nie umieszczaliśmy.

3.3 Babel

Służy do transpilacji kodu JS.

Instalacja:

```
npm install babel-loader@^8.0.5  @babel/core@^7.4.0 @babel/preset-env@
^7.4.2 --save
```

Zmiana `webpack.config.js` :

```
    module: {
        rules: [{
            test: /\.css$/,
            // use: ["style-loader/url","file-loader"]
            use: [
                {loader: "style-loader/url"},
                {loader: "file-loader",
                options: {name: "[name].[ext]"}}
            ]
        },
    {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
            loader: 'babel-loader',
            options: {
                presets: ['@babel/preset-env']
            }
        }
    }]
    }
```

Wykonaj polecenie `npm run build:dev` i zobacz, czy faktycznie kod został transpilowany.

3.4 SCSS

Instalacja:

```
npm install sass-loader@^7.1.0  node-sass@^4.11.0 --save
```

W pliku konfiguarcyjnym podmień css na scss i dopisz do `use`:

```
module: {
        rules: [{
            test: /\.scss$/,  //zamiana css na scss
            use: [
                "style-loader",
                "css-loader",
                "sass-loader"] //dopisanie komponentu
        },
    {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
            loader: 'babel-loader',
            options: {
                presets: ['@babel/preset-env']
            }
        }
    }]
    }
```

W folderze `src` stwórz plik `index.scss` z następującą regułą:

```
body{
    h1{
        color: red;
    }
}
```

Nie zapomnij również o zmianie importowanego pliku w pliku `index.js`

3.5 Obrazki

Za modułem służącym do transpilacji JS, utwórz nowy, w celu przekazania obrazka:

```
{
        test: /\.(png|svg|jpg|gif)$/,
        use: {
                    loader: "file-loader",
                    options: {
                        name: "[name].[ext]"
                    }
                }

    }
```

W pliku `index.js` umieść z kolei:

```
import Icon from "./assets/img/proba.png"

let myIcon = new Image();
myIcon.src = Icon;
document.querySelector("div").appendChild(myIcon);
document.querySelector("div").classList.add("change");
```

Nie zapomnij o stworzeniu diva w pliku html! W konsoli wpisz `npm run build:dev`, a obrazek powinien zostać przeniesiony!

Oczywiście możesz stworzyć odniesienie do obrazka w pliku scss:

```
body{
    h1{
        color: red;
    }

    background: url("./../assets/img/proba.png");

    .change{
    background-color: yellow;
    }
}
```

3.6 HTML

Źródła:

- https://github.com/webpack-contrib/html-loader
- https://webpack.js.org/plugins/

Instalacja:

```
npm install html-loader@^0.5.5  html-webpack-plugin@^3.2.0 --save
```

Przenieś plik index.html z folderu `dist` do folderu `src`, zakomentowując przy okazji linijkę dotyczącą odniesienia do skryptu `bundle.js`.

Na początku pliku `webpack.config.js` dopisz:

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

za devServer:

```
plugins: [
        new HtmlWebpackPlugin({
            template: "./src/index.html"
        })
    ],
```

a na samym końcu reguł (w tablicy) dopisz:

```
{
    test: /\.(html)$/,
    use: ["html-loader"]
}
```

Całość:

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

module.exports = {
    entry: "./src/index.js",
    output: {
        path: path.resolve(__dirname, "dist"),
        filename: "bundle.js"
    },
    devServer: {
        contentBase: path.join(__dirname, "dist"),
        port: 9000,
        watchContentBase: true
    },
    plugins: [
        new HtmlWebpackPlugin({
            template: "./src/index.html"
        }),
    ],
    module: {
        rules: [
            {
                test: /\.scss$/,
                use: ["style-loader",
                    "css-loader",
                    "sass-loader"
                ]
            },
            {
                test: /\.js$/,
                exclude: /node_modules/,
                use: {
```

```
                    loader: "babel-loader",
                    options: { presets: ["@babel/preset-env"] }
                }
            },
            {
                test: /\.(png|svg|jpg|gif)$/,
                use: {
                    loader: "file-loader",
                    options: {
                        name: "[name].[ext]"
                    }
                }
            },
            {
                test: /\.(html)$/,
                use: ["html-loader"]
            }
        ]
    }
}
```

Dzięki temu strona jest dynamicznie tworzona przez webpacka!

3.7 Czyszczenie folderu dist przed build

Instalacja:

```
npm install clean-webpack-plugin@^2.0.1 --save
```

W pliku `webpack.config.js` dodaj:

```
 const CleanWebpackPlugin = require("clean-webpack-plugin");
```

oraz za devServer:

```
plugins: [
        new HtmlWebpackPlugin({
            template: "./src/index.html"
        }),
        new CleanWebpackPlugin()
    ],
```

3.8 Hashing

Dopisz:

```
output: {
        path: path.resolve(__dirname, "dist"),
        filename: "[contenthash].bundle.js"
    },
```

Gdy dokonasz zmian w pliku `index.js` zobaczysz, że zmieniła się nazwa pliku!

Oczywiście uruchamiasz to poleceniem: `npm run build:dev`

3.9 Minifikacja

Instalacja:

```
npm install mini-css-extract-plugin@^0.5.0 --save
npm install optimize-css-assets-webpack-plugin@^5.0.1 --save
```

W pliku konfiguracyjnym dopisz:

```
const MiniCssExtractPlugin = require("mini-css-extract-plugin");
const OptimizeCssAssetsPlugin = require("optimize-css-assets-webpack-p
lugin");
```

oraz w `plugins` dopisz:

```
new MiniCssExtractPlugin({
        filename: "[name].[hash].css"
    }),
new OptimizeCssAssetsPlugin({
    assetNameRegExp: /\.css$/
}),
```

podmień również regułę na:

```
 use: [
                // "style-loader",
                MiniCssExtractPlugin.loader,
                "css-loader",
                "sass-loader"]
        },
```

A następnie zbuduj aplikację I zaobserwuj zmiany!

3.10 Browser-sync

Instalacja:

```
npm install browser-sync browser-sync-webpack-plugin --save
```

Zmiany w pliku konfiguracyjnym:

```
const BrowserSyncPlugin = require("browser-sync-webpack-plugin");
```

Dodaj do `plugins`:

```
new BrowserSyncPlugin({
            host: 'localhost',
            port: 9100,
            proxy: 'http://localhost:9000'
        }, {
            reload: false
        })
```

Po zbudowaniu i wystartowaniu servera zobaczysz:

```
   Local: http://localhost:9100
External: http://192.168.0.173:9100
 --------------------------------------
          UI: http://localhost:3001
 UI External: http://localhost:3001
 --------------------------------------

PANEK ZARZĄDZANIA: http://localhost:3001
```

Co się stanie, gdy zmienisz wartość właściwości `reload` na `true`?

3.11 Autoprefixer

Instalacja:

```
npm install autoprefixer@9.7.4  postcss-loader@^3.0.0 --save
```

W folderze głównym utwórz plik `postcss.config.js` i wpisz:

```
module.exports = {
    plugins: {
        "autoprefixer": {}
    }
}
```

W pliku konfiguracyjnym webpacka:

```
const autoprefixer = require("autoprefixer");
const webpack = require("webpack");
```

do pluginów dopisz:

```
new webpack.LoaderOptionsPlugin({
    options: {
        postcss: [
            autoprefixer()
        ]
    }
})
```

a do reguły `css` dopisz:

```
{
            test: /\.scss$/,
            use: [
                // "style-loader",
                MiniCssExtractPlugin.loader,
                "css-loader",
                "sass-loader",
                "postcss-loader"]
        },
```

Do pliku html dopisz:

```
<div class="autoprefixer">hej</div>
```

a do styli SCSS:

```
.autoprefixer {
    user-select: none;
}
```

Zbuduj i w inspektorze sprawdź, czy zostały dodane!

3.12 Many entrypoints

Wskazanie

```
  entry: {        //umieszczenie w nawiasach jest ważne
        "index": "./src/index.js",
    },
    output: {
        path: path.resolve(__dirname, "dist"),
        filename: "[name].[contenthash].bundle.js"
    },
    devServer: {
        // contentBase: path.join(__dirname, "dist"),
        port: 9000,
        // watchContentBase: true
    },
    plugins: [
        new HtmlWebpackPlugin({
            template: "./src/index.html",
        }),
        new HtmlWebpackPlugin({
            template: './src/kontakt.html',
            inject: true,
            chunks: ['index'],
            filename: 'kontakt.html'
        }),
```

3.13 CopyWebpackPlugin

Instalacja:

```
npm install copy-webpack-plugin@^5.1.1 --save
```

Dodaj

```
const CopyWebpackPlugin = require('copy-webpack-plugin');
```

Wskaż które pliki mają zostać przeniesione skąd dokąd:

```
    new CopyWebpackPlugin([{
        from: './src/assets',
        to: './dest/assets'
    }]),
```