Zprovoznění WebPacku 5 VanillaJS 01/2024

Data pro kurz: https://bit.ly/webpack-vanilla

Základ

Vytvořit nový Nodejs projekt a nainstalovat **Webpack** a **Webpack-cli** (Command Line Interface)

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

Vytvořit strukturu

```
root
├─ src
│  └─ index.js
└─ dist
└─ index.html
```

V **package.json** smazat `"main": "index.js"`, a přidat `"private": true,`
Nainstalovat **lodash**

```
npm install lodash
```

Napsat JS kód do **index.js**

```
import isEqual from 'lodash/isEqual';

const array1 = [1, 2, 5];
const array2 = [1, 2, 5];

console.log(isEqual(array1, array2).toString());
```

Do **dist/index.html** přidat odkaz na skript ./main.js

```
<script src="main.js" defer></script>
```

Spustit

```
npx webpack
```

(npx spouští binární soubor ./node_modules/.bin/webpack, který byl nainstalovaný s WebPackem) - to vytvoří produkční main.js

Konfigurace

Na hlavní úrovni vytvořit **webpack.config.mjs** (defaultní, nemusí se uvádět)

```
import { dirname, resolve } from 'node:path';
import { fileURLToPath } from 'node:url';

const directoryName = dirname(fileURLToPath(import.meta.url));

const config = {
mode: 'production',
entry: './src/index.js',
output: {
filename: 'main.js',
path: resolve (directoryName, 'dist'),
},
};

export default config;
```

a spustit

```
npx webpack --config webpack.config.mjs
```

do sekce scripts v **package.json** přidat `"build": "webpack"`
a zkusit vybuildit

```
npm run build
```

Moduly

Do souboru package.json přidat vlastnost

```
"type": "module",
```

protože budeme pracovat s ES6 moduly
Soubor **index.js** změnit na **index.mjs** a upravit v souboru **webpack.config.mjs** entry point

```
entry: './src/index.mjs',
```

a zkusit vybuildit

```
npm run build
```

Tentokrát to vybuildit nejde, protože Lodash je psán v CommonJs (používá require). Je tedy do souboru **webpack.config.mjs** přidat následující sekci

```
module: {
```

```
rules: [
{
test: /\.mjs$/i,
resolve: {
byDependency: {
esm: {
fullySpecified: false,
},
},
},
},
],
},
resolve: {
extensions: ['.js', '.mjs', '.webp'], // TODO otewstovat
}
```

a zkusit vybuildit.

CSS vytvářený pomocí LESS

CSS

Do **index.html** přidat

```
<div id="app"></div>
```

a přidat atribut pro skript `defer`
Do **index.mjs** přidat

```
const header = document.createElement('h1');
header.textContent = 'Moje Webpack aplikace';
document.querySelector('#app').appendChild(header);
```

Vytvořit složku **styles** a v ní soubor **index.css**

```
root
└─ src
└─ css
└─ index.css
```

do souboru **index.css** přidat

```
:root {
font-size: 16px;
font-family: Verdana, Geneva, Tahoma, sans-serif;
}
```

```
h1 {
font-size: 250%;
color: darkred;
}
```

Nainstalovat loadery

```
npm install --save-dev style-loader css-loader
```

style-loader nahraje CSS jako js modul (`import './style.css';` v js souboru)
css-loader interpretuje `@import` a `url()` jako `import/require()`
Do configu webpacku přidat do sekce `module.rules` objekt

```
{
test: /\.css$/u,
use: [
'style-loader',
'css-loader',
],
},
```

Naimportovat styl do **index.mjs**

```
import './css/index.css';
```

a vybuildit.

Rozdělení na funkční bloky (CSS moduly)

Vytvořit soubor **headers.module.css** a přesunout do něj deklaraci selektoru `h1` (změnit barvu nadpisu) a selektor `h1` změnit na selektor třídy `.headerOne`.
Upravit sekci `use` pro CSS v **webpack.config.mjs**:

```
use: [
'style-loader',
{
loader: 'css-loader',
options: {
importLoaders: 1,
modules: true,
}
},
],
```

- *importLoaders: 1* - nastaví počet loaderů spuštěných před css loaderem (0 vypíná)
- *modules: true* - zapne podporu modulů

Následně naimportovat v souboru **index.mjs** styly a přidat třídu:

```
import * as style from './css/headers.module.css';

…

header.className = style.headerOne;
```

a vybulidit.

Pokud je třeba lepší název třídy než hash, stačí upravit ve **webpack.config.mjs** u CSS loaderu volbu modules (z true na objekt):

```
modules: {
localIdentName: '[name]_[local]_[hash:base64:5]',
},
```

více v https://github.com/webpack-contrib/css-loader#mode

LESS

```
npm install --save-dev less less-loader
```

upravit config webpacku:

```
{
test: /\.(css|less)$/u,
use: [
'style-loader',
{
loader: 'css-loader',
…
},
'less-loader',
],
},
```

Změnit přípony na ***.less** (i v **index.mjs**) a upravit less soubory:
variables.less

```
@main-size: 16px;
@color-red: rgb(177, 0, 0);
@color-blue: rgb(0, 14, 206);
@color-green: rgb(0, 129, 39);
```

index.less

```
@import './variables.less';
```

```
:root {
font-size: @main-size;
font-family: Verdana, Geneva, Tahoma, sans-serif;
}
```

headers.module.less

```
@import './variables.less';

.headerOne {
font-size: 250%;
color: lighten(@color-green, 25%);
}
```

a vybuildit.
Lze to ještě upravit takto:

index.mjs

```
const header = document.createElement('h1');
header.className = style.headerOne;

const webpackElement = document.createElement('span');
webpackElement.className = style.webpack;
webpackElement.textContent = 'Webpack';

header.appendChild(document.createTextNode('Moje '));
header.appendChild(webpackElement);
header.appendChild(document.createTextNode(' aplikace s less'));

document.querySelector('#app').appendChild(header);
```

headers.module.less

```
.headerOne {
font-size: 2.5rem;
color: lighten(@color-green, 25%);

.webpack {
color: @color-green;
}
}
```

CSS moduly - vypnutí jmenných importů

CSS třídy z modulu se importují takto:

```
import * as style from './css/headers.module.css';
…
header.className = style.headerOne;
webpackElement.className = style.webpack;
```

resp.

```
import { headerOne, webpack } from './css/headers.module.less';
…
header.className = headerOne;
webpackElement.className = webpack;
```

Jde o jmenný import (named import). Lépe se pracuje s defaultním importem:

```
import style from './css/headers.module.css';
…
header.className = style.headerOne;
webpackElement.className = style.webpack;
```

Je to potřeba nastavit v souboru **webpack.config.mjs** v sekci `module` u CSS pravidla

```
modules: {
localIdentName: '[name]_[local]_[hash:base64:5]',
namedExport: false,
},
```

Obrázky (např. pozadí a ikony - statické)

nainstalovat clsx:

```
npm install clsx
```

do **webpack.config.mjs** pak přidat do sekce `rules (import převede na url a přesune do výstupní složky)`

```
{
test: /\.(webp|png|svg|jpg|jpeg|gif)$/i,
type: 'asset/resource',
},
```

udělat strukturu

```
root
└─ src
├─ css
│  └─ image.module.less
├─ createImage.mjs
└─ images
```

```
├─ webpack.png
├─ webpack.svg
└─ webpack.webp
```

do **createImage.mjs** přidat

```
import clsx from 'clsx';
import style from './css/image.module.less';

const createImage = (source, className) => {
const container = document.createElement('div');

const image = new Image();
image.className = clsx(style.image, className);
image.src = source;

container.appendChild(image);

return container;
};

export default createImage;
```

index.mjs

```
import Logo from './images/webpack.svg';
import createImage from './createImage.mjs';
import style from './css/headers.module.less';
import imageStyle from './css/image.module.less';
```

místo

```
document.querySelector('#app').appendChild(header);
```

vložit

```
const app = document.querySelector('#app')
app.appendChild(header);
app.appendChild(createImage(Logo));
```

image.module.less

```
@import './variables.less';

.image {
aspect-ratio: 1/1;
border: 1px solid lighten(@color-red, 25%);
width: 20rem;
```

```
}

.withPadding {
padding: 1.5rem;
}
```

Vyzkoušet i **png** a **webp**, při nich přidat do funkce druhý atribut **imageStyle.withPadding**:

```
app.appendChild(createImage(Logo, imageStyle.withPadding));
```

Aliasy

Do **webpack.config.js** přidat sekci

```
resolve: {
alias: {
Images: resolve(directoryName, 'src/images/'),
},
},
```

Aby VScode umělo napovídat aliasy, je potřeba vyrobit soubor v rootu jsconfig.json:

```
root
└─ jsconfig.json
```

jsconfig.json:

```
{
"compilerOptions": {
"baseUrl": "./",
"module": "ESNext",
"paths": {
"Images/*": ["src/images/*"]
},
"target": "ES6"
},
"exclude": ["node_modules"]
}
```

a pak přepsat v **index.mjs**

```
import Logo from 'Images/webpack.svg';
```

Dev a Produkce

Příprava

Soubor **webpack.config.mjs** přejmenovat na **webpack.config.common.mjs** a přídat soubory

```
root
├─ webpack.config.dev.mjs
└─ webpack.config.prod.mjs
```

Nainstalovat

```
npm i -D webpack-merge
```

V souboru **webpack.config.common.mjs** smazat řádek

```
mode: 'production',
```

webpack.config.dev.mjs

```
import { merge } from 'webpack-merge';
import common from './webpack.config.common.mjs';

const devConfig = merge(common, {
mode: 'development',
});

export default devConfig;
```

webpack.config.prod.mjs

```
import { merge } from 'webpack-merge';
import common from './webpack.config.common.mjs';

const prodConfig = merge(common, {
mode: 'production',
});

export default prodConfig;
```

package.json

```
"scripts": {
"build": "webpack --config webpack.config.prod.mjs",
"dev": "webpack --config webpack.config.dev.mjs"
},
```

a zkusit (pak se mrknout na výsledný ./dist/main.js)

```
npm run build
```

a

```
npm run dev
```

Watching

package.json

```
"scripts": {
"build": "webpack --config webpack.config.prod.mjs",
"dev": "webpack --config webpack.config.dev.mjs",
"watch": "webpack --watch --config ./webpack.config.dev.mjs"
},
```

spustit

```
npm run watch
```

a zkusit něco změnit ve zdrojácích

Dev server

nainstalovat

```
npm i -D webpack-dev-server
```

V **webpack.config.common.mjs** vyexportovat konstantu directoryName:

```
export const directoryName =
dirname(fileURLToPath(import.meta.url));
```

do **webpack.config.dev.mjs** přidat importy a sekci

```
import { join } from 'node:path';
…
import common, { directoryName } from './webpack.config.common.mjs';
…
devServer: {
historyApiFallback: true,
open: true,
port: 9000,
static: [
{
directory: join(directoryName, "./dist/"),
publicPath: '/', // nepovinné; defaultně je totiž '/'
},
],
},
```

a do **package.json** do sekce scripts přidat

```
"start": "webpack serve --config ./webpack.config.dev.mjs",
```

Smazat obsah ve složce **dist** (kromě HTML souboru) a spustit příkaz

```
npm start
```

Source mapy pro vývoj

- definují se pomocí `devtool` vlastnosti v konfigu
- pro produkční prostředí je defaultně `devtool: false`
- pro vývojové prostředí je `devtool: 'eval'` (rychlý build i rebuild, ale nemá přesně řádky)

Do **index.mjs** přidat

```
throw new Error('test');
```

a otestovat dev i produkci.
Do **webpack.config.dev.mjs** přidat sekci

```
devtool: false,
```

a otestovat dev.

- `'eval-source-map'`: pomalejší při inicializaci, nejkvalitnější sourcemapy
- `'inline-source-map'`: vloží mapy do bundlu (pomalý rebuild)
- `'cheap-source-map'`: vytvoři jeden mapový soubor pro všechny moduly (pomalý rebuild)
- `'eval-cheap-source-map'`: rychlý rebuild

Jsou i další varianty (viz https://webpack.js.org/configuration/devtool/)

Formátování a linting kódu

Prettier má velice zastaralý plugin i loader

ESlint

Lintery slouží k získání vyšší kvality kódu (např. arrow funkce atp.)
do souboru **index.mjs** vložit (bacha, dvojité uvozovky)

```
console.log("debug Webpack")
debugger;
```

Nainstalovat eslint a základní konfiguraci pro vanilla js

```
npm i -D eslint globals eslint-config-airbnb-base
```

Poznámka: eslint je kvůli nekompatibilním modulům potřeba držet na verzi 8.57.0 (15. 9. 2024)
Vytvořit

```
root
├─ eslint.config.mjs
└─ eslint.config.prod.mjs
```

Do souboru **eslint.config.mjs** přidat konfiguraci z https://pastebin.com/5mJGBGMR

Fix for flat config (nová konfigurace)

Protože airbnb-base není prozatím ve *flat config*, je potřeba udělat dočasný fix (https://github.com/eslint/eslintrc)

Nainstalovat oficiální záchrannou utilitu:

```
npm i -D @eslint/eslintrc
```

```
root
└─ flatCompatibility.mjs
```

flatCompatibility.mjs:

```
/* eslint-disable import/no-extraneous-dependencies */
import path from 'node:path';
import { fileURLToPath } from 'node:url';
import { FlatCompat } from '@eslint/eslintrc';

const filename = fileURLToPath(import.meta.url);
const dirname = path.dirname(filename);

const compat = new FlatCompat({
baseDirectory: dirname,
resolvePluginsRelativeTo: dirname,
});


export default compat;
```

eslint.config.mjs:

```
// import airbnb from 'eslint-config-airbnb-base';
import compat from './flatCompatibility.mjs';

const config = [
...compat.extends('airbnb-base'),
…
```

Až bude **airbnb config** ve *flat configu*, bude stačit smazat soubor **flatCompatibility.mjs** a změnit soubor **eslint.config.mjs**:

```
import airbnb from 'eslint-config-airbnb-base';

const config = [
airbnb,
```

…

Podívat se na pravidla, zejména na pravidlo *import/extensions*.
Do **eslint.config.prod.mjs** vložit

```
import common from './eslint.config.mjs';

const config = [
...common,
{
rules: {
'no-console': [
'error',
{
allow: [
'warn',
'error',
],
},
],
'no-debugger': 'error'
},
},
];

export default config;
```

Protože Eslint neumí ještě aliasy, je třeba v souboru **index.mjs** upravit řádky

```
// import Logo from 'Images/webpack.webp';

…

app.appendChild(createImage(/* Logo */ './x.webp',
imageStyle.withPadding));
```

Pak zkusit

```
npx eslint ./src
```

a

```
npx eslint --config eslint.config.prod.mjs ./src
```

Aliasy v eslintu

Nainstalovat

```
npm i -D eslint-import-resolver-alias
```

Do souboru **eslint.config.mjs** přidat sekci

```
settings: {
'import/resolver': {
alias: {
map: [
['Images', './src/images/'],
],
extensions: ['.webp'],
},
},
},
```

Je potřeba ještě upravit pravidlo pro koncovku souboru pravidle **import/extensions**:

```
'import/extensions': [
'error',
'never',
{
mjs: 'always',
webp: 'always',
},
],
```

VSCode ESlint plugin problém

Plugin ve VSCode vyhodnocuje všechny soubory, tedy i ty, které nejsou v src (třeba konfigurák webpacku).
Pokud tot nechci, tak do souboru **eslint.config.mjs** musím změnit import nekompatibilního modulu a předhodit mu všechny ignorované soubory:

```
{
...compat.config({
extends: 'airbnb-base',
env: {
es2020: true,
node: true,
},
ignorePatterns: [
'eslint.config.mjs',
'eslint.config.prod.mjs',
'flatCompatibility.mjs',
'webpack.config.common.mjs',
'webpack.config.dev.mjs',
```

```
'webpack.config.prod.mjs',
'dist/**/*',
],
}),
…
}
```

nebo stačí

```
{
...compat.config({
extends: 'airbnb-base',
env: {
es2020: true,
node: true,
},
ignorePatterns: [
'/*.mjs',
'dist/**/*',
],
}),
…
}
```

Vždy je potřeba restartovat VSCode (pro jistotu)!

Oprava některých chyb

```
npx eslint --fix --config ./eslint.config.prod.mjs ./src
```

Přidání eslintu do webpacku

```
npm i -D eslint-webpack-plugin
```

Do souborů **webpack.config.dev.mjs** a **webpack.config.prod.mjs** přidat

```
import ESLintPlugin from 'eslint-webpack-plugin';
```

a

```
plugins: [
new ESLintPlugin({
configType: 'flat', // používám flat config
eslintPath: 'eslint/use-at-your-own-risk', // který je pro eslint
plugin experimentální
extensions: ['js', 'mjs'],
}),
```

```
],
```

- *extensions*: jaké soubory se mají kontrolovat, defaultně je to *'js'*, takže se nemusí zadávat.

V produkčním souboru přidat vlastnost pro přepsání konfigurace:

```
overrideConfigFile: './.eslintrc.prod.json',
```

StyleLint

nainstalovat

```
npm i -D stylelint stylelint-config-standard stylelint-config-
recommended-less
```

vytvořit

```
root
├─ .stylelintignore
└─ .stylelintrc.mjs
```

a do **.stylelintrc.mjs** vložit

```
/** @type {import('stylelint').Config} */

const config = {
extends: [
'stylelint-config-standard',
'stylelint-config-recommended-less',
],
rules: {},
};

export default config;
```

Komentář v prvním řádku zapne našeptávač pro konfig (vyzkoušeno ve VS Code)
.stylelintignore

```
/dist/*.css
```

zkusit

```
npx stylelint src/**/*.less
```

Automatický fix:

```
npx stylelint src/**/*.less --fix
```

Pokud chci využít původní třídy napsané v camelCase, je třeba přidat pravidlo do sekce rules:

```
'selector-class-pattern': [
'^([a-z][a-z0-9])([a-zA-Z0-9]+)*$',
{
message: (selector) => `Expected class selector "${selector}" to be
camellCase`,
},
],
```

Možná bude třeba restartovat editor, pokud je nainstalovaný plugin Stylelint

Přidání stylelintu do webpacku

Pozn.: k 20. 1. 2024 (11/24 už neplatí) tento plugin vyžaduje stylelint <= 15.0.0. Pokud vznikne při instalaci chyba, je třeba:

- odinstalovat
  - stylelint
  - stylelint-config-standard
  - stylelint-config-recommened-less

```
npm un stylelint stylelint-config-standard stylelint-config-
recommended-less
```

a nainstalovat konkrétní verze

```
npm i -D stylelint@15.10.2 stylelint-config-standard@34.0.0
stylelint-config-recommended-less@2.0.0
```

teprve poté nainstalovat plugin do webpacku

```
npm i -D stylelint-webpack-plugin
```

Do souboru **webpack.config.common.mjs** přidat

```
import StylelintPlugin from 'stylelint-webpack-plugin';
```

a

```
plugins: [
new StylelintPlugin({
extensions: ['less', 'css'],
}),
],
```

Linting

Do sekce *scripts* v souboru **package.json** přidat:

```
"lint": "eslint --config eslint.config.prod.mjs ./src && stylelint
src/**/*.less",
```

a pak zkusit

```
npm run lint
```

Pomůcky

Přidání šablony

(index.html vzít z ./dist)

```
root
└─ templates
└─ index.html
```

smazat tag `<script></script>`, změnit tag `<title></title>` v hlavičce

```
<title><%= htmlWebpackPlugin.options.title %></title>
```

Nainstalovat

```
npm i -D html-webpack-plugin
```

a vložit do **webpack.config.common.mjs**

```
import HtmlWebpackPlugin from 'html-webpack-plugin';
```

a do pole **plugins**

```
new HtmlWebpackPlugin({
template: './templates/index.html',
title: 'První Webpack aplikace',
}),
```

Přidání proměnné na základě webpackového konfigu

DefinePlugin provede přímé nahrazení proměnné její hodnotou, proto musí být hodnota uvedena jako řetězec (ideálně přes JSON.stringify)
do souboru **webpack.config.dev.mjs** vložit

```
import webpack from 'webpack';

const { DefinePlugin } = webpack;
```

a do pole **plugins** dát

```
new DefinePlugin({
PRODUCTION: JSON.stringify(false),
}),
```

do souboru **webpack.config.prod.js** vložit

```
import webpack from 'webpack';

const { DefinePlugin } = webpack;
```

a do pole `plugins` dát

```
new DefinePlugin({
PRODUCTION: JSON.stringify(true),
}),
```

a následně lze renderovat kód v **index.mjs**

```
const para = document.createElement('p');
para.textContent = `${PRODUCTION ? 'Jde' : 'Nejde'} o produkční
kód.`;
app.appendChild(para);
```

Jak se zbavit eslint chyby nedefinovaná proměnná `PRODUCTION`:

do konfiguračního souboru Eslintu přidat do sekce `languageOptions.globals`

```
languageOptions: {
...
globals: {
...
PRODUCTION: true,
},
},
```

Promazání dist složky

Do sekce *output* ve **webpack.config.common.mjs** vložit

```
output: {
clean: true,
...
},
```

Nefunguje to při dev serveru, protože ten si ukládá soubory do paměti.

PostCSS a CSS soubory samostatně

nainstalovat

```
npm i -D postcss postcss-loader
npm i -D postcss-preset-env
npm i -D mini-css-extract-plugin
```

z **webpack.config.common.mj**s přesunout do **prod** a **dev** souborů sekci

```
module: {
rules: [
{
test: /\.css|less$/u,
use: […],
},
],
},
```

do **webpack.config.prod.mjs** přidat postcss loader za css-loader
vyhodíme CSS do samostatného souboru pomocí pluginu a loaderu
MiniCssExtractPlugin.loader, kterým vyměníme *style-loader*:

```
import MiniCssExtractPlugin from 'mini-css-extract-plugin';

plugins: [
new MiniCssExtractPlugin({
filename: '[name].[contenthash].css',
}),
],

module: {
rules: [
{
test: /\.css|less$/u,
use: [
MiniCssExtractPlugin.loader,
{…},
{
loader: 'postcss-loader',
options: {
postcssOptions: {
plugins: [
'postcss-preset-env',
'autoprefixer',
],
},
},
},
'less-loader',
],
},
]
```

```
},
```

Vytvořit

```
root
└─ .browserslistrc
```

.browserslistrc

```
last 2 version
> 1% in CZ
not dead
```

podporované prohlížeče lze ověřit pomocí

```
npx browserslist
```

do souboru **headers.module.less** přidat do třídy *headerOne* :

```
.headerOne {
...
appearance: value;
...
}
```

a spustit produkční build a zkontrolovat CSS

Minifikace

Javascript

nainstalovat

```
npm i -D terser-webpack-plugin
```

webpack.config.prod.mjs

```
import TerserPlugin from 'terser-webpack-plugin';

const prodConfig = merge(common, {
…
optimization: {
minimize: true,
minimizer: [
new TerserPlugin(),
],
},
…
```

```
});
```

CSS

nainstalovat

```
npm i -D css-minimizer-webpack-plugin
```

přidat do **webpack.config.prod.mjs** do sekce *optimization.minimizer* nový plugin:

```
import CssMinimizerPlugin from 'css-minimizer-webpack-plugin';

minimizer: [
new TerserPlugin(),
new CssMinimizerPlugin(),
],
```

otestovat minifikaci pomocí `npm run build`

Obrázky

https://webpack.js.org/plugins/image-minimizer-webpack-plugin/#getting-started
Obrázky jsou optimalizovány pomocí ImageMinimizerWebpackPlugin. K dispozici jsou nástroje pro bitmapy

- imagemin
- sharp (nejvýkonější (podle dokumentace webpacku)

a pro SVG

- svgo

nainstalovat plugin pro minimalizaci a minimalizátor

```
npm i -D image-minimizer-webpack-plugin sharp svgo
```

Bezztrátová komprese

do sekce *optimization.minimizer* ve **webpack.config.prod.mjs** přidat nový plugin:

```
import ImageMinimizerPlugin from 'image-minimizer-webpack-plugin';

optimization: {
…
minimizer: [
…
new ImageMinimizerPlugin({
minimizer: {
implementation: ImageMinimizerPlugin.sharpMinify,
options: {
encodeOptions: {
},
},
```

```
},
}),
],
…
});
```

Ztrátová komprese

Do sekce *encodeOptions* ve **webpack.config.prod.mjs** přidat konfiguraci pro všechny typy:

```
encodeOptions: {
jpeg: {
quality: 100,
},
webp: {
lossless: true,
},
avif: {
lossless: true,
},
png: {
quality: 100,
},
gif: { },
},
```

A vyzkoušet s příkazem `npm run build`, kdy dojde ke zmenšení *.webp obrázku. Zkusit i *.png obrázek.

V **index.mjs** změnit import

```
import Logo from 'Images/webpack.png';
```

A spustit `npm run build`.

Pozn:

- u PNG je *quality: 100* stejné jako *bezztrátová komprese*, jde tedy vynechat
- GIF nemá ztrátovou kompresi

Více na https://sharp.pixelplumbing.com/api-output

SVG

do sekce *optimization.minimizer* ve **webpack.config.prod.mjs** přidat nový plugin:

```
optimization: {
…
minimizer: [
…
new ImageMinimizerPlugin({
minimizer: {
```

```
implementation: ImageMinimizerPlugin.svgoMinify,
options: {
encodeOptions: {
multipass: true,
plugins: [
"preset-default",
],
},
},
},
}),
],
…
});
```

a v souboru **index.mjs** vyměnit png za svg

```
import Logo from 'Images/webpack.svg';
```

A spustit `npm run build`. Podívat se na minifikovaný soubor ve složce */dist*.
Více na https://github.com/svg/svgo/blob/main/README.md

Vytvoření WebP a avif

Assets

Do obrázkového minimizeru ve **webpack.config.prod.mjs** přidat hned za sekci *minimizer* sekci *generator*:

```
generator: [
{
type: "asset",
implementation: ImageMinimizerPlugin.sharpGenerate,
options: {
encodeOptions: {
webp: {
quality: 90,
},
},
},
},
],
```

Toto je nastavení pro statické assety, které se kopírují pomocí *CopyPluginu*:

```
npm i -D copy-webpack-plugin
```

Do sekce *plugins* ve **webpack.config.common.mjs** přidat kopírování:

```
import CopyPlugin from 'copy-webpack-plugin';
plugins: [
…
new CopyPlugin({
patterns: [
{ from: "./assets/images", to: "images" },
],
}),
],
```

Husky

Když potřebuju odeslat do repozitáře gitu korektní kód (který prošel lintováním)

```
npm i -D husky
```

Do sekce *scripts* v souboru **package.json** přidat:

```
"prepare": "husky install",
```

A spustit

```
npm run prepare
```

Toto se děje jen jedinkrát, skript se bude nadále spouštět při instalaci naší aplikace vždy automaticky.
Tím se vyrobí správná složka

```
root
└─ .husky/
```

Do složky přidáme githook s kódem, který chce spustit:

```
npx husky add .husky/pre-commit "npm run lint"
```

A můžeme vesele komitovat s jistotou bezchybného kódu

```
git add ./.husky/pre-commit
git commit -m "pre-commit"
```

Jde to obdobně i na pre-push

```
npx husky add .husky/pre-push "npm run lint"
git add ./.husky/pre-push
git commit -m "pre-push"
git push
```