**Webpack**

**Встановлення:**

1. Скачуємо Node JS
2. Відкриваємо консоль від Git Bush в папці проекту (мати встановлений git )
3. Прописати команду: **npm init** – для встановлення папки(package.json)
4. **npm i -D  webpack webpack-cli** – команди для ініціалізації webpack
5. Створити файл для конфігурацій: webpack.config.js

**Повторне використання збірки в інших проектах:**

1. Переносимо файли package.json, webpack.config.js
2. Прописуємо команду: npm i

**Особливості:**

1. Плагіни встановленні з прапорцем **--save** - бере участь і при роботі проекту

З прапорцем **--save –dev (-D)** – плагіни беруть участь лише при розробці

1. В файлі package.json – в полі ‘script’ – вказуємо команди, які ми викликатимемо для запуску тої чи іншої частини коду і викликається в терміналі: **npm run comandName**

**Команди:**

**npm i -D  webpack webpack-cli** – встановлення клієнт вебпака

**npm i webpack-dev-server –D –** встановлення dev server для відображення змін

**npm i cross-env -D** функціонал, який визначає в якій ОС ми знахоимось і задає системні змінні **NODE\_ENV** в вказаній йому конфігурації згідно вимог ОС

**Babel**

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

**npm install --save @babel/polyfill**

добавляє бабел лоадер і поліфіли, що переводить новий синтаксис js в зрозумілий усім браузерам**.** В самі ж конфігурації в поле **entry** дописати в масив шляхів для кореневого файла, що потребує поліфіл і додати шлях **- @babel/polifill // [‘@babel/polifill’, ‘./index.js’]**

**При виникненні помилок уважно прочитати їх і додати плагіни, що виріщують її**

**Плагіни і їх конфігурації:**

1. **Html-webpack-plugin –** для взаємодії із html файлом і підключення в нього актуальних скриптів

**npm i html-webpack-plugin –D**

* **Без налаштувань** переносить папку index.html в dist + автоматично підключає усі скрипти збандлені webpack в даний файл, враховуючи їх хеші при створені імені файлу з даним функціоналом
* **Налаштування** ({
  + **template**: **‘path\_to\_src\_htmlFile’**
  + **minify**: {

**collapseWhitespace:** **true/false –** мініфікація html

}

})

1. **Clean-webpack-plugin –** чистить папку діст перед загрузкою нового бандлу

**npm I clean-webpack-plugin –D**

1. **CopyWebpackPlugin –** плагін для копіювання файлів з і в точно вказане місце

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

* Налаштування плагіна записуються в масив ([]), масив об’єктів, що містять поля **from** i **to**

[{

     from: path.resolve(\_\_dirname, 'src/favicon.ico'),

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

}]

1. **MiniCssExtractPlugin()** – плагін для збірки css стилів підключених в корінний js в окремий файл в dist

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

1. **TerserWebpackPlugin(), OptimizeCssAssetPlugin()-** оптимізація css/js при білді

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

**npm i optimize-css-assets-webpack-plugin –D**

1. **Imagemin-webp-webpack-plugin –**конвертує jpg/jpeg/png в webp

**npm i imagemin-webp-webpack-plugin**

1. **BundleAnalizerPlugin() –** плагін для наглядного аналізу бібліотек, що використовуються в проекті із інфографіками їх місткості в проекті

**npm i webpack-bundle-analyzer**

**Будова коду:**

**const path** = require('path'); - змінна для роботи із шляхами

**\_\_dirname** – глобальна секція, яка містить шлях до кореня проекту

1. **module.exports = {}** – об’єкт для задання **конфігурацій webpack**
2. **entry: './index.js' –** поле для задання корінного js файлу, в який підключені всі решта js файли.
   1. Якщо необхідно вказати декілька кореневих файлів, то вписуємо їх в масив шляхів (**entry:** **[‘./src/index\_1.js’, ‘./src/index\_2.js’]**)

1. **output: { filename: 'bundle.js', path: … } –** поле для задання назви файлу, в якому після компіляції міститиметься весь js код проекту і який буде переміщений в dist
   1. **path: 'D:\projects\app\dist' –** ручне налаштування кореневої деректорії, в яку компілюватиместься код
   2. **path: path.resolve(\_\_dirname, 'dist') – автопошук** кореневої деректорії
2. **context: path.resolve(\_\_dirname, 'src') –** бажане налаштування для конфігурації - деректорії з якої братимуться файли для компіляції
3. **mode: ‘development’ – режим** компіляцї коду
4. **moduls: {} –** використовується для **створення** будь-яких **налаштувань, робота із лоадерами –** що дають webpack розуміти формати відмінні від js I json
   1. **rules[] –** масив об**’**єктів для загрузки loader-ів для роботи із файлами певних типів**,** і визначає правила виконання загрузчиків
      1. **test: /\.js$/** - вказує які файли потрібно трансформувати (в даному випадку із розширенням js); якщо webpack в імпорті зустрічає файл із вказаним типом то він **use** – використовує потрібні лоадери
      2. **exclude: /node\_modules/ -** вказуємо які файли потрібно виключити
      3. [**use**](https://webpack.js.org/configuration/module/#rule-use)**: {}** – визначаємо додаткові налаштування

**-** поле для зазначення утиліт для трансформації коду

**Лоадери:**

Всі данні в форматі відмінному від js для компіляції через webpack в папку збілдженого проекту – повинні бути підгружені в кореневий js файл і так як webpack не розуміє формати відмінні від .js i .json ми підключаємо в лоадери для розширення розуміння webpack сторонніх типів данних. Лоадери працюють із права на ліво

-.**сss** – **‘css-loader’**, **‘style-loader’**. Стилі компілюються в html в тег style

//style-loader – дозволяє webpack розуміти формат .css

//css-loader – підключає стилі в index.html в тег style

// Для загрузки css в окрему деректорію потрібно в лоадери підключити **mini-css-extract-plugin** замість css-loader

**// npm i -D style-loader css-loader**

* **(.png/.jpg/.gif/.svg/.ttf/woff…) – ‘file-loader’**

**// npm I –D file-loader**

**-.xml – ‘xml-loader’**

**// npm I –D xml-loader**

**-.csv – ‘csv-loader’**

**// npm i –D csv-loader**

**//npm I –D papaparse**

**-.less – ‘less-loader’**

**//npm i –D less-loader**

**-.sass/.scss – ‘node-sass sass-loader’**

**// npm i node-sass sass-loader -D**

**-.js – ‘babel-loader’**

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

**//npm install @babel/polyfill --save**

* **.ts – ‘@babel/preset-typescript’**

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

* **.jsx - ‘@babel/preset-react’**

**//npm install @babel/preset-react --save-dev**

**-jsling** для будь-якого типу файлів, що використовують синтаксис js**– ‘esling-loader’**

**//npm I esling-loader –D**

1. **Plugins:** [] – масив для налашування плагінів
   1. const **HtmlWebpackPlugin** = require('html-webpack-plugin'); - підключення плагіна до збірки
   2. plugins: [new **HtmlWebpackPlugin**()] – приклад підключення плагіна
2. **mode:** **‘development’ / ‘production’** – формат в якому наші дані будуть білдитись
3. **resolve: {**

**extensions: [‘.js’, ‘.png’, ‘.json’, …] –** в extensions вказуємо типи даних, які буде розуміти webpack по замовчуванню при прописувані імпортів в js

**alias:** {} – об’єкт в якому ми прописуємо скорочення шляхів

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

1. **optimization:** - конфігурація, що задається при наявності декількох точок входу, що мають спільні імпорти одного файлу, завдяки налаштуванню нижче, спільний для обох фінальних файлів файл виноситься окремо

{

**splitChunks:** {

**chunk: ‘all’**

}

}

1. **devServer:** - конфігурація для активації dev server для відображення змін при розробці автоматично. Дана конфігурація використовується лише в режимі розробки, так як файли з білда загружаються в оперативну пам’ять, а не в папку dist

(**npm i –D webpack-dev-server**)

{

**port:** **4000**…

}

1. **devtool:** ‘source-map’ – поле, що дозволяє добавити sourcemap в проект. Можна налаштувати для роботи із конкретним режимом

**devtool:** **isDev ? ‘source-map’ : ‘’**

**Патерни розробки конфігурації:**

* + - 1. При наявності декількох точок входу (кореневих файлів), які нам потрібно перенести в dist – ми використовуємо

**entry:** {

**bundle1:** **‘path1…’**

**bundle2:** **‘path2…’**

}

**output:** {

**filename:** **‘[name].[contenthash].js’**

}

В **[name]** – підставиться ключ об’єкта **entry**, а завдяки **contenthash**, ми позбуваємось проблем із кешуванням файлів браузером при розробці

Найкраще використовувати із плагіно (**Clean-webpack-plugin, Html-webpack-plugin({template: ‘…’})**)

* + - 1. Для активації певного функціоналу при роботі того чи іншого режимо збірки необхідно визначити змінну, що повертатиме true/false який ми передаватимемо через мінну в конфігурацію того чи іншого модуля (**npm i -D cross-env**)

**const isDev = process.env.NODE\_ENV === 'development'**

Щоб задати **NODE\_ENV** потрібно в **package.json** в полі **script** до потрібної команди дописати **cross-env NODE\_ENV=development/production**