Webpack_ВебКадеми

node.js, npm

Сайт node.js: https://nodejs.org/en/
Сайт npm: https://www.npmjs.com/
Сайт webpack: https://webpack.js.org/

Вместе с node.js устанавливается npm
$ node -v - версия node.js
$ npm -v - версия npm

$ npm i - установить все модули (если есть package.json)

Начальная структура проекта:
src/ - источники
    js/
    index.html
dist/ - скомпилированные файлы
    css/
    img/
    js/
    index.html
    favicon.ico

Инициализируем проект:
$ npm init --yes - новый проект без вопросов
В корне появится файл package.json

Webpack. Начальные настройки

Установим webpack и webpack-cli

$ **npm i webpack webpack-cli --save-dev**

После этого появились node_modules и package-lock.json

Напишем config файл для webpack
Создадим в корне файл `webpack.config.js`:
! Пути пишем от корня и **./**

```
const path = require('path');  //утилита path, часть окружения node.js

module.exports = {
  entry: './src/js/main.js',   // точка входа
  output: {    // куда компилируем
    path: path.resolve(__dirname, 'dist/js'),  // путь. __dirname - константа, определяющая путь к директории
    filename: 'bundle.js',  // название файла
  }
};
```

Создадим тестовый test.js:

```
console.log("Imported module");
export default 110;
```

В main.js импортируем этот файл:

```
import num from "./test";  // здесь расширение js  можно не писать
console.log(`Тестирование импорта. Импортировали число - ${num}`);
```

В файле package.json определить команды для запуска webpack

```
"scripts": {
    "start": "webpack",
 },
```

В терминале **$ npm run start**
Появится файл dist/js/bundle.js

Далее можем запустить через live server **dist/index.html** и убедимся, что сборка работает.
Мы получили **минифицированный** файл dist/js/bundle.js. А хотим получить файл **в режиме разработки dev**. Тогда в webpack.config.js добавим строку:

```
module.exports = {
   entry: './src/js/main.js',
   output: {
      path: path.resolve(__dirname, 'dist/js'),
      filename: 'bundle.js',
   },
   mode: "development"   // для режима разработки
};
```

Сделаем то же, но удобнее. В webpack.config.js уберем строку mode: "development" и в package.json переделаем строки:

```
"scripts": {
   "dev": "webpack --mode development",  // для разработки
   "build": "webpack --mode production"  // для production
 },
```

Теперь **команды для запуска**:
$ npm run dev - для разработки
$ npm run build - для production

Установка webpack dev server

Устанавливаем пакет `webpack-dev-server`:

npm i webpack-dev-server --save-dev

Добавляем настройку `devServer` для сервера в файл с конфигурацией webpack.config.js:
Также там внесем изменения в **output**, так как нам нужно следить не за **js файлом**, а делать обновление **index.html**. Также меняем настройку пути для результирующего файла. Итого получим:

```
module.exports = {
 entry: "./src/js/main.js",
 output: {
   path: path.resolve(__dirname, "dist"),
   filename: "js/bundle.js",  // также подкорректируем путь
 },
 devServer: {
   contentBase: "./dist", // путь к папке, из которой нужно поднимать сервер и следить за проектом
 },
};
```

Добавляем команду для старта сервера в файл `package.json`:

```
 "scripts": {
   "dev": "webpack --mode development",
   "build": "webpack --mode production",
   "start": "webpack-dev-server --mode development --open"
 },
```

Запускаем сервер:
Предварительно удалим **bundle.js**

$ **npm run start**

! Сервер запускается. **bundle.js** не создается в папке dist/js/, а **генерируется виртуально**
Если **bundle.js** нужен физически, то тогда использовать команду
$ npm run dev - для разработки
или
$ npm run build - для production

HTML шаблон с помощью html-webpack-plugin

Сделаем так, чтобы index.html создавался автоматически и попадал в нужную папку **dist** из папки **src**

Установим плагин `html-webpack-plugin`:

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

Подключим данный пакет в настройки webpack.
Файл webpack.config.js.

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

module.exports = {
...
 devServer: {
...
 },
 plugins: [new HtmlWebpackPlugin()],
};
```

Запустим dev server. Будет пустая страница. Сборка модулей работает. При этом index.html **на диске не формируется**.
Добавим параметры:

```
 plugins: [
  new HtmlWebpackPlugin({
   filename: "index.html",  // имя выходного файла
   template: "./src/index.html",  // имя шаблона
  }),
 ],
```

Теперь из **src/index.html** можно **убрать** подключение скрипта **main.js**. Запустим сервер **dev server** или в режиме **dev** и увидим в **dist/index.html** подключения скрипта в шапке автоматически:

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

Настройка babel. Babel polyfill

Установка пакетов:

npm i babel-loader @babel/core @babel/preset-env -D

Подключаем babel в webpack.config.js
Работаем по документации: https://webpack.js.org/loaders/babel-loader/

```
module.exports = { ...
 plugins: [ ...
 ],
 module: {
  rules: [
   {
    test: /\.m?js$/,   // для каких файлов будет работать
    exclude: /node_modules/,   // папки для исключения
    use: { // опции для babel-loader
     loader: "babel-loader",
     options: {
      presets: ["@babel/preset-env"],
     },
    },
   },
  ],
 },
};
```

Создаем в корне файл `.babelrc` с настройками для babel:

```
{
 "presets": [
  [
   "@babel/env",
   {
    "useBuiltIns": "usage",
    "corejs": "3",
    "targets": {
     "browsers": ["last 5 versions", "ie >= 8"] // какие версии поддерживать, для ie отдельно
```

```
      }
    }
  ]
 ]
}
```

Можно протестировать добавив новый ES6 синтаксис в JS файлы и посмотреть на итоговый скомпилированный код.
В src/main.js добавим:

```
import num from "./test";
const x = 123;
console.log(`Тестирование импорта. Импортировали число - ${num}`);
console.log(`Variable x => ${x}`);
```

Запустим в режиме dev. В **bundle.js** видим ... var x=123;

Кроме исправления синтаксиса нужны еще **полифилы** - функции, которые "говорят" старым браузерам, как должны работать новые функции

Немного теории:
Babel – это транспилер. Он переписывает современный JavaScript-код в предыдущий стандарт. На самом деле, есть две части Babel:

- Во-первых, **транспилер**, который переписывает код. Современные сборщики проектов, такие как webpack или brunch, предоставляют возможность запускать транспилер автоматически после каждого изменения кода, что позволяет экономить время.
- Во-вторых, **полифил**.

polyfill.io – сервис, который автоматически создаёт скрипт с полифилом в зависимости от необходимых функций и браузера пользователя.

Добавим поддержку полифилов

npm i @babel/polyfill -S

Подключаем полифиллы в самом начале нашего основного JS файла, src/main.js:

```
require('@babel/polyfill');
```

Протестируем:
Можно протестировать работу полифилов. Добавив код на промисах и проверив его работу в старых версиях браузеров, например в IE 11

src/main.js:

```
function functionFirst() {
 return new Promise(function (resolve, reject) {
  setTimeout(function () {
   console.log("functionFirst");
   resolve();
  }, 1500);
 });
}

function functionSecond() {
 return new Promise(function (resolve, reject) {
  setTimeout(function () {
   console.log("functionSecond");
   resolve();
  }, 1000);
 });
}

function functionThird() {
 return new Promise(function (resolve, reject) {
  setTimeout(function () {
   console.log("functionThird");
   resolve();
  }, 500);
 });
```

```
}

console.log("Start");

functionFirst()
   .then(function () {
      return functionSecond();
   })
   .then(function () {
      return functionThird();
   })
   .then(function () {
      console.log('Next code');
   });

console.log('Final');
```

Выносим babel polyfill в отдельный файл

В текущем варианте сверху всех js файлов нужно подключать полифилы. Можно сделать по-другому:
В файле webpack.config.js вместо entry: "./src/js/main.js", добавим

```
 entry: {
   babelpolyfill: "@babel/polyfill",  // полифил
   index: "./src/js/main.js",  // наш файл
 },
```

Там же нужно изменить выходные точки:

```
 output: {
   path: path.resolve(__dirname, "dist"),
   filename: "js/[name].bundle.js",  // теперь в папке js будет не один файл
 },
```

В src/main.js удалим строку:

```
require('@babel/polyfill');
```

Протестируем, что все по-прежнему работает

В **dist/js/** появятся файлы babelpolyfill.bundle.js и index.bundle.js и оба файла подключены в index.html