```
npm init –y
```

```
npm i -D webpack webpack-cli
```

Создаем директорию «src» для хранения файлов приложения. Я начну с создания простого файла «index.js»:

Базовая настройка

Создаем файл «webpack.config.js» в корневой директории проекта.

Точка входа

```
// webpack.config.js
const path = require('path')
        module.exports = {
        entry: {
                main: path.resolve(__dirname, './src/index.js'),
        },
}
```

Точка выхода

```
module.exports = {
// ...
output: {
        path: path.resolve(__dirname, './dist'),
        filename: '[name].bundle.js',
        },
}
```

Добавляем скрипт «build» в файл **«package.json»**

```
// package.json
 "scripts": {
        "build": "webpack"
 }
```

Запускаем вебпак:

```
npm run build
```

В директории **«dist»** создается файл «index.bundle.js». Файл не изменился, но мы успешно осуществили сборку проекта.

Плагины

- html-webpack-plugin — создает HTML-файл на основе шаблона

•npm i -D html-webpack-plugin

Создаем файл **«template.html»** в директории **«src»**.

Добавляем в настройки вебпака свойство «plugins», где определяем плагин, название выходного файла (index.html) и шаблон:

```
// webpack.config.js
const path = require('path')
const HtmlWebpackPlugin = require('html-webpack-plugin')
```

```
module.exports = {
// ...
plugins: [
        new HtmlWebpackPlugin({
                title: 'webpack Boilerplate',
                template: path.resolve(__dirname, './src/template.html'), // шаблон
                filename: 'index.html', // название выходного файла
                }),
        ],
}
```

Запускаем сборку. Директория «dist» теперь содержит файл «index.html» с подключенным в нем скриптом.

```
// index.js
// создаем элемент заголовка
const heading = document.createElement('h1')
heading.textContent = 'Как интересно!'

// добавляем заголовок в DOM
const root = document.querySelector('#root')
root.append(heading)
```

Очистка

Установим clean-webpack-plugin, очищающий директорию «dist» при каждой сборке проекта.

•clean-webpack-plugin — удаляет/очищает директорию сборки проекта

npm i -D clean-webpack-plugin

```
// webpack.config.js
const HtmlWebpackPlugin = require('html=webpack-plugin')
const { CleanWebpackPlugin } = require('clean-webpack-plugin')

module.exports = {
// ...

plugins: [
// ...
        new CleanWebpackPlugin(),
        ],
}
```

Babel (JavaScript)

Babel — это инструмент, позволящий использовать будущий JavaScript сегодня.

- babel-loader — транспиляция файлов с помощью Babel и вебпака
- @babel/core — транспиляция ES2015+ в обратно совместимый JavaScript
- @babel/preset-env — полезные стандартные настройки Babel
- @babel/plugin-proposal-class-properties — пример кастомной конфигурации Babel (установка свойств экземпляров в теле класса, а не в его конструкторе)

npm i -D babel-loader @babel/core @babel/preset-env @babel/plugin-proposal-class-properties

```
// webpack.config.js
module.exports = {
// ...

module: {
        rules: [
        // JavaScript
                {
                test: /\.js$/,
                exclude: /node_modules/,
                use: ['babel-loader'],
                },
        ],
        }
}
```

создаем файл **".babelrc"** в корне проекта:

```
// .babelrc
{
        "presets": ["@babel/preset-env"],
        "plugins": ["@babel/plugin-proposal-class-properties"]
}
```

Запускаем сборку с помощью **npm run build**. Теперь все работает.

Изображения

Создаем директорию «src/images», помещаем туда изображение и пытаемся импортировать его в файле «index.js»:

```
// index.js

import example from './images/smile.jpeg'

// ...
```

При запуске сборки будет выброшено исключение.

Для изображений следует использовать тип «asset/resource». Обратите внимание, что речь идет именно о типе (type), а не о загрузчике (loader):

```
// webpack.config.js

module.exports = {

// ...



module: {
        rules: [
        // изображения
                {
                test: /\.(?:ico|gif|png|jpg|jpeg)$/i,
                type: 'asset/resource',
                },
        ],
```

```
        },
}
```

В директории **«dist»** появляется новый файл.

Шрифты и другие встраиваемые данные

Вебпак также имеет встроенный модуль для обработки некоторых встраеваемых данных, таких как шрифты и SVG. Для этого достаточно указать тип «asset/inline»:

```
// webpack.config.js
module.exports = {
// ...

module: {
        rules: [
        // шрифты и SVG
        {
                test: /\.(woff(2)?|eot|ttf|otf|svg|)$/,
                type: 'asset/inline',
                },
        ],
        },
}
```

Стили

- sass-loader — загружает SCSS и компилирует его в CSS
- node-sass — Node Sass
- css-loader — загрузка стилей
- style-loader — применение стилей к элементам DOM

npm i -D sass-loader css-loader style-loader node-sass

```
// webpack.config.js
module.exports = { // ...
        module: {
                rules: [ // CSS, Sass
                        {
                        test: /\.(scss|css)$/, use: ['style-loader', 'css-loader', 'sass-loader'],
                        },
                ],
        },
}
```

- Для проверки работоспособности названных инструментов создадим файл «src/styles/main.scss»
- Импортируем этот файл в index.js и добавляем 4 загрузчика.
- *// index.js*
- import './styles/main.scss'

• *// ...*

Разработка

webpack-dev-server — это сервер для разработки

npm i -D webpack-dev-server

```
// webpack.config.js
const webpack = require('webpack')

 module.exports = { // …
 mode: 'development',
 devServer: {
  historyApiFallback: true,
   contentBase: path.resolve(__dirname, './dist'),
   open: true, compress: true, hot: true, port: 8080, },


 plugins: [ // ... // применять изменения только при горячей перезагрузке
new webpack.HotModuleReplacementPlugin(), ],
 }
```

Для запуска сервера используется команда «webpack serve»:

```
// package.json
"scripts": { "start": "webpack serve" }
```

npm run start