Webpack

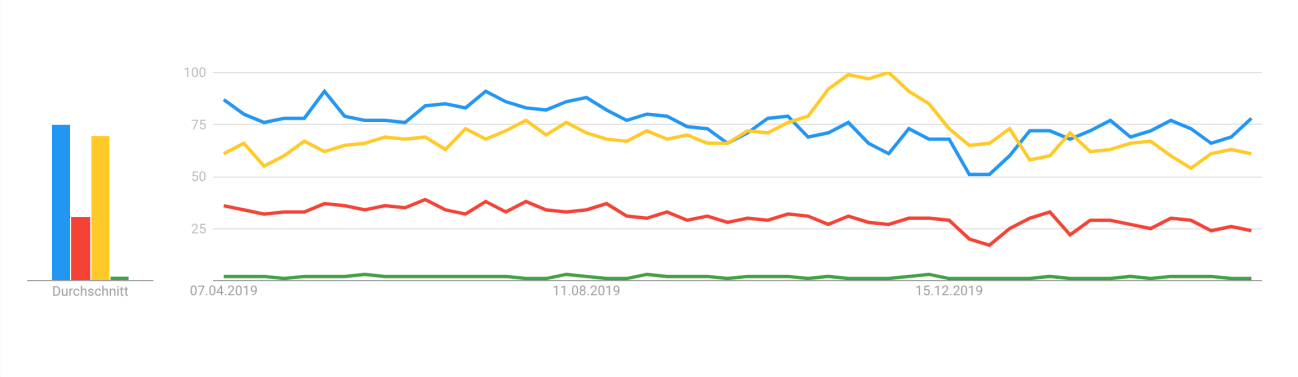
Erste Einblick

Webpack ist ein sogenannter Module Bundler, das heißt, es hilft beim Bündeln von JavaScript-Modulen unter Berücksichtigung der Abhängigkeiten. Webpack kennt man jedoch auch zum Transformieren und Zusammenfassen anderer Dateien wie Stylesheets und Bilder.

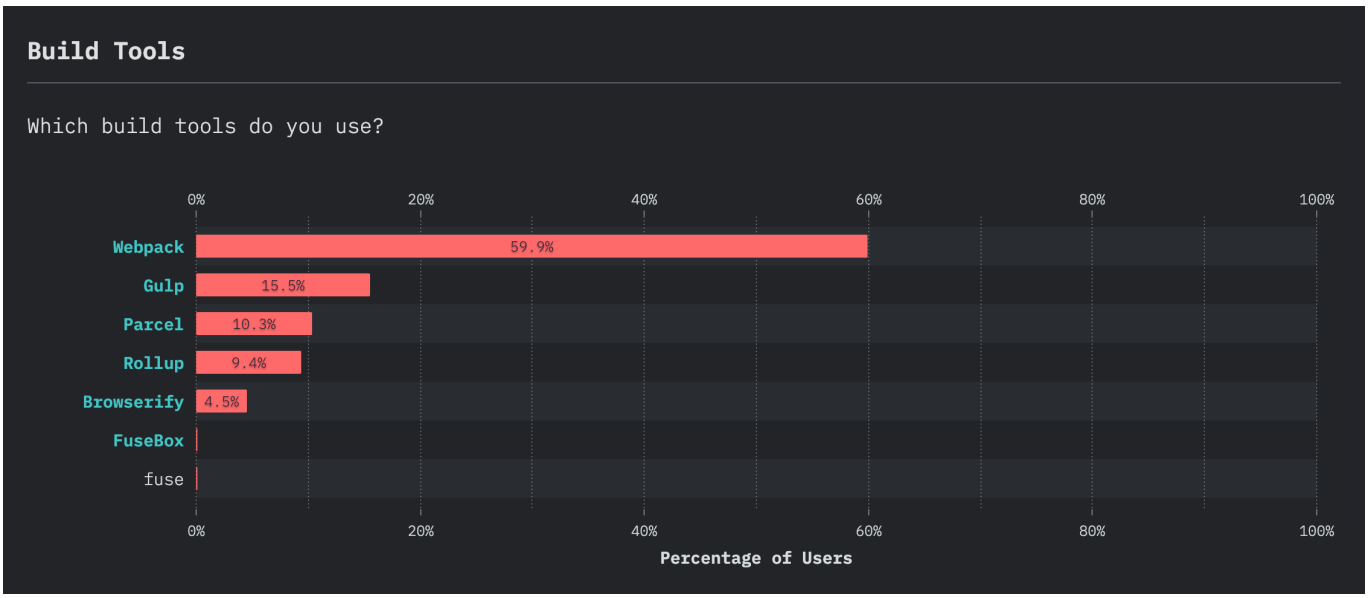
Build Tools

Die Aktuellen Google Trends (stand 06.04.2020)

- Webpack
- Gulp
- Grunt
- Browserify



Auswertungen von stateofjs.com



Modul Bundling

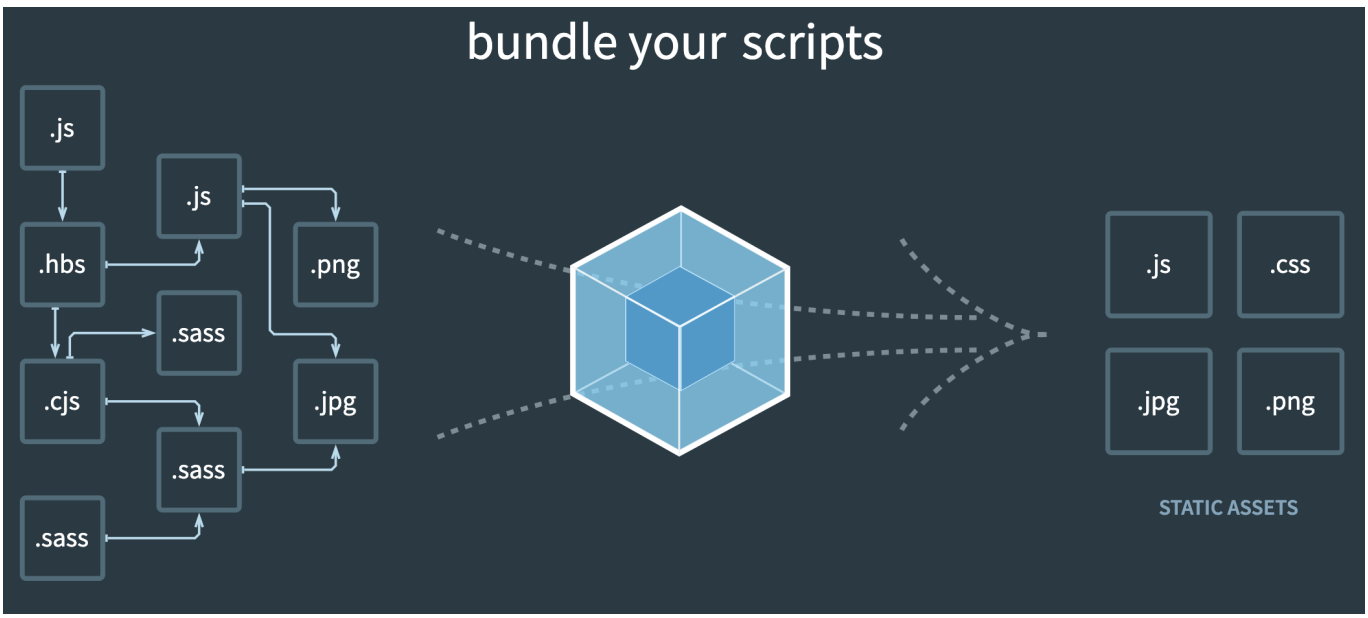
Modul Bundling ist ein komplexerer Vorgang als das Reine-Datei-Zusammenfassen, denn es werden auch Abhängigkeiten berücksichtigt. Webpack behandelt dabei nicht nur JavaScript Module sondern berücksichtigt auch alle anderen Komponenten einer Webseite, also CSS Dateien und Bilder.

Webpack Module

Module sind bei Webpack weit gefasst. Es gehören dazu ES2015 import-Anweisungen, require()-Anweisungen, @import innerhalb von css/sass/less, und es gehören dazu Bildverweise im Stylesheet oder in der HTML-Datei. Webpack erledigt dabei nicht nur das Zusammenfassen der Dateien unter Berücksichtigung der Abhängigkeiten, sondern hilft auch bei der Transformation, wenn zum Beispiel, ES6-Code benutzt wird oder SASS, dann kann die Bearbeitung der Dateien ebenfalls über Webpack durchgeführt werden.

Entry Point

Ausgehend von sogenannten Entry-Points beginnt Webpack die Abhängigkeiten zu analysieren, und dann werden alle Pakete zu statischen Bundeln zusammgefasst, die vom Browser geladen werden können.



Das Bild zeigt den Vorgang. Wir haben unsere unterschiedlichen Module, die Abhängigkeiten voneinander aufzeigen, und daraus werden dann die statischen Assets generiert.

4 Begriffe

- Entry Point:
  Von dort beginnt Webpack, die Abhängigkeiten zu analysieren.
- Output:
  Output ist der Speicherort, wo die Bundels erzeugt werden.
- Loader:
  Webpack versteht von Haus aus nur JavaScript, und Loaders transformieren andere Dateitypen in Module.

- Plugins:
  Es gibt viele Plugins für weitergehende Aktionen wie beispielsweise Kompilierung oder Definition von Codeteilen etc..

Zusammenfassung

Webpack kümmert sich nicht nur um JavaScript-Module, sondern ebenfalls um andere Dateien wie Bilder oder CSS. Und außerdem kann es auch für verschiedene Transformationen verwendet, die benötigen werden, also beispielsweise von SASS oder ES6-Code.

Webpack Installation

nodejs wird benötig install nodejs

Um Webpack zu installieren muss eine package.json Vorhanden sein. Mit

```
npm init -y
```

kann diese schnell generiert werden. Mit **-y** werden die default Einstellungen verwendet.

```
npm install --save-dev webpack
```

der Zusatz **--save-dev** sag npm das webpack nur zum Entwickeln benötigt wird.

Zum initialisieren von Webpack

```
./node_modules/.bin/webpack init
```

Was jetzt geschieht, ist, dass zum einen ein Ordner **node_modules** mit allen Modul eingelegt wird, und außerdem wird die Angabe zu Webpack in der Datei package.json festgehalten inklusive der benutzten Version.
Webpack wird lokal für das Projekt installiert. Es ist aber auch möglich Webpack mit dem zusatz **-g** global zu installieren. Das ist aber im allgemeinen nicht empfehlenswert, da projekte oft mit unterschiedlichen versionen von Webpack arbeitet.
In der **package.json** Datei wurde ein neuer Eintrag generiert

```
"devDependencies":{
  "webpack":"^4.42.1"
}
```

Um zu testen ob Webpack installiert wurde und auch funktioniert folgenden Befehl in die Komandozeile eingeben.

```
# MacOS
./node_modules/.bin/webpack --help

# Windows
.\node_modules\.bin\webpack --help
```

Zusammenfassung

Zum installieren von Webpack wird nodejs benötigt. Eine package.json Datei muss vorhanden sein. Anschließend kann webpack mittls npm installiert werden.

First steps

Um Datien mit Webpack bundeln zu können wird folgender Befehl für die CLI benötigt.

```
./node_modules/.bin/webpack input.File -o output.File
```

Wenn **-o output.File** nicht angegeben wird, wird der default Ordner von Webpack ersellt **dist/bundle.js**

Übung 1

Der public Ordner wird von Webpack automatisch erstellt. In dem Ordner befindet sich die erstelle Javascript Datei von Webpack.

Zusammenfassung

Bundle-Vorgänge können direkt über die CL durchgeführt werden. Wichtig ist, dass zuerst der Webpack-Befehl angeben wird, dann den Entry-Point, wo soll also begonnen werden, die Abhängigkeiten zu analysieren, und als Letztes den Namen und den Pfad zur Datei, die erstellt werden soll. Webpack beginnt dann mit der Analyse bei dem Entry-Point.

Webpack Config Datei

Sämtliche Webpack Optionen können über die CLI aufgerufen werden, doch ist es wesentlich angenehmer nicht jedesmal sämtliche Parameter zu schreiben. Aus diesem Grund bietet Webpack die Möglichkeit einer config Datei in der die Optionen definiert werden. Diese Config Datei muss im root Ordner des projektes liegen.

Diese Datei kann beliebig benannt werden. Es muss aber beim ausfüren des Webpack Befehls der Name der config Datei angegeben werden. Webpack sucht von sich aus nach einer **webpack.config.js** Datei. Ich empfehle den Namen beizubehalten.

Der Inhalt der Config Datei schaut wie folgt aus:

```
// path ist ein Modul von node.js und hilft bei der arbeit mit Dateien und Ordn
ern
const path = require('path');

module.exports = {
  // Der Pfad zur Ausgangsdatei an der Webpack beginnt die Abhängigkeiten zu an
alysieren
  entry: '',
  // Welche Datei soll geschrieben werden
  output: {
    filename: ,
    path: path.resolve(__dirname, 'output.Folder')
  }
};
```

Da wir nun eine webpack.config.js Datei erstellt haben

```
./node_modules/.bin/webpack
```

Übung 2

Zusammenfassung

Der Standartname für Config Datei bei Webpack ist also webpack.config.js.
Folgend ist die einfachst mögliche Config Datei:

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

module.exports = {
  entry: './src/app.js',
  output: {
    filename: 'app.js',
    path: path.resolve(__dirname, 'public')
  }
};
```

Das Ganze ist eine JavaScript-Datei, und das Objekt muss exportiert werden.

Easy CL

Um beim Aufruf von Webpack nicht jedesmal den gesamten Pfand angeben zu müssen kann Webpack auch im script Bereich der package.json Datei angegeben werden.

```
...

  "scripts": {
```

```
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
...
```

Nun genügt die folgende Angabe um Webpack auszuführen:

```
npm run build
```

Übung 3

Zusammenfassung

Um nicht jedesmal den Pfad von Webpack angeben zu müssen kann die package.json Datei verwendet werden. Mit npm run * werden die Scripts ausgeführt.

Tree Shaking

Wenn ein neuer Computer gekauft wird und dieser viele Features besitzt, die überhaupt nicht gebraucht werden, sind dieses teuer Bezahlt. Ähnlich gilt dies für nicht Verwendeten Code, dieser ist unnötig und unvorteilhaft für die Performance. Webpack hilft. Das sogenannte **Tree Shaking** eliminiert Code der nicht genutzt wird. Es wird beim bundeln nur der Code importiert der auch wirklich benötigt wird.
Um Tree Shaking zu verstehen, muss man sich veranschaunlichen, dass Webpack die Abhängigkeiten analysiert und intern einen Abhängigkeitsgraph bzw. Abhängigkeitsbaum erstellt. Und wenn man diesen Baum schüttelt, dann fallen tote Äste herunter, das heißt, das, was nicht benötigt wird, wird ausgeklammert.

Übung 4

Zusammenfassung

Webpack analysiert den Code und übersprint Code Zeilen die nicht verwendet werden. Wenn in der config Datei kein Modus angegeben wird, wird der production modus verwendet.

Loaders

Mit Webpack kann ES6 Code geschrieben werden und mit einem Loader, z.B. Bable in gängigen JavaScript-Code umgewandelt werden. Loader werden bei Webpack eingesetzt, um unterschiedliche Dateien zu verarbeiten und unterschiedliche Aktionen durchzuführen. Es gibt Loader für die verschiedensten Arbeiten, also beispielsweise für JSON-Dateien. Es gibt Loader fürs Packaging, es gibt Loader für verschiedene Dialekte wie auch ES6, es gibt Babel für Templates und mehr.

Liste der Loaders

Zuerste muss ein Loader über npm installiert werden und anschließend in der config Datei definiert werden.

```
npm install -D babel-loader @babel/core @babel/preset-env webpack
```

webpack.config.js

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

module.exports = {
  entry: './src/app.js',
  output: {
    filename: 'app.js',
    path: path.resolve(__dirname, 'public')
  },

  //neu
  module: {
    //hier definieren wir unsere Regeln
    rules: [
    {
      // bei Tests geben wir an welche Dateitypen bearbeitet werden sollen. In
diesem Beispiel alle .js Dateien (diese werden mit einer regex ausgewählt)
      test: /\.fileName$/,
      // welche Ordner sollen nicht berücksichtigt werden.
      exclude: /(dirName)/,
      // bei use werden die Loader angegeben
      use: {
        loader: '',
        //zusätzlich können gewissen Optionen definiert werden.
        options: {
          presets: ['']
        }
      }
    }
  ]
  ////////////
}
};
```

Übung 5

In der src/app.js Datei befindet sich eine kurze ES6 Funktion. Diese addiert zu jedem Entry in diesem Array aus Zahlen +1. Bable wird nun dazu benutzt das die Array Function in eine Funktion umgewandelt wird die auch ältere Browser verstehen.

Zusammenfassung

Loader sind Zusatzfunktionen die das Arbeiten mit Webpack erleichtert. Es gibt viele verschiedene, auch für CoffeScript oder Typescript.

Entry Points

Sollte der Fall eintreten das für verschiedene Unterseiten unterschiedlicher Code benötigt wird, können mehrere Entry Points definiert werden. In der webpack.config.js bei entry die verschiedenen Punkte angeben, nach dem Schema:
Dateiname: 'pfad zur datei'
Bei output wird nun mit einem Platzhalter gearbeitet, dieser steht für die vergebenen Namen.

```
...
  entry: {
    main: '',
    second: ''
  },
  output: {
    filename: '[name].js',
    path: path.resolve(__dirname, 'public')
  }
```

Übung 6

Zusammenfassung

In Webpack ist es möglich mehrere Entry Points zu definieren. Durch das arbeiten mit einem Platzhalter werden die Dateien in dem entsprechendem Ordner gespeichert.

Cache

Wichtig für eine gute Performance ist die richtige Caching-Strategie, denn am schnellsten lädt Code, der gar nicht erst heruntergeladen werden muss, sondern beispielsweise noch im Browser-Cache vorhanden ist. Eine Caching-Strategie kann beispielsweise so aussehen, dass je nach Dateityp vorgegeben wird, wie lange dieser gecached werden darf. Also wie lange er noch aus dem Cache angefordert werden kann, bevor er dann neu aufgerufen wird.

- Klassische Methode:
  - app.js?build=1
  - app.js?build=2

Die klassische Methode besteht darin, dass an den Dateinamen Parameter angehängt werden, und wenn sich etwas ändert, dann ändernt sich der Parameter, und dadurch wird dann eine neue Datei angefordert, auch wenn im Cache angegeben ist.

Webpack bietet die Funktion eines Hash Wertes im Dateinamen z.B.:

```
build.738128djeshw8239s.js
```

Platzhalter:

- [hash] -> der Hash ist für alle Dateien des Builds gleich.
- [chunkhash] -> der Hash ist für jede Datei individuell

```
filename: '[name].[chunkhash].js'
```

Übung 7

Manifest.json

Wenn mit einem Hash Wert gearbeitet wird, unterscheidet sich der Name des Output-Files jedes mal wenn Webpack ausgeführt wird. Es muss ein Weg gefunden werden um alles dynamisch zu gestallten. Der einfachste Weg ist die Erstellung eines eigenen kleinen Plugins.

```
// unter den Plugins in der webpack.config.js
...

function(){
  //sobald Webpack fertig mit dem Compilieren ist wird es ausgeführt.
  this.plugin('done', stats => {
    //stats beeinhaltet sämtliche Informationen die beim Compilieren erstellt w
erden.
    require('fs').writeFileSync(
      path.join(__dirname, 'src/manifest,json'),
      // die stats zu json umwandeln und es in eine Datei namens manifest.json
speichern
      JSON.stringify(stats.toJson())
    )
  })
}
```

Die erstelte Datei wird sämtliche Informationen beeinhalten, bei größeren Projekten mit vielen Dependencies kann dieses File mehrere Hundert Zeilen lang sein. Aus diesem Grund ist es nicht zu empfehlen alle Dateien zu schreiben. **assetsByChunkName** beinhaltet die caching relevanten Daten.

```
...
"assetsByChunkName": {
  "main": "main.lkjsdkfjlskdjflks4582.js"
}
...
```

Jedesmal wenn Webpack Compiliert, wird auch die manifest.json neu generiert mit den benötigten Dateinamen.

```
...
    //Nur die benötigten ChunkNames in die manifest.json Datei schreiben.
```

```
    JSON.stringify(stats.toJson().assetsByChunkName)
...
```

Abhänig von dem Projekt kann kann die Datei ausgelesen werden und im index File eingebunden werden.

Zusammenfassung

Webpack bietet die Möglichkeit Dateien mit einem Hashwert zu versehen um bei der Handhabung mit dem Cache zu helfen. Durch das beifügen eines Platzhalters [hash] oder [chunkhash] wird der Hashwert automatisch generiert.

CSS

Webpack kümmert sich nicht nur um die Verarbeitung von JavaScript-Dateien, sondern kann ebenfalls CSS bearbeiten. CSS-Code können inline eingebunden werden, CSS-Dateien können gebündelt werden und natürlich auch LESS und SASS in CSS umwandeln.

Übung 8

Zusammenfassung

Wenn CSS-Code inline eingebunden wird, funktioniert das also folgendermaßen: Zunächst müssen die benütigten Loader installiert werden: css-loader, style-loader, und bei Bedarf auch den url-loader (für andere externe Dateien, di in Stylesheets vorkommen). Diese werden in der webpack.config.js Datei angegeben. In der app.js wird der CSS code importiert. Sobald Webpack ausgeführt wird, wird der CSS-Code inline eingebunden.

Externe CSS Dateien

CSS-Dateien inline einzubinden ist sinnvoll bei SPA, ansonsten ist es besser mit externen Stylesheets zu arbeiten, weil diese auch gecached werden können.

Übung 9

Zusammenfassung

Um ein externes Stylesheet zu erzeugen, wird das **mini-css-extract-plugin** benötigt und die Konfigurationsdatei muss entsprechend angepasst werden, dass dieses auch benutzt wird.

SASS

Um SASS zu verwenden muss zuerst der Sass-Loader & node-sass installiert werden. Weiters wird das Plugin mini-css-extract-plugin benötigt um die Datei in eine externe CSS Datei umzuwandeln.

Übung 10

Images

Zur Verarbeitung von Biuldern wird der File-Loader und der Url-Loader benötigt. Standardmäßig bindet Webpack Bilder inline ein. Es kann aber in der Config Datei auch bestimmt werden ab welcher größe Bilder inline eingebunden werden sollen. Das inline Einbinden von Bildern ist gut für die Performance, weil man dadurch HTTP-Request spart, aber es ist nicht immer sinnvoll und es ist auch gerade nicht sinnvoll bei sehr großen Bildern.

Übung 11