Frontend Tooling

Inhalt

Bundler

Was ist ein Bundler?

Bundler konzentrieren sich auf das Bündeln und Optimieren des Codes der Applikation. Sie helfen dabei, verschiedene Module und Dateien zu verwalten und zusammenzuführen, wodurch Funktionen wie Code-Splitting, Tree Shaking und Lazy Loading ermöglicht werden. Dies ist besonders nützlich in der modernen Webentwicklung, wo Anwendungen oft aus zahlreichen Modulen oder Komponenten bestehen, die in separaten Dateien geschrieben sind. Bundler analysieren die Abhängigkeiten zwischen diesen Modulen und generieren optimierte Bündel, die von Browsern effizient geladen werden können.

Bis heute ist der am weitesten verbreitete Bundler: Webpack. Neuere Bundler wie Vite oder Turbopack sind aber auf dem Vormarsch. Vite wird dabei schon sehr häufig eingesetzt und viele Entwickler sind von der Performance begeistert.

Webpack

> *At its core, webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it internally builds a dependency graph from one or more entry points and then combines every module your project needs into one or more bundles, which are static assets to serve your content from.*
> *Source: Webpack*

Quickstart

Installation

*Ihr müsste diesem Guide nicht folgen, da ich bereits ein Repository vorbereitet habe. Ihr könnt direkt das **Webpack Demo** klonen.*

Damit wir Webpack nutzen können, müssen wir `webpack` und `webpack-cli` lokal installieren. Mit Webpack version 5.0 muss keine Konfigurationsdatei angegeben werden.

```
mkdir webpack-demo
cd webpack-demo
npm init -y
npm install webpack webpack-cli --save-dev
```

Erstellung der Filestruktur

```
/webpack-demo
|-- /dist
|   |-- index.html
|-- /src
```

```
|   |-- index.js
|-- package.json
```

```
// src/index.js
function component() {
  const element = document.createElement('h1');
  element.innerHTML = ['Hello', 'webpack'].join(' ');
  return element;
}

document.body.appendChild(component());
```

```
<!-- dist/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Webpack Practice</title>
  <script defer src="./main.js"></script>
</head>
<body>
</body>
</html>
```

Ausführen des Tasks

Beim Ausführen von Webpack ohne Konfigurationsdatei wird standardmässig `src/index.js` als Entry (Input) genommen. Die Output-Datei wird anschliessend als `dist/main.js` gespeichert.

```
npx webpack
```

Anschliessend kann das `dist/index.html` im Browser geöffnet werden. Wenn alles funktioniert hat, sollte ein 'Hello webpack' ersichtlich sein.

Konfigurationsdatei

Die Konfigurationsdatei für Webpack hat normalerweise `webpack.config.js` als Dateiname. Es kann aber auch irgend ein Dateiname verwendet werden (falls der Dateiname vom default abweicht, muss dies beim Ausführen von Webpack noch als Parameter angeben werden).

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

module.exports = {
  entry: './src/index.js', // entry/input
  output: {
    filename: 'main.js', // output filename
    path: path.resolve(__dirname, 'dist'), // output directory
```

```
  },
};
```

Webpack Loaders

Ein Loader definiert, was webpack mit einem gewissen Filetype machen soll.

Beispiel `babel-loader`

Installieren der genutzten Packages

```
npm install babel-loader @babel/core @babel/preset-env --save-dev
```

Konfigurieren vom Loader

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

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'main.js',
    path: path.resolve(__dirname, 'dist'),
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /(node_modules)/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: ['@babel/preset-env']
          }
        }
      }
    ]
  }
};
```

> ***Das Wichtigste in Kürze***
>
> - *Wenn ein bestimmter Dateityp von Webpack unterstützt werden soll, muss der entsprechende Loader installiert und konfiguriert werden.*
> - *Bei der Einrichtung von Projekten mit Angular CLI oder `create-next-app` muss die Webpack-Konfiguration in der Regel nicht manuell erstellt werden.*

Weitere Bundler

- Vite
- turbopack
- rollup.js

- parcel

Hilfreiche Links

- Webpack Website
- Getting starteg Guide
- Webpack Loaders

Nützliche Plugins/Loaders & Packages

- SASS/SCSS (CSS Präprozessoren) → CSS Asset Building
- PostCSS (CSS Postprozessor) → Advanced CSS Asset Building
- Babel → JavaScript Asset Building / Transpiling
- ⭐ esbuild → JavaScript Asset Building / Transpiling
- ⭐ swc → JavaScript Asset Building / Transpiling

SASS/SCSS (CSS Präprozessoren)

SASS/SCSS ist ein CSS-Präprozessor. Er ermöglicht eine vereinfachte Schreibweise von CSS und bietet viele nützliche Funktionen, die das Verfassen von CSS erleichtern und dessen Wartbarkeit verbessern. SCSS ist eine Variante von SASS, die eine etwas andere Syntax aufweist. Sowohl SASS als auch SCSS bieten jedoch dieselben Funktionen, und die meisten Entwickler verwenden die SCSS-Syntax.

Beispiel

```scss
/* SCSS */
$color-main: #73c92d;

a {
  color: $color-main;
}

.highlight {
  background-color: $color-main;
}
```

```css
/* CSS */
a {
  color: #73c92d;
}

.highlight {
  background-color: #73c92d;
}
```

Website: https://sass-lang.com/
Loader: https://webpack.js.org/loaders/sass-loader/

Weitere CSS-Präprozessoren

- LESS
- Stylus

PostCSS

PostCSS ist ein Loader, der in Kombination mit zusätzlichen Plugins genutzt werden kann.
Er wird sehr oft genutzt, da er anhand seiner Plugins sehr viele Transformationen unterstützt.

Beispiel (Autoprefixr)

```
.box {
  border-radius: 1em;
}
```

Wird zu

```
.box {
  -webkit-border-radius: 1em;
  -moz-border-radius: 1em;
  -ms-border-radius: 1em;
  border-radius: 1em;
}
```

Oft genutzte Plugins:

- Autoprefixr → Legacy Browser support
- Stylelint → QA (CSS Linter)
- PostCSS CSS Variables → Legacy Browser support

Website: https://postcss.org/
Loader: https://webpack.js.org/loaders/postcss-loader/

Babel

Babel wird verwendet, um JavaScript zu transpilieren und zu transformieren.
Es ermöglicht die Verwendung der neuesten ECMAScript-Standards und "übersetzt" diese in ältere Standards, damit unser JavaScript auch in älteren Browsern funktioniert.

Website: https://babeljs.io/
Loader: https://webpack.js.org/loaders/babel-loader/

Hilfreiche Links

- Liste von Loader

Practice 🔥

Klone das Repository der Demo **Webpack Demo** als Startpunkt.
Die Zeit reicht nicht aus um alles umzusetzen. Wähle das, was Dich am meisten interessiert

- [ ] Analysiere das `package.json` um zu sehen welche npm scripts das es gibt
- [ ] Beobachte die Unterschiede im generierten Bundle wenn du `npm run build` bzw. `npm run build:prod` ausführst
- [ ] Aktiviere den `style-loader` & `css-loader` in der Webpack-Konfiguration und importiere das CSS im `index.js`, damit die Styles verwendet werden

- [ ] Aktiviere den `babel-loader` in der Webpack-Konfiguration, sodass unser JavaScript zu einer älteren Version transpiliert wird (schau Dir das generierte JavaScript am Schluss an)
- [ ] Aktiviere Source-Maps, damit wir unsere originalen JavaScript-Dateien im Browser ansehen können (achte nach dem Aufstarten des Dev-Server, von welchen Dateien die `console.log()` ausgeführt werden)
- [ ] Versuche, einen anderen Transpiler als Babel einzusetzen (z.B. esbuild-loader)

Zeit: ~ 10 min

SASS/SCSS Deepdive

SCSS ist ein *Superset* von CSS und wird zu CSS transpiliert. In gewisser Weise kann es ein wenig mit TypeScript verglichen werden. Es ist nicht an die Beschränkungen von reinem CSS gebunden und ermöglicht das Schreiben von effizienterem und wartbarem CSS.

Variablen

```scss
/* SCSS */
$color-main: #73c92d;

a {
  color: $color-main;
}

.highlight {
  background-color: $color-main;
}
```

```css
/* CSS */
a {
  color: #73c92d;
}

.highlight {
  background-color: #73c92d;
}
```

`@mixin`

```scss
/* SCSS */
@mixin visuallyhidden() {
  border: 0;
  clip: rect(0 0 0 0);
  height: 1px;
  margin: -1px;
  overflow: hidden;
  padding: 0;
  position: absolute;
  width: 1px;
```

```
}

.skiplink {
  @include visuallyhidden;
}

.hidden-title {
  @include visuallyhidden;
}
```

```
/* CSS */
.skiplink {
  border: 0;
  clip: rect(0 0 0 0);
  height: 1px;
  margin: -1px;
  overflow: hidden;
  padding: 0;
  position: absolute;
  width: 1px;
}

.hidden-title {
  border: 0;
  clip: rect(0 0 0 0);
  height: 1px;
  margin: -1px;
  overflow: hidden;
  padding: 0;
  position: absolute;
  width: 1px;
}
```

Mit Parameter

```
/* SCSS */
@mixin triangle($direction: "down", $color: #000, $size: 1em, $selector: "after") {
  &::#{$selector} {
    height: 0;
    width: 0;
    content: "";
    position: absolute;
    border: $size solid transparent;

    @if $direction == "up" {
      border-bottom-color: $color;
    }

    @if $direction == "down" {
      border-top-color: $color;
```

```
    }
  }
}

.expand {
  @include triangle;
}

.collapse {
  @include triangle("up");
}
```

```
/* CSS */
.expand::after {
  height: 0;
  width: 0;
  content: "";
  position: absolute;
  border: 1em solid transparent;
  border-top-color: #000;
}

.collapse::after {
  height: 0;
  width: 0;
  content: "";
  position: absolute;
  border: 1em solid transparent;
  border-bottom-color: #000;
}
```

Mit Content Block

```
/* SCSS */
@mixin breakpoint($size) {
  @media (min-width: $size) {
    @content;
  }
}

.element {
  background: blue;

  @include breakpoint(1000px) {
    background: red;
  }
}
```

```css
/* CSS */
.element {
  background: blue;
}

@media(min-width: 1000px) {
  .element {
    background: red;
  }
}
```

`@extend`

```scss
/* SCSS */
%box {
  padding: 1em;
  border: 1px solid #ccc;
}

.message {
  @extend %box;
  width: 100%;
  color: #333;
}

.success {
  @extend .message;
  border-color: green;
}

.error {
  @extend .message;
  border-color: red;
}
```

```css
/* CSS */
.message,
.success,
.error {
  padding: 1em;
  border: 1px solid #cccccc;
  width: 100%;
  color: #333;
}

.success {
  border-color: green;
}
```

```
.error {
  border-color: red;
}
```

Caveats

Position des zu extendenden Selektors ist relevant für ein generiertes Resultat.
Jedes vorkommen wird extended:

```
/* SCSS */
.message + .message {
  margin-bottom: .5em;
}

.message-error {
  @extend .message;
}
```

```
/* CSS */
.message + .message,
.message-error + .message-error,
.message + .message-error,
.message-error + .message {
  margin-bottom: .5em;
}
```

Einschub: `@mixin` vs `@extend` 👀

Es ist wichtig, das generierte CSS im Blick zu behalten:

- `@mixin`:
  - Einfach wartbar, Resultat vorhersehbar
  - Resultat kann sehr redundant sein. Aber durch die gzip-Komprimierung ist dies nicht so schlimm.
- `@extend` (mit oder ohne %placeholder):
  - Reduktion des generierten Codes
  - Resultat nicht immer vorhersehbar, funktioniert nicht innerhalb von Media-Queries

Strukturierung durch Verschachtelung

```
/* SCSS */
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  a {
    display: block;
```

```
    text-decoration: none;
  }
}
```

```
/* CSS */
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}

nav a {
  display: block;
  text-decoration: none;
}
```

Verschachtelung: "Parent-Selektor"

```
/* SCSS */
a {
  color: black;

  &:hover,
  &:focus {
    color: darkred;
  }
}

.headline {
  margin-top: 5em;

  .sidebar & {
    margin-top: 2em;
  }
}
```

```
/* CSS */
a {
  color: black;
}

a:hover,
a:focus {
  color: darkred;
}

.headline {
  margin-top: 5em;
}
```

```scss
.sidebar .headline {
  margin-top: 2em;
}
```

Partials/Imports

Durch Partials kann eine Modularisierung erreicht werden. Man kann seine SCSS-Dateien aufteilen und sie nach Bedarf importieren.
Zum Beispiel ist es üblich, Mixins, Platzhalter und Variablen in separate Partials aufzuteilen.

```
/styles
|-- /modules
|   |-- _module1.scss
|   |-- _module2.scss
|   |-- _module3.scss
|-- /settings
|   |-- _variables.scss
|   |-- _functions.scss
|   |-- _mixins.scss
|-- main.scss
```

```scss
/* main.scss */
// Settings
@import "settings/variables";
@import "settings/functions";
@import "settings/mixins";

// Modules
@import "modules/module1";
@import "modules/module2";
@import "modules/module3";
```

Operationen

```scss
/* SCSS */
.demo {
  height: 20rem + 5rem;
  background: rgba(#f00, .5);
}
```

```scss
/* CSS */
.demo {
  height: 25rem;
  background: rgba(255,0,0,0.5);
}
```

Funktionen

```scss
/* SCSS */
$baseFont: 16px;

@function pxToRem($px, $ref: $baseFont) {
  @return toRem(toPx($px) / toPx($ref));
}

@function toRem($val) {
  @return ($val + 0rem);
}

@function toPx($val) {
  @return ($val + 0px);
}

.example {
  width: pxToRem(480);
  height: pxToRem(60);
}
```

```css
/* CSS */
.example {
  width: 30rem;
  height: 3.75rem;
}
```

Loops

```scss
/* SCSS */

// List of all link icons
$icons: close, rss, document;

.link {
  border: 1px solid #444;

  // Generate class for each icon
  @each $icon in $icons {
    &.var_#{$icon} {
      /* ... */
      @if $icon == close {
        /* ... */
      }
    }
  }
}
```

```
/* CSS */
.link {
  border: 1px solid #444;
}

.link.var_close {
  /* ... */
}

.link.var_rss {
  /* ... */
}

.link.var_document {
  /* ... */
}
```

Maps

```
/* SCSS */
$colors: (primary: red, secondary: blue);

button {
  color: map-get($colors, primary);

  &:hover,
  &:focus {
    color: map-get($colors, secondary);
  }
}
```

```
/* CSS */
button {
  color: red;
}

button:hover,
button:focus {
  color: blue;
}
```

Interpolation

```
/* SCSS */
$class: foo;
$property: color;
$value: red;
```

```
.#{$class} {
  #{$property}: $value;
}
```

```
/* CSS */
.foo {
  color: red;
}
```

Practice 🔥

Öffne diesen **SASS/SCSS Playground** als Startpunkt.

- [ ] Schreibe einen SASS/SCSS Code, damit du das folgende CSS erhältst (achte darauf, dass das geschriebene SCSS möglichst wartbar sein sollte):

```
.link.var_error {
  color: firebrick;
}
.link.var_success {
  color: olivedrab;
}
.link.var_info {
  color: lightblue;
}
.link.var_organic {
  color: tomato;
}

a {
  color: blue;
  text-decoration: none;
}
a:hover {
  text-decoration: underline;
}
a.important {
  color: red;
  font-weight: 700;
}
```

Zeit: ~ 10 min

▶ **Solution**