Anleitung zum Aufsetzen eines Node.js Projekts

Um ein Node.js Projekt aufzusetzen, muss man zunächst eine Distribution von Node.js auf https://nodejs.org herunterladen. In diesem Beispiel benutze ich `Node.js v12.15.0` (`npm, npx v6.14.9`). Nachdem die Kommandos `node`, `npm` und `npx` verfügbar sind, kann mit dem Projekt anfangen.

Initialisierung

Zunächst muss man das Projekt in einem neuen Ordner initialisieren. Dazu benutzt man das Kommando

```
npm init
```

Es erfolgt eine Abfrage der Projektdaten, wie z.B. Name, Version usw. Am Ende wird eine Datei namens `package.json` erstellt. Sie ist die Hauptdatei eines Node.js Projekts.

Abhängigkeiten hinzufügen

Webpack

Um Webpack als Abhängigkeit hinzuzufügen, führt man folgendes Kommando aus:

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

Ein neuer Ordner namens `node_modules` wird erstellt. In diesem befinden sich alle Abhängigkeiten. Die Namen der Abhängigkeiten (in diesem Fall `webpack` und `webpack-cli`) werden außerdem in der zentralen `package.json` Datei gespeichert. In einer anderen neuen Datei namens `package-lock.json` wird ein genauer Baum von `node_modules` gespeichert, damit zukünftige Installationen genau die gleichen Dateien generieren. Die Option `--save-dev` bewirkt, dass die Abhängigkeit als Entwickler-Abhängigkeit deklariert wird, wodurch diese nur in der Entwicklungsumgebung verfügbar gemacht wird. Nur Abhängigkeiten, die ohne diese Option hinzugefügt wurden, können später auch in der fertigen Distribution benutzt werden.

TypeScript

Parallel dazu wird der TypeScript Kompilierer mit folgendem Kommando installiert:

```
npm install typescript --save-dev
```

Genaueres ist auf der offiziellen TypeScript Seite zu finden: https://www.typescriptlang.org/download.

Um eine Schnittstelle zwischen Webpack und TypeScript zu haben, muss man zusätzlich noch das Paket `ts-loader` hinzufügen.

```
npm install ts-loader --save-dev
```

Webpack Konfigurieren

Um Webpack zu konfigurieren, kann man sich die offizielle Anleitung von https://webpack.js.org/guides/getting-started/. Grundlegend erstellt man sich erst einmal eine Konfigurationsdatei namens `webpack.config.js`, in welcher das Verhalten von Webpack angepasst wird. In diese Datei schreibt man folgenden Quelltext:

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

module.exports = {
  entry: './src/index.ts',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
};
```

Webpack betrachtet nun die `index.ts` Datei im `./src/` Verzeichnis als Startpunkt des Programms. Aufgrund dieser Datei wird die gebündelte Datei `bundle.js` im `./dist/` Verzeichnis generiert.

Damit `.ts` Dateien geladen werden können, muss nun die Schnittstelle `ts-loader` eingebunden werden:

```
// webpack.config.js
...
  module: {
    rules: [
      {
        test: /\.tsx?$/,
        loader: 'ts-loader'
      }
    ]
  },
  resolve: {
    extensions: [ '.ts' ],
  },
...
```

TypeScript konfigurieren

Auch dieses Mal kann man alles genau auf der offiziellen Website nachlesen: https://www.typescriptlang.org/docs/handbook/migrating-from-javascript.html. Damit TypeScript richtig funktionieren kann, muss man zunächst die Konfigurationsdatei `tsconfig.json` erstellen. In diese schreibt man:

```
// tsconfig.json
{
  "compilerOptions": {
    "outDir": "./dist",
    "allowJs": true,
    "target": "es5"
  },
  "include": [
    "./src/**/*"
  ]
}
```

Dies verweist darauf, dass der TypeScript Quelltext kompatibel mit dem ES5 Standard sein soll. Außerdem wird JavaScript Quelltext aus `.js`-Dateien einfach mit übernommen. Das Resultat soll auch im `./dist/` Verzeichnis landen und es sollen alle Dateien aus dem `./src/` Verzeichnis gesichtet werden.

Quelltext - Anfang

Nun muss nur noch die `./src/index.ts` Datei erstellt werden. In diese kommt später der Quelltext, welcher gebündelt werden soll. Für den Anfang kann man einfach Folgendes schreiben:

```
// ./src/index.ts
console.log("Hello World");
```

Kompilieren

Den Quelltext kann man mit dem Kommando

```
npx webpack --config webpack.config.js
```

kompilieren lassen, wodurch im `./dist/` Verzeichnis die `bundle.js` Datei auftaucht.

Quelltext - HTML

Um eine `index.html` Datei in der produzierten Distribution anzufertigen, erstellt man diese z.B. einfach im `./src/` Verzeichnis. Diese schreibt man wie eine ganz normale HTML Datei, nur, dass ein Skript-Tag vorhanden sein muss, der auf die später generierte `bundle.js` Datei referenziert.

```
<!-- index.html -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Hello World</title>
  </head>
  <body>
   <script src="bundle.js"></script>
  </body>
</html>
```

Damit diese Datei von Webpack geladen werden kann, muss man eine weitere Schnittstelle namens `file-loader` installieren.

```
npm install file-loader --save-dev
```

Außerdem muss man diese für HTML Dateien einbinden:

```
// webpack.config.js
...
      {
        test: /\.html$/,
        loader: 'file-loader',
        options: {
          name: '[name].[ext]'
        }
      },
...
```

Nun muss man die HTML nur noch wie folgt einbinden:

```
// ./src/index.ts
import './index.html';
...
```

Nun wird folgender Dateibaum generiert:

```
dist
├── index.ts
└── index.html
```

Quelltext - CSS

Um CSS einzufügen, kann man das Paket `css-loader` in Verbindung mit dem Paket `style-loader` für Webpack benutzen:

```
npm install css-loader style-loader --save-dev
```

Diese müssen auch wieder eingebunden werden:

```
// webpack.config.js
...
      {
        test: /\.css$/,
        use: [
          'style-loader',
          'css-loader',
        ]
      },
...
```

Schließlich kann man eine CSS-Datei, wie z.B. `./src/style.css` erstellen.

```
/* ./src/style.css */
body {
    background: green;
}
```

Wie auch schon bei der HTML-Datei, muss man die CSS-Datei importieren, damit sie von Webpack gefunden wird:

```
// ./src/index.ts
import './style.css';
```

Der CSS-Quelltext wird in die gebündelte `bundle.js` Datei in Form von JavaScript-Quelltext verpackt. Auf diese Weise kann Webpack abermals optimieren.

Erstellen eines NPM Skripts

Um nicht immer das Lange Kommando zum Kompilieren einzutippen, kann man ein NPM Skript definieren:

```
// package.json
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "private": true,
  "scripts": {
    "build": "webpack --config webpack.config.js"
  },
  "author": "Lukas Leicher",
  "license": "ISC",
  "devDependencies": {
    "css-loader": "^5.0.1",
    "file-loader": "^6.2.0",
    "style-loader": "^2.0.0",
    "ts-loader": "^8.0.13",
    "typescript": "^4.1.3",
    "webpack": "^5.11.1",
    "webpack-cli": "^4.3.1"
  }
}
```

Auf diese Weise kann man auch das Kommando

```
npm run build
```

verwenden, um das Projekt zu kompilieren.

Finaler Dateibaum

```
<Projektordner>
├── src
│   ├── index.html
│   ├── index.ts
│   └── style.css
├── dist
│   ├── bundle.js
│   └── index.html
├── node_modules
├── package.json
├── package-lock.json
├── tsconfig.json
└── webpack.config.js
```