TypeScript + Webpack

Webpack ist ein Bündelungstool, das hilft den Workflow in einem Projekt zu vereinfachen. Alle Files die wir für die Entwicklung anlegen werden weboptimiert compiled und abgelegt. In unserem Fall nutzen wir die Funktionalität, um TypeScript-Files in JavaScript-Files zu compilen und Module zu erstellen, die wir exportieren und im jeweiligen TypeScript-File includieren. Jeder geschriebene TypeScript Code soll in einem zentralen JS-File gebündelt werden, so dass wir nur noch dieses in unserem HTML inkludieren müssen.

Rot: Konsoleneingaben

Blau: Wichtige Files

Lila: Wichtige Ordner

Grün: JSON-Attribute

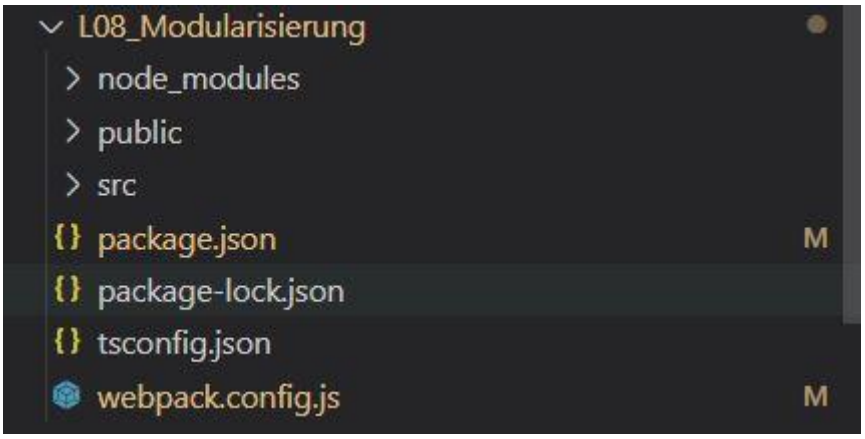
Setup

1. Konsoleneingabe (Im jeweiligen Projekt): **tsc –init** (Damit erstellen wir eine **tsconfig.json**)
    a. Folgende Attribute sollten angepasst werden
        i. **“target“: “es6“**
        ii. **“module“: “es2015“**
        iii. **“sourceMap“: “true“** (Für das Debuggen)

2. Konsoleneingabe (Im jeweiligen Projekt): **npm-init** (Damit erstellen wir eine package.json)
    a. (Die darauffolgenden Fragen in der Konsole entsprechend beantworten)

3. Konsoleneingabe (Im jeweiligen Projekt):
    a. **npm install webpack** (Damit machen wir unser Projekt für webpack kompatibel)
    b. **npm install webpack-cli** (Damit können wir mit webpack über die Kommandozeile interagieren)
    c. Es ist von Vorteil wenn wir alles in die Dev-Depandencies abspeichern (**package.json** und **package-lock.json**). Das erreichen wir in dem wir nach jeden vorangegangenen Befehl **-D** schreiben
    d. Durch diese Befehle erstellen wir einen Ordner node_modules, in dem alle installierten Depandencies abgelegt werden. Diesen sollte man nicht manuell bearbeiten. Außerdem macht es Sinn ein .gitignore-File anzulegen und diesen Ordner dort zu vermerken, da sonst das pushen in ein Repo zur Qual wird.
    e. Zusätzlich wird ein package-lock.json-File angelegt
    f. **npm install ts-loader**
        i. Auch wenn man TypeScript global installiert hat sollte man es noch einmal lokal im jeweiligen Projekt installieren, um die Funktionalität zu gewährleisten

4. Folgende Ordnerstruktur wird für die Entwicklung empfohlen



   a. Ordner: **src** (Hier liegen die Files in denen wir coden/entwickeln)
      i. **Main.ts**
   b. Ordner: **public** (Ich würde hier eher **dist** bevorzugen, da ich es so kenne. Dies hat auf die Funktionalität jedoch keinen Einfluss)
      i. **Index.html** (Hier bundle.js inkludieren)
   c. Datei: **webpack.config.js** mit folgendem Code:

```
const path = require('path'); //Absoluter Pfad, in dem das bundle.js abgelegt werden soll - Funktioniert nur mit Node.js

module.exports = {
    mode: 'development', //Damit werden wir eine nervige Warnung in der Kommandozeile los, die bei npm run build auftaucht
    devtool: 'eval-source-map', //Hier gibt man an welche Art von Source Maps man haben will. eval = gut für die Entwicklung
    entry: './src/Main.ts', //Mit diesem File fängt der Compiler an
    module: {
        rules: [ //Mit folgender Regel geben wir an, dass wir ts-files in js-files compilieren wollen
            {
                test: /\.ts$/, //Es wird getestet ob das jeweilige ts-file valide ist
                use: 'ts-loader', //Falls ja, wird der ts-loader ausgeführt um es zu compilieren
                include: [path.resolve(__dirname, 'src')] //Hier geben wir an wo unsere ts-files abgelegt sind
            }
        ]
    },
    resolve: { //Nur nötig wenn man Module bauen will
        extensions: [ //Welche extensions können aufgelöst werden
            '.ts',
            '.js'
        ]
    },
    output: {
        publicPath: 'public', //Relativer Pfad, der dem Dev-Server mitteilt, wo er seinen compilierten Code eintragen soll
        filename: 'bundle.js', //Name es JS-Files in das alles compiliert wird
        path: path.resolve(__dirname, 'public') //Hiermit erstellen wir den absoluten Pfad __dirname: abs. Pfad zur webpack.config
    }
}
```

5. package.json
   a. In der Property: **„scripts“** sollte man folgende property einfügen: **“build“: “webpack“**. Somit können wir über die Kommandozeile mit: **npm run build** webpack für uns alles compilieren lassen
   b. Folgende property in **„scripts“** ermöglicht es uns einen live-server zu starten: **“serve“ : “webpack-dev-server“**
      i. Dazu ist die Konsoleneingabe: **npm install webpack-dev-server -D** nötig
      ii. Damit der compilierte code in die **bundle.js** eingefügt, ist folgender Eintrag in der output-property der **webpack.config.json** nötig : **“publicPath“ : “Public“.** Allerdings muss man die Seite immer noch manuell refreshen.

iii. Alternativ kann man auch die VS-Code Extension: „Live Server“ nutzen. Dann muss man allerdings nach jeder Änderung **npm run build** laufen lassen

6. Modules
    a. Wir können nun im Order: **src** neue ts-files anlegen und diese exporten: **export...**
    b. Wollen wir das file nun in ein anderes importen dann benötigen wir folgenden Code: **import {nameDerKlasseOderMethode} from ‘/.nameDesFiles‘**

Das Ganze habe ich von folgendem Youtube-Channel geklaut:

https://www.youtube.com/watch?v=sOUhEJeJ-kI&list=PL4cUxeGkcC9hOkGbwzgYFmaxB0WiduYJC&index=1

Fette Props an „The Net Ninja“. Ein aktuelles Tutorial zu Webtechnologien, bei dem noch alle Console Commands funktionieren findet man selten auf Youtube.