HTML5-Tutorium: JavaScript: Hello World 06

Wechseln zu:Navigation, Suche

Dieser Artikel wird derzeit von einem Autor gründlich bearbeitet. Die Inhalte sind daher evtl. noch inkonsistent.

Dieser Artikel erfüllt die GlossarWiki-Qualitätsanforderungen **nur teilweise**:

Korrektheit: 3 (zu größeren Teilen überprüft)	**Umfang**: 4 (unwichtige Fakten fehlen)	**Quellenangaben**: 3 (wichtige Quellen vorhanden)	**Quellenarten**: 5 (ausgezeichnet)	**Konformität**: 3 (gut)

Vorlesung MMProg

Inhalt | Teil 1 | Teil 2 | Teil 3 | Teil 4

Musterlösung: `index.html`, (SVN-Repository)

Inhaltsverzeichnis

1 Anwendungsfälle (Use Cases)

Gegenüber dem vierten Teil des Tutoriums ändern sich die die Anwendungsfälle nur geringfügig. Das Begrüßungsformular soll *zweimal* im Browser angezeigt werden, einmal auf Deutsch und auf Englisch. Beide Begrüßungsformulare sollen unabhängig voneinander funktionieren.

Ein Grundprinzip der Programmierung lautet Don't repeat yourself (DRY). Sie könnten nun einfach den vorhandenen Code duplizieren und entsprechend anpassen. Aber genau das würde gegen das Prinzip „Keep your code DRY“ verstoßen. Code-Duplikation kann mit Hilfe von Klassen vermieden werden. Sie stellen eine Art „Blaupause“ dar, um Objekte mit ähnlichen Eigenschaften ganz einfach erstellen zu können.

2 Erstellen eines neuen Projektes

Erzeugen Sie wie im vierten Teil des Tutoriums ein neues Projekt, diesmal allerdings mit dem schönen Namen `HelloWorld05`“ und speichern Sie dieses wieder in Ihrem SVN-Repository (Achtung: `node_modules` muss existieren und auf der Ignore-Liste stehen).

Kopieren Sie nun aus Ihrer Lösung des vierten Teil des Tutoriums die Ordner `src` und `web` (jeweils samt allen darin enthaltenen Ordnern und Dateien) sowie die Dateien `gruntfile.js`, `package.json`, `package-lock.json` und `webpack.config.js` fügen Sie sie ins Wurzelverzeichnis des fünften Teils ein.

Weisen Sie WebStorm wiederum an, die Inhalte der Ordner `node_modules` und `web/js` nicht auf Syntaxfehler zu überprüfen. Gegebenenfalls müssen Sie auch wieder die Dateien `gruntfile.js` ... `webpack.config.js` unter die Versionsverwaltung von SVN stellen (`Add to VCS`).

Ändern Sie in den Dateien `package.json` und `web/index.html` die Versionsnummern, die Beschreibung und gegebenenfalls den Pfad zum Repository Ihres Projektes.

Zu guter Letzt öffnen Sie das WebStorm-Terminal und führen den Befehl `npm install`aus. Damit werden dieselben Node.js-Pakete installiert wie im vierten Teil des Tutoriums.

Löschen Sie die Datei `web/js/app.bundle.js` und rufen Sie anschließend `grunt` auf, um diese Datei erneut zu erzeugen. Wenn Sie jetzt die Datei `index.html` im Browser öffnen, sollte die Web-Anwendung so wie in Teil 4 der Aufgabe funktionieren.

3 Verfeinerung der Webpack-Umgebung

Zunächst benötigen Sie ein paar weitere Node.js-Module

```
npm install --save-dev es6-autobind
npm install --save-dev node-sass sass-loader extract-text-webpack-plugin
```

Das Modul `es6-autobind` wird im Zusammenhang mit der Definition der Klasse `Greet` benötigt . Daher wird es erst später besprochen.

Im vierten Teil des Tutoriums wurde die Datei `initial.css` nicht von Webpack verwaltet, sondern direkt in das Verzeichnis `web/css` eingefügt. Damit liegt auch die Verantwortung für die Komprimierung dieser Datei beim Programmierer. Das soll sich ändern. Sie erweitern die Webpack-Konfiguration so, dass künftig auch diese Datei unter der Kontrolle von Webpack steht.

3.1 SCSS

Anstelle von CSS-Dateien verwenden Sie künftig SCSS-Dateien. Das ist zunächst einmal nicht weiter schwer, das jede CSS-Datei automatisch auch eine SCSS-Datei ist. Bei der Verwendung von CSS können Sie allerdings das Prinzip „Don't repeat yourself“ (DRY) nicht beachten. Sie müssen für verschiedene Elemente ständig Informationen wiederholen (gewünschter Font, Hintergrundfarbe, Größenangaben etc.). Deshalb gibt es mehrere Projekte wie z. B. LESS und Sass , die den CSS-Standard erweitern, um dieses Problem zu vermeiden. Sass verwendet allerdings eine ganz andere Syntax als CSS. Allerdings unterstützt das Sass-Projekt auch die Sprache SCSS, die CSS einfach um

weitere syntaktische Elemente ergänzt. Zum Beispiel kann man in SCSS Konstanten definieren:

```
$background-color:  #C5EFFC;
```

Und nun kann man in der CSS-Datei anstelle von `#C5EFFC` stets `$background-color` schreiben, um die Hintergrundfarbe von bestimmten Elementen zu definieren. Wenn man nun dieses Farbe ändern will, muss das nicht mehr bei allen Elementen einzeln passieren. Es reicht, die Farbe in der Konstantendefinition abzuändern. Der CSS-Code ist nun DRY.

Leider unterstützt kein Browser SCSS direkt. Aber für was gibt es Transpiler wie Webpack, die eine SCSS-Datei automatisch in eine CSS-Datei übersetzen können.

Ersetzen Sie in Ihrem Projekt die Datei `src/css/main.css` durch drei SCSS-Dateien: `_config,scss`, `initial.scss` und app.scss.

In der Datei `_config,scss` werden alle von den anderen SCSS-Dateien benötigten Konstanten definiert. Wenn man künftig am Layout einen Zeichensatz, eine Farbe, einen Hintergrund oder Ähnliches ändern will, reicht es die entsprechenden Konstanten anzupassen und schon erzeugt Webpack automatisch aktualisiere CSS-Dateien.

Die Datei `initial.scss` enthält möglichst wenig Code. Das ist Vorlage für die Datei `initial.bundle.css`, die im HTML-Head-Bereich geladen vor dem Body-Inhalt werden soll. Die eigentlichen CSS-Informationen der Web-Anwendung werden in `app.scss` gespeichert. Die zugehörigen CSS-Anweisungen werden von Webpack wie gehabt in `app.bundle.js` integriert.

src/css/_config,scss

```
$font-family-sans:  Verdana, Helvetica, sans-serif;
$font-family-serif: "Times New Roman", Times, serif;
$background-color:  #C5EFFC;
```

Diese Datei kann nun mit Hilfe des SCSS-Befehls `@import` in die anderen beiden SCSS-Dateien eingefügt werden. Webpack (genauer gesagt das SASS-Modul von Webpack) kennt diesen Befehl und führt im beim Erstellen der zugehörigen CSS-Dateien auch aus.

Der Import-Befehl ist hier notwendig, da in der Datei `initial.scss` die Konstante `$background-color` verwendet wird. Ansonsten enthält dies Datei nur den schon bekannten CSS-Code.

src/css/initial.scss

```
@import 'config';

body
{ background-color: $background-color;
}

.hidden
{ display: none;
}
```

Die Datei `app.scss` ist deutlich umfangreicher. Daher soll der zugehörige CSS-Code ja auch erst gemeinsam mit der Datei `app.bundle.js` geladen werden. Das meiste ist üblicher CSS-Code, den Sie schon im vierten Teil des Tutoriums kennengelernt haben. Es gibt ein paar neue Elemente, wie die CSS-Klassen-Elemente `.left` und `.right`, die in der HTML-Datei dieses Tutoriums zusätzlich verwendet werden. Außerdem sehen Sie in der Datei noch ein paar weitere SCSS-Elemente. Diese werden im Anschluss an den folgenden code beschrieben.

src/css/app.scss

```
@import 'config';

html, body
{ height:           100%;
  font-size:        2vw !important;
  font-family:      $font-family-sans;
  background-color: $background-color;
}

html
{ display: table;
  width:   100%;
}

body
{ display:          table-cell;
  text-align:       center;
  vertical-align:   middle;
}

h1
{ padding-bottom: 0;
  margin-bottom:  0;
}

#section_form
{ text-align:   center;
  width:        21em;
  margin-left:  auto;
```

```
  margin-right: auto;

  h1
  { font-size:     200%;
    margin-bottom: 0.5ex;
  }
}

#section_hello
{ h1
  { font-size: 270%;
  }
}

p, label
{ font-family: $font-family-serif;
  font-size:   100%;
}

label, input, button
{ width:         10em;
  font-size:     100%;
  display:       inline-block;
  box-sizing:    border-box;
  margin-bottom: 0.5ex;
}

label
{ text-align: right;
}

@mixin panel()
{ padding: 0;
  width:   50%;
}

.left
{ float: left;
  @include panel();
}
.right
{ float: right;
  @include panel();
}
```

In diese SCSS-Datei werden folgende SCSS-Erweiterungen verwendet:

Import-Befehl: `@import 'config';`
Konstanten: `$font-family-sans`, `$font-family-serif`, `$background-color`

Verschachtelte Anweisungen: z. B. `h1` innerhalb von `#section_form`
Mixins, um eine Liste von Attributen nur einmal zu definieren und und mehreren Elemente wieder verwenden zu können.

Die verschachtelten Anweisungen strukturieren den CSS-Code deutlich besser. Es ist nicht mehr notwendig einen Selektor wie `#section_form` mehrfach zu verwenden.

Sehen Sie sich den Unterschied an. In der Datei `main.css` vom vierten Teil des Tutoriums hatten Sie folgendes geschrieben:

```
#section_form
{
  text-align:   center;
  width:        21em;
  margin-left:  auto;
  margin-right: auto;
}

#section_form > h1
{
  font-size:     200%;
  margin-bottom: 0.5ex;
}
```

Die SCSS-Variante ist deutlich strukturierter und prägnanter.

```
#section_form
{ text-align:   center;
  width:        21em;
  margin-left:  auto;
  margin-right: auto;

  h1
  { font-size:     200%;
    margin-bottom: 0.5ex;
  }
}
```

Insbesondere bei großen CSS-Dateien erweist sich diese kompaktere Schreibweise als vorteilhaft.

Ein weiteres Element, das nur SCSS bietet sind die Mixins. Sie definieren zunächst ein Mixin, das eine Folge von CSS-Attributen zusammenfasst. (Auch hier könnten wieder Verschachtelungen und andere SCSS-Erweiterungen verwendet werden!)

```
@mixin panel()
{ padding: 0;
  width:   50%;
}
```

Und nun können Sie dieses Mixin in diversen andere Elemente einbinden:

```
.left
{ float: left;
  @include panel();

.right
{ float: right;
  @include panel();
}
```

In normalen CSS-Code, müssten Sie denselben Code Non-DRY formulieren:

```
.left
{ float:   left;
  padding: 0;
  width:   50%;
}

.right
{ float:   right;
  padding: 0;
  width:   50%;
}
```

Wenn Sie hier das Layout der Panels ändern wollten (andere Breite, andere Ränder etc.), müssen Sie den CSS-Code von jedem Panel einzeln ändern. In der SCSS-Datei erfolgt die Änderung dagegen DRY: Sie ändern einfach den Mixin-Code ab.

Wenn die aktuellen Browser SCSS verstehen würden, könnte sie `initial.scss` und `app.scss` direkt vom HTML-Code aus laden. Allerdings geht das derzeit leider nicht. Somit muss Webpack angewiesen werden, CSS-Code aus den SCSS-Dateien zu erzeugen. Dies ist mit den beiden Node.js-Modulen `node-sass` und `sass-loader`, die Sie bereits installiert haben, ganz einfach möglich.

Ersetzen Sie in der Datei `webpack.config.js` folgenden Code

```
{ test:   /\.css$/,
  use:    [ 'style-loader',
            { loader: 'css-loader', options: { minimize: true } }
          ]
},
```

durch folgenden:

```
{ test:   /\.(css|scss|sass)$/,
  use:    [ 'style-loader',
            { loader: 'css-loader', options: { minimize: true } },
            'sass-loader'
          ]
}
```

Der Test wird so erweitert, dass diese Regel jetzt nicht nur für die Verarbeitung von Dateien mit der Endung `.css` zuständig ist, sondern auch für Dateien mit den Endungen `.sass` und `.scss`.

Die Liste der zugehörigen Loader wird einfach um einen Loader erweitert: `sass-loader`. Diese Liste wird von hinten nach vorne abgearbeitet. Das heißt, eine Datei wie beispielsweise `app.scss` wird erst mit Hilfe des Sass-Loaders in eine CSS-Datei umgewandelt, diese wird mit Hilfe des CSS-Loaders komprimiert und das Ergebnis dieser Datei wird mit Hilfe des Style-Loaders in die zugehörige JavaScript-Ausgabedatei integriert.

Jetzt müssen nur noch die zugehörigen JavaScript-Dateien erstellt werden.

Im `entry`-Objekt der Datei `webpack.config.js` ist bereits die Datei `src/js/app.js` eingetragen. Für Sie wird von Webpack die Datei `web/js/app.bundle.js` erzeugt.

Der Name der Zieldatei wurde im `output`-Objekt von `webpack.config.js` spezifiziert. **Achtung:** Überprüfen Sie, ob das `output`-Objekt folgendermaßen definiert wurde.

```
  output:
  { filename: '[name].bundle.js',
    path:     absolutePath('web/js')
  }
```

Im vierten Teil des Tutoriums stand der Unterpfad `js` noch im Dateinamen, was problematisch ist. Verbessern Sie gegebenenfalls das Output-Element.

Damit diese Zieldatei insbesondere den komprimierten Code der Datei `src/css/app.scss` enthält, muss `src/js/app.js` leicht modifiziert werden.

Ersetzen Sie in dieser Datei den Import-Befehl

```
import '../css/main.css';
```

durch den Import-Befehl

```
import '../css/app.scss';
```

Die Datei `web/css/main.css` gibt es ja nicht mehr; sie wurde durch `web/css/app.scss` ersetzt.

Damit auch `web/css/initial.scss` durch webpack verwaltet wird, benötigen wir eine zweite JavaScript-Datei.

src/js/initial.js

```
import '../css/initial.scss';
```

Sie sehen das richtig, die Datei `src/js/initial.js` enthält eine einzige Code-Zeile. Sie erinnern sich? Webpack dient eigentlich nur dazu JavaScript-Dateien zu packen. Also geben wir diesen Tool eine JavaScript-Datei, die die gewünschte CSS-Datei einfach mittels eine ES6-Import-Befehls importiert (ES6 = EcmaScript 6). Fügen Sie nun in das `entry`-Objekt der Datei `webpack.config.js` vor der Zeile folgende zweite Zeile ein:

```
initial: absolutePath('src/js/initial.js')
```

Beachten Sie, dass in einem JavaScript-Objekt die Attribute durch Kommas voneinander getrennt werden. Fügen Sie also an passender Stelle auch noch ein Komma ein.

Wenn Sie jetzt Grunt aufrufen, sollten zwei Dateien generiert werden: `web/js/app.bundle.js` und `web/js/initial.bundle.js`.

Löschen Sie nun den Ordner `web/css` (die darin befindliche Datei `initial.css` wird nicht mehr benötigt) und ersetzen Sie in der Datei `index.html` die Zeile

```
<link rel = "stylesheet" type = "text/css" href = "css/initial.css"/>
```

durch folgende Zeile:

```
<script type = "text/javascript" src = "js/initial.bundle.js"></script>
```

Wenn Sie jetzt die Datei `index.html` im Browser öffnen, sollte sie wieder genauso funktionieren wie zuvor.

Anmerkung: Die Datei `src/js/initial.js` könnte natürlich noch mehr machen, als einfach nur eine CSS-Datei zu laden. Sie könnte beispielsweise eine kleine Animation einblenden (die berühmte Sanduhr, nur etwas moderner), die solange läuft, bis die eigentliche Web-Anwendung, d. h. `app.bundle.js` vollständig geladen wurde und die Animation wieder beendet.

3.2 Erzeugen einer CSS-Datei

Zurzeit enthält die JavaScript-Datei `web/js/initial.bundle.js` nur eine komprimierte CSS-Datei und etwas Code, um diesen in das HTML-Dokument einzufügen. Die Google-Suche bewertet HTML-Dateien, die sauberen CSS-Code i. Allg. enthalten, höher, als HTML-Datei, die nur JavaScript-Code enthalten. Daher wäre es in diesem Fall sinnvoll, die komprimierte CSS-Datei direkt in den HTML-Code einzubinden. Das ist aber gar nicht so einfach, da Webpack keine CSS-Dateien erstellt, sondern nur JavaScript-Dateien. Aber mit einem kleine Hack funktioniert das trotzdem.

Sie haben vor auch noch die Node.js-Module `extract-text-webpack-plugin` installiert. Mit diesem Sie eine CSS-Datei aus einer JavaScript-Datei extrahieren.

Ersetzen Sie zunächst in der Datei `webpack.config.js` den Test

```
test:   /\.(css|scss|sass)$/,
```

durch

```
test:   /app\.(css|scss|sass)$/,
```

Damit erreichen Sie dass diese Regel jetzt nur noch für Dateien funktioniert, die auf `app.css` oder `app.scss` oder `app.sass` enden. Die Datei `src/css/app.scss` in die Datei `src/js/app.bundle.js` integriert werden.

Laden Sie zunächst die ExtractTextPlugin-Erweiterung, indem Sie in die Datei `webpack.config.js` folgende zweite Zeile einfügen:

```
const ExtractTextPlugin = require('extract-text-webpack-plugin');
```

Für die Datei `src/css/initial.scss` fügen Sie nun in die Liste mit den Regeln eine weitere Regel ein (Komma nicht vergessen):

```
{ test:   /initial\.(css|scss|sass)$/,
  use:   ExtractTextPlugin.extract
         ({fallback: "style-loader",
           use:      [{ loader: 'css-loader', options: { minimize: true }
},
                      'sass-loader'
                     ]
         })
}
```

Diese benutzt das zuvor erwähnte Plugin, um die komprimierte CSS-Datei aus der Datei `src/js/initial.bundle.js` zu extrahieren. Das erledigt sie aber nur fast. Man muss dem Plugin noch mitteilen wohin Sie die CSS-Datei speichern soll.

Dazu müssen Sie die bislang leere Plugin-Liste am Ende der Datei

```
plugins:
[]
```

mit Inhalt füllen:

```
plugins:
[ new ExtractTextPlugin("../css/[name].bundle.css") ]
```

Führen Sie nun `grunt` aus. Es sollte die Datei `web/css/initial.bundle.css` erzeugt worden sein.

Nun müssen sie in der Datei `index.html` die Zeile

```
<script type = "text/javascript" src = "js/initial.bundle.js"></script>
```

wieder durch eine CSS-Link-Zeile ersetzen (da ja in der Datei `initial.bundle.js` kein sinnvoller Inhalt mehr enthalten ist; sie wurde durch das Extract-Text-Plugin „ausgelutscht“).

```
<link rel = "stylesheet" type = "text/css" href =
"css/initial.bundle.css"/>
```

Sie könnten nun noch die überflüssige Datei `web/js/initial.bundle.js` von webpack wieder entfernen lassen. Wie das geht entnehmen Sie bitte der Seite https://stackoverflow.com/questions/37408873/delete-or-not-create-a-file-for-each-entry-in-webpack sowie der Musterlösung. Wichtig ist das nicht. Wenn Sie später mal eine kleine Warteanimation erstellen sollten, bräuchten Sie dieses Datei sowieso wieder.

4 JSON

Bislang haben sie den CSS-Code unter Webpack-Kontrolle gestellt und damit DRY gemacht. Als nächstes gehen wir das nächste Problem an: Ihr Code enthält Konstanten.damit verstoßene Sie gegen ein weiteres fundamentales Programmierprinzip:

Konstante Werte sollten im Allgemeinen nicht direkt in den Code eingefügt werden, sondern als konstante Werte separat definiert werden (Ausnahme: triviale Konstanten, die sich sicher nie ändern werden, wie z.B. Vergleiche mit dem Wert 0). Konstanten, die das Programmverhalten beeinflussen, sollten im Allgemeinen beim Programmstart aus einer Konfigurationsdatei ausgelesen werden.

TO BE DONE

5 Wiederverwendbarkeit

TO BE DONE

6 Quellen

1. **Kowarschick (MMProg)**: Wolfgang Kowarschick; Vorlesung „Multimedia-Programmierung“; Hochschule: Hochschule Augsburg; Adresse: Augsburg; Web-Link; 2018; Quellengüte: 3 (Vorlesung)

Kategorien:
Multimedia-Programmierung/Tutorium
Praktikum:MMProg
HTML5-Tutorium: JavaScript: Hello World
HTML5-Beispiel
Kapitel:Multimedia-Programmierung:Beispiele