HTML5-Tutorium: JavaScript: Hello World 04

Wechseln zu:Navigation, Suche

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 WebProg

Inhalt | Teil 1 | Teil 2 | Teil 3 | Teil 4 | Teil 5 | Teil 6

Musterlösung: `index0.html`, `index1.html`, `index2.html` (Git-Repository v04a), `index.html` (Git-Repository v04b), `index.html` (Git-Repository v04c)

Inhaltsverzeichnis

1 Anwendungsfälle (Use Cases)

Gegenüber dem dritten Teil des Tutoriums ändern sich die die Anwendungsfälle nicht. Die Anwendung leistet also genau dasselbe wie zuvor.

In diesem Teil des Tutoriums geht es darum, die Anwendung besser zu strukturieren, d. h. zu modularisieren (vgl. Programmierprinzipien).

2 Modularisierung

Eine Anwendung, wie z. B. ein HTML5-Spiel, besteht aus diversen unterschiedlichen Komponenten mit ganz unterschiedlichen Aufgaben. Bei einem Web-Frontend müssen beispielsweise verschieden Komponenten (Buttons, Scrolbars, Textfelder, Datumsfelder etc.) erstellt werden. Bei einem HTML5-Spiel müssen die Spielfiguren, die Spielszenen, die Spiellogik, die Benutzereingaben, die Darstellung das Spiels im Browser etc. erstellt und verwaltet werden, wobei das Erstellen und Verwalten teilweise während der Spielentwicklung und teilweise während der Spielausführung (Runtime) erfolgt.

An der Entwicklung eines Web-Systems oder eines Spiels sind üblicherweise mehrere oder gar viele Entwickler gleichzeitig beteiligt. Oft müssen einzelnen Komponenten an neue Endgeräte oder Betriebssysteme angepasst werden. Beispielsweise muss eine Tastatursteuerung durch eine Touch- oder Gestensteuerung ersetzt werden, wenn eine Anwendung so erweitert werden soll, dass es auch auf einem Smartphone oder Tablet läuft.

Das alles ist nur machbar, wenn man die Anwendung modularisiert, d. h. in kleine, möglichst unabhängige Komponenten unterteilt. Ein großer Feind der Modularisierung sind globale Variablen.

Je mehr Dateien von unterschiedlichen Autoren erstellt werden, desto größer ist die Gefahr, dass es beim Zugriff auf globale Variablen zu Namenskollisionen kommt. Außerdem weiß niemand, welche globale Größe noch benötigt wird und welche nicht. Sicherheitshalber löscht man daher keine dieser Variablen, auch wenn sie vollkommen veraltet sind.

Daher gilt der Grundsatz: Verwende so wenig globale Größen wie möglich.

3 Erstellen eines neuen Projektes

Erstellen Sie ein neues Projekt `HelloWorld04` als Fork von `HelloWorld03` (siehe Hello-World-02-Tutorium):

```
git clone https://gitlab.multimedia.hs-augsburg.de/ACCOUNT/HelloWorld03
HelloWorld04
//git clone https://gitlab.multimedia.hs-augsburg.de/kowa/WK_HelloWorld03
HelloWorld04

## Änderungen zur Projektidentifikation vornehmen (z.B. Titel anpassen)
git commit -m "HelloWorld03 fork created"

# Neues Repository in Gitlab anlegen:
git remote -v
git remote remove origin
git remote add origin https://gitlab.multimedia.hs-
augsburg.de/ACCOUNT/HelloWorld04
git remote -v
git push --set-upstream origin master
```

4 Erstellen einer Ordnerstruktur

Grundsätzlich gilt auch bei der Programmierung: Ordnung ist das halbe Leben.

Web-Anwendungen werden sehr schnell sehr groß. Also sollte man sich eine geeignete Ordnerstruktur überlegen. Üblicherweise legt man CSS-Dateien in einen Ordner namens „`css`“ und JavaScript-Dateien in einen Ordner namens „`js`“ oder „`lib`“. Sollten es viele CSS- und/oder JavaScript-Dateien werden, legt man im entsprechenden Ordner geeignete Unterordner an.

Legen Sie im `web`-Verzeichnis Ihres Projektes folgende Ordner an:

`web/css`: Hier werden die eigentlichen CSS-Dateien der Web-Anwendung gespeichert, die von der App dynamisch nachgeladen werden sollen. In diesem Ordner könnten zur weiteren Strukturierung Unterordner angelegt werden. Allerdings ist das hier nicht notwendig. Es wird nur die Dateien „`head.css`“ und „`body.css`“ geben. **Verschieben Sie die Datei `web/main.css` in diesen Ordner.**
`web/js`: Hierher kommen die JavaScript-Dateien, die von der Anwendung benötigt werden. **Verschieben Sie die Datei `web/main.js` in diesen Ordner.**

Nehmen Sie in der Datei `web/index.html` folgende beiden Ersetzungen vor:

`main.css` → `css/main.css`
`main.js` → `js/main.js`

Wenn Sie jetzt die Datei `index.html` mittels `run` ausführen, sollte Ihre Anwendung wieder funktionieren.

Sollte dies der Fall sein, so sollten Sie die Änderungen committen und evtl. auch gleich per `git push` auf den Git-Server hochladen. **Es schadet übrigens nie, vor einen Commit oder gar einen Push noch einmal zu überprüfen, ob die Anwendung noch läuft.**

5 Die App

5.1 Laden der CSS- und der JavaScript-Dateien

Eine Web-Anwendung funktioniert nur – wie Sie bereits erfahren haben –, wenn nicht nur die HTML-Datei, sondern auch die zugehörigen CSS- und JavaScript-Dateien geladen werden. Allerdings gibt es dabei ein Problem: Die CSS- und JavaScript-Dateien werden im Laufe der Zeit immer zahlreicher und/oder größer. Diese Dateien zu laden, dauert seine Zeit.

In der dritten Version der Web-Anwendung stehen die Verweise auf diese Dateien im `head`-Bereich des Dokuments. Dieser wird vollständig geladen, bevor der `body`-Bereich eingelesen wird. (Genauer gesagt, nur die CSS-Datei wird vollständig geladen. Durch die Angabe von `async="async"` im Script-Tag wird erreicht, dass die JavaScript-Datei `main.js` geladen wird, während der Body-Bereich gerendert wird.)

Das heißt aber, dass der Browser keine Inhalte des HTML-Dokuments darstellen kann, solange er CSS-Datei (und evtl. weitere Dateien) lädt. Wenn dies zu lange dauert, verliert der Besucher die Geduld und verlässt die Seite vorzeitig.

Besser wäre es daher andersherum vorzugehen: Es wird zuerst der Body-Bereich geladen und dann die JavaScript- und die CSS-Dateien. Im Falle von JavaScript ist das durchaus sinnvoll (und durch die Angabe von `async="async"` auch machbar), aber im Falle von CSS hat das den Effekt, dass der

Browser noch keine Layout-Vorgaben erhalten hat, wenn er mit dem Rendern der Seite beginnt. Also verwendet er die browserspezifischen Defaultwerte. Das heißt, die Seite sieht zunächst ganz anders aus, als vom Designer geplant. Wenn dann die CSS-Dateien geladen wurden, wird die Seite erneut gerendert und verändert ihr Aussehen. Auch das ist verwirrend und wirkt unprofessionell.

Was also machen?

5.2 CSS-Dateien

Damit das Problem mit dem falschen Layout nicht auftritt, empfiehlt Google (PageSpeed Insights), ganz wenige, wichtige CSS-Befehle direkt – d. h. als CSS-Befehle innerhalb eines `style`-Elements – in dem HTML-Head-Bereich einzufügen. Alle anderen CSS-Anweisungen sollen erst am Ende der HTML-Datei dynamisch mittels JavaScript (oder mittels des PageSpeed-Optimization-Modul von Google) gelesen werden.

PageSpeed Insights (für Musterlösung index0.html):

> Empfehlung
>
> Ressourcen blockieren den First Paint Ihrer Seite. Versuchen Sie, wichtiges JS und wichtige CSS inline anzugeben und alle nicht kritischen JS und Stile aufzuschieben. Weitere Informationen.

Gehen Sie so vor, wie es Google empfiehlt. Genauer gesagt, gehen Sie so vor, wie es unter Multimedia-Programmierung: Best Practices beschrieben ist. In diesem und den folgenden Tutorien werden die dort beschriebenen Prinzipien berücksichtigt.

Legen Sie die Datei `src/css/head.css` an und fügen Sie folgenden (sehr kurzen!) Code ein:

```
body
{ background-color: #C5EFFC; }

.hidden
{ display: none; }
```

In der Datei `src/index.html` müssen Sie das Link-Element in Header folgendermaßen

```
<link rel="stylesheet" href="css/head.css"/>
```

anpassen. Das heißt, Sie laden im HTML-Head-Bereich nur noch die kleine CSS-Datei. Damit funktioniert Ihre Web-Anwendung derzeit nur noch rudimentär: Es gibt kaum noch CSS-Anweisungen, die berücksichtigt werden. Noch besser ist es gemäß den Google-Empfehlungen, wenn man die CSS-Regelsätze direkt in die HTML-Datei einfügt. Ersetzen Sie also das Link-Element durch folgendes Style-Element:

```
<style>
body
{ background-color: #C5EFFC; }

.hidden
{ display: none; }
</style>
```

In der Datei `src/css/main.css` können Sie nun das `.hidden`-Element und das Farbattribut `background-color: #C5EFFC;` entfernen.

Es ist sinnvoll, die Datei `main.css` in `body.css` umzubenennen: `head.css` wird im HTML-Head-Bereich vor dem Body-Bereich geladen, `body.css` soll parallel zum Body-Bereich geladen werden und dann zur Verfügung stehen, wenn der Body vollständig geladen wurde.

Leider ist CSS-Link-Element nicht möglich, `async="async"` anzugeben, wie wir dies beim Skript-Element gemacht haben, um JavaScript-Code parallel zum Body-Element zu laden. Hier hilft derzeit nur ein kleiner Hack

```
<link rel="stylesheet" href="css/body.css"
      media="none" onload="this.media='all'"
/>
```

Die Datei `body.css` wird für den Medientyp `none` geladen. Diesen Medientyp gibt es aber gar nicht. Es gibt die Typen `screen`, `print`, `speech`, `all` sowie ein paar weitere veraltete Typen. Da der Medientyp `none` vorerst nicht gebraucht wird, laden ihn moderne Browser asynchron. Sobald das HTML-Dokument vollständig geladen wurde, feuert das Ereignis `onload`. Sobald dies passiert, wird der Medientyp `none` durch `all` ersetzt. Daran erkennt der Browser, dass dies eine CSS-Datei ist, die durchaus zum Layouten der aktuellen Seite verwendet werden soll. Und das macht der Browser dann auch.

Zusammenfassung:

> In `head.css` stehen möglichst wenig CSS-Anweisungen. Sie sollten ausreichen, um die aktuelle Seite “above the fold” („über dem Zeitungsknick“) fehlerfrei zu rendern. Das heißt, es sollte alles korrekt gerendert werden, was nach dem Laden im größtmöglichen Browserfenster sichtbar ist.
> Der Inhalt der `head.css` sollte direkt in den Head-Bereich der HTML-Seite eingefügt werden.
> Die restlichen CSS-Anweisungen werden in die Datei `body.css` eingetragen.
> Dies Datei wird mittels eines Hacks asynchron, d. h. parallel zum Body-Element geladen.

5.3 `index.html`

Da die Datei `head.css` nur die Hintergrundfarbe sowie die `.hidden`-Regel enthält, sollten alle HTML-Elemente, die gerendert werden sollen, zunächst unsichtbar gemacht werden. Bei einer Web-Anwendung, wie einem Spiel, die zum Laden länger braucht, könnte man eine Sanduhr oder einen Ladebalken einfügen, um anzuzeigen, dass man noch etwas warten muss. Üblicherweise, sollte man dafür sorgen, dass das Laden einer Seite so schnell geht, dass der Benutzer gar nicht erst ungeduldig wird.

Fügen Sie die CSS-Klasse `hidden` in beide HTML-Sektions ein:

```
<section id="section_form" class="hidden">
...
</section>
<section id="section_hello" class="hidden">
...
</section>
```

Wenn man jetzt das Dokument im Browser öffnet, sieht man nichts, außer einem hellblauen Hintergrund.

Abhilfe schafft hier ein kleiner JavaScript-Befehl, der `hidden` von der Eingabe-Section `hidden-form` entfernt, sobald das Dokument vollständig geladen wurde (samt allen CSS- und JavaScript-Dateien):

```
document.getElementById('section_form').classList.remove('hidden');
```

An welche Stelle dieser Befehl eingefügt wird, erfahren Sie im nachfolgenden Abschnitt.

5.4 JavaScript-Dateien

Für JavaScript-Dateien empfiehlt Google ebenfalls, diese nicht schon zu Beginn zu laden, sondern das Laden „aufzuschieben“. Das machen wir bereits mit Hilfe des Async-Attributs.

Fügen Sie nun den zuvor erwähnten Befehl

```
document.getElementById('section_form').classList.remove('hidden');
```

als letzen Befehl in den Rumpf der Funktion `init` in der Datei `web/js/main.js` ein.

Wenn Sie jetzt die Web-App starten, sollte sie wieder funktionieren .

Was haben Sie erreicht? Solange die Anwendung geladen wird, sehen Sie nur einen blauen Hintergrund ohne Inhalt, bzw. i. Allg. einen so genannten Splash Screen. Wenn dann alles, d. h. alle CSS- und JavaScript-Dateien geladen wurden, wird die eigentliche Anwendung sichtbar.

Vergessen Sie nicht: Commit!

5.5 Modularisierung der JavaScript-Dateien

Die JavaScript-Datei `main.js` aus dem dritten Teil des Tutoriums wird in zwei Teile aufgespalten: `greet.js` und `main.js`. Die erste Datei enthält die eigentliche Begrüßungsfunktionalität, die zweite initialisiert die App, d. h. aktiviert die Begrüßungsfunktion.

Erstellen Sie die JavaScript-Datei `web/js/greet.js` und verschieben Sie die beiden Funktionen `sayHello` und `sayHelloOnEnter` (jeweils samt Funktionsrumpf :-)) in diese Datei.

Schreiben Sie vor jede der beiden Funktionsdefinitionen das Schlüsselwort `export`. Mit dieser ES-6-

Anweisung legen Sie fest, dass die beiden Funktionen von anderen ES-6-Dateien mittels eines Import-Befehls importiert werden können. (Funktionen ohne Export-Anweisung könnten von anderen ES-6-Dateien nicht importiert werden.)

In der Datei `web/js/main.js` werden jetzt diese beiden Funktionen importiert. Fügen Sie ganz am Anfang der Datei (d. h. vor der verbliebenen Init-Funktion) die folgende Import-Anweisung ein:

```
import * as greet from './greet.js';
```

Damit wird erreicht, dass alle (*) Elemente, die von der Datei `greet.js` exportiert werden, unter ihrem jeweiligen Namen im Objekt `greet` gespeichert werden. (Man könnte auch nur bestimmte Elemente von `greet.js` importieren, sofern man nicht alle benötigt.) Das heißt, innerhalb der Datei `main.js` gibt es nun die beiden Funktionen `greet.sayHello` und `greet.sayHelloOnEnter`.

Damit die Web-Anwendung wieder funktioniert, müssen Sie also in der Datei `main.js` vor jede Verwendung dieser beiden Funktionen die Zeichenfolge `greet.` einfügen.

Fügen Sie jetzt noch das Attribut `type="module"` in das Script-Element in der Datei `index.html` ein. Damit teilen Sie dem Browser mit, dass Sie die ECMAScript-6-Befehle `import` und `export` verwenden möchten. Anderenfalls hätten sie Zugriff auf alle Objekte, Konstanten, Variablen und Funktionen einer geladenen Datei, so wie die bis ECMAScript 5 üblich war. Das heißt, das Laden einer Datei konnte zu massiven Problemen führen, wenn in zwei verschiedenen Dateien unterschiedliche Objekte etc. zufälligerweise gleich benannt wurden. Um diese Probleme zu umgehen gab es zahlreiche ziemlich aufwändige Hacks. Mit dem Modul-Konzept von ECMAScript gehören diese Probleme der Vergangenheit an. Damit sollte Ihre Anwendung wieder funktionieren.

Vergessen Sie nicht, Ihr aktuelles Projekt zu committen und auf dem Git-Server zu speichern, sobald alles funktioniert.

5.6 Google-Pagespeed-Test

Wenn man die Anwendung noch einmal mit Google testet, erhält man eine neue Empfehlung

PageSpeed Insights (für Musterlösung index2.html):

> Halten Sie die Anfrageanzahl niedrig und die Übertragungsgröße gering 5 Anfragen • 5 KiB.

Das heißt, wir sollen die Dateien komprimieren: Überflüssige Kommentare, Leerzeichen und Zeilenumbrüche löschen, lange Variablen-, Konstanten- und Funktionsnamen verkürzen etc. Außerdem sollten wir die beiden Dateien `main.js` und `greet.js` wieder zu einer Datei zusammenfügen. Da dies gegen das Prinzip der Modularisierung spricht, wird dies nicht im Sourcecode, sondern automatisch mit Hilfe eines geeigneten Tools (webpack, vite, ...) realisiert (siehe Tutorium: Teil 5). Das heißt, der Sourcecode wird weiterhin modular aufgebaut. Anschließend wird der Sourcecode mit Hilfe eines [[Transpiler]s (Source-to-Source-Compiler) in eine kompakte Darstellung transformiert. Die Anzahl der Dateien, die vom Browser geladen werden müssen, wird drastisch reduziert, überflüssige Leerzeichen und Kommentare werden entfernt, Variablennamen werden durch kurze Namen ersetzt etc. Ein Transpiler kann auch die Programmiersprache ändern. Zum Beispiel kann ECMAScript 6 in ECMAScript 5 übersetzt werden. Oder man verwendet eine Sprache wie

TypeScript, CoffeeScript etc. oder sogar Java und übersetzt diese in ECMAScript, damit der Browser den Code interpretieren kann.

5.7 Weitere Modularisierung

Teilen Sie die Datei `greet.js` in zwei Dateien `greet.js` und `greet_on_enter.js` mit jeweils eines Funktion auf, um dem Single Responsibility Principle zu genügen.

Musterlösung: `index.html` (Git-Repository v04b)

5.8 Konfigurierbarkeit

Entfernen Sie programmspezifische Konstanten aus den JavaScript-Dateien und definieren Sie diese in einer Datei `config.js` oder (sehr viel besser) `config.json`, um dem Prinzip der Konfigurierbarkeit zu genügen.

Musterlösung: `index.html` (Git-Repository v04c)

6 Fortsetzung des Tutoriums

Wenn man die Vorschläge von Google beachtet, beschleunigt dies die Übertragung der Dateien vom Server zum Client, macht aber den Code unlesbar. Die Zusammenführung zweier Dateien widerspricht dem Prinzip der Modularisierung. Das Entfernen von Kommentaren, Leerzeichen und Leerzeilen ist aus Entwickler-Sicht vollkommen kontraproduktiv.

Daher hat es sich eingebürgert, modularen Code mit Kommentaren und Leerzeichen zu schreiben. Dieser wird dann, bevor er an einen Browser übergeben wird automatisch zusammengefasst und komprimiert (minifiziert).

Sie sollten nun Teil 5 des Tutoriums bearbeiten. Dort erfüllen wir die Google-Vorgaben mittels `webpack`, einem mächtiges Werkzeug zur Umwandlung von Web-Dateien. Leider ist die Konfiguration dieses Werkzeugs sehr komplex.

7 Quellen

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