Web-Entwicklung: Übungsblatt 5

Aufgabe 1: Anlegen eines npm-Moduls

Legen Sie ein Projekt in Form eines eigenen npm-Moduls mit einer entsprechenden Konfigurationsdatei an. Belegen Sie zumindest die Pflichtfelder in der Konfigurationsdatei mit sinnvollen Werten.

Aufgabe 2: Implementierung einer Demo-Anwendung

Erstellen Sie zur Demonstration eine kleine Browser-Anwendung, die das Browser-Fenster sukzessive mit einem Blindtext füllt (z.B. ein Satz pro Sekunde). Zum Generieren des Blindtextes soll das npm-Modul `lorem-ipsum` verwendet werden. Legen Sie eine einfache CSS-Datei an, die grundlegende visuelle Eigenschaften der Website festlegt.

Hinweis: Um eine Funktion wiederholt aufzurufen, können Sie die globale Funktion setInterval() einsetzen.

Aufgabe 3: Einrichten eines npm-Build-Prozesses

Definieren Sie mithilfe von entsprechenden Abhängigkeiten und Build-Skripten einen einfachen npm-Build-Prozess, der Folgendes leistet:

- Ein Build-Skript `lint` lintet Ihren JavaScript-Code mithilfe von ESLint. Als ESLint-Konfigurationsdatei können Sie z.B. die der Hausarbeit einsetzen.
- Die Build-Skripte `html` und `css` kopieren die HTML- und CSS-Dateien aus dem Quellordner in einen zu erstellenden Release-Ordner (z.B. "dist").
- Ein Build-Skript `js` konkateniert den JavaScript-Code mithilfe von Browserify und legt die entstehende Datei im Release-Ordner ab.
- Ein Build-Skript `build` erzeugt einen vollständigen Build. Dabei wird zunächst das Projekt gelintet. Sollte dieser Schritt Fehler aufdecken, bricht der Prozess ab.
- Ein Build-Skript `start` erzeugt einen vollständigen Build und startet mithilfe des npm-Moduls `http-server` einen HTTP-Server, der Ihre Anwendung aus dem Release-Ordner zur Verfügung stellt.

Testen Sie sowohl die zusammengesetzten Build-Skripte, als auch jedes einzelne Build-Skript für sich.

Aufgabe 4: Analyse der Ausgabe von Browserify

Schauen Sie sich die von Browserify erzeugte Datei an und versuchen Sie deren Aufbau nachzuvollziehen.