Web-Entwicklung: Übungsblatt 5

Aufgabe 1: Anlegen eines npm-Projekts

Legen Sie mithilfe von `npm init` ein neues Projekt in Form eines eigenen npm-Moduls mit einer Konfigurationsdatei *package.json* an. Belegen Sie die Pflichtfelder in der Konfigurationsdatei mit sinnvollen Werten.

Erzeugen Sie außerdem eine einfache Ordnerstruktur für eine Browser-Anwendung wie in der Vorlesung gezeigt.

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. Fügen Sie das Modul daher Ihrem Projekt als (Laufzeit-)Abhängigkeit hinzu.

Denken Sie daran, dass Ihr Browser keine Unterstützung für npm-Module bietet und auch keine globale Funktion `require()` im Browser existiert. Sie müssen also das npm-Modul `browserify` einsetzen, um Ihre JavaScript-Datei und alle davon direkt oder indirekt referenzierten npm-Module in einer einzigen JavaScript-Datei zu bündeln. Fügen Sie auch dieses Modul Ihrem Projekt als (Entwicklungs-)Abhängigkeit hinzu.

Legen Sie außerdem eine einfache CSS-Datei an, die grundlegende visuelle Eigenschaften der Website festlegt.

Hinweise:

- Um eine Funktion zeitlich gesteuert oder wiederholt aufzurufen, können Sie die globale Funktion `setInterval()` einsetzen.
- Sie werden bald genau lernen, wie man den Browser-Fensterinhalt zur Laufzeit feingranular modifiziert. Für diese Aufgabe ist es ausreichend, den Wert der Eigenschaft `document.body.textContent` zu verändern.

Aufgabe 3: Einrichten eines Build-Prozesses mithilfe von npm

Definieren Sie über weitere Abhängigkeiten und Build-Skripte einen einfachen npm-Build-Prozess, der Folgendes leistet:

- Ein Build-Skript `clean` löscht alle im Zuge des Build-Prozesses heruntergeladenen oder erstellten Dateien.
- Ein Build-Skript `lint` lintet Ihren gesamten JavaScript-Code mithilfe von semistandard.
- Ein Build-Skript `format` wendet die semistandard-Regeln auf Ihren gesamten JavaScript-Code an.
- Die Build-Skripte `html` und `css` kopieren die (statischen) HTML- und CSS-Dateien aus dem Quellordner *src* in einen zu erstellenden Release-Ordner (z.B. *dist*).
- Ein Build-Skript `js` bündelt den JavaScript-Code mithilfe von browserify und legt die entstehende Datei im Release-Ordner ab.
- Ein Build-Skript `debug` erzeugt einen vollständigen Build. Dabei wird zunächst das Projekt gelintet. Sollte dieser Schritt Fehler aufdecken, bricht der Prozess ab.
- Ein Build-Skript `build` erzeugt ebenfalls einen vollständigen Build, minifiziert und obfuskiert aber die von browserify erzeugte Datei mithilfe von terser und minifiziert die CSS-Datei mithilfe von less und less-plugin-clean-css.
- Ein Build-Skript `start` erzeugt einen vollständigen Debug-Build und startet mithilfe des npm-Moduls `http-server` einen HTTP-Server, der Ihre Anwendung aus dem Release-Ordner heraus zur Verfügung stellt.

Testen Sie sowohl die zusammengesetzten Build-Skripte als auch jedes einzelne Build-Skript für sich. Die Ausführung der Anweisung `npm install && npm start` muss dazu führen, dass Ihre Anwendung korrekt startet.