

[Dokumenttitel]

[Untertitel des Dokuments]



Inhalt

[Persönliche Reflexion 2](#_Toc37237527)

[Dokumentation 3](#_Toc37237528)

[Feature 1 – Seiten Ladeanimation 3](#_Toc37237529)

[Feature 2 – Toggle Dark Mode 3](#_Toc37237530)

[Feature 3 – Live-Clock 4](#_Toc37237531)

[Feature 4 – Mail-Alert 5](#_Toc37237532)

[Feature 5 – Back-Up-Prank 5](#_Toc37237533)

# Persönliche Reflexion

Am Ende der Bearbeitung des Assignment angekommen, möchte ich hier nun einen kurzen Rückblick über Gedankengänge zur Aufgabe geben und ggf. einige genannte Aspekte der Dokumentation aufgreifen und beschreiben, bzw. begründen. Ich habe schon viel mit Vanilla javaScript gearbeitet und kenne mich dort relativ gut aus. Ich „schreibe“-Websites in meiner Freizeit oftmals für private Projekte oder helfe Freunden, welche damit nebenbei ihr Geld verdienen. Ich habe mir viele Gedanken gemacht, ob ich also für mich in diesem Assignment ein sehr hohes Niveau erreichen möchte und meine Features extrem ausbauen möchte. Ich bin allerdings ein Fan der Schlichtheit und des „Perfektionierens“ des Codes. Also habe ich mich für Features entschieden, welche einen schönen Effekt oder Animation bieten. Ich habe zwei Features selbst noch nicht gebaut gehabt und also trotzdem etwas Neues gelernt. Diese Features haben mich allerdings gefühlt so viel Zeit gekostet, wie fünf Features zusammen. Zu sagen ist allerdings, dass ich die Funktionslogik schon einmal benutzt habe, jedoch nicht in dem Zusammenhang. D.h. ich habe meinen bereits erstellten Code nicht kopiert, sondern lediglich die erlernte Syntax erneut verwendet!

Die Entscheidung für das Einbauen der Features ist mir mit der Anzahl der Features immer schwerer gefallen. Wenn man die Möglichkeit hat aus so vielen Dingen auszuwählen, finde ich es immer schwieriger mit der Zeit ein Feature zu finden, welches man dann für das Assignment einbauen möchte.

# Dokumentation

## Feature 1 – Seiten Ladeanimation

Ich habe vor ein paar Monaten eine Page Ladeanimation gebaut mit einem GIF in Verbindung von HTML CSS und JavaScript. Da dachte ich, es ist ein „Fancy“ Feature eine Ladeanimation für das Template zu schreiben. Als erstes ging es darum, welches Ladesymbol ich verwende. Ich hatte keine Lust alles selbst per HTML und CSS zu gestalten, zumal dies ja sowieso nicht Bewertungskriterium ist. Deshalb verwendete ich https://loading.io/ und baute mir damit mein Lade-GIF. Eingebunden habe ich das GIF per HTML <img> Tag. Zunächst ging es dann darum das GIF richtig zu positionieren per CSS und eine Hidden-Animation einzufügen. Hier habe ich für das Fade out die Annotation @keyframes verwendet, mit der man einzelne Schritte einer Animation bearbeiten kann. Schließlich habe ich mich an den JavaScript Teil heranbewegt. Ich erstelle ein Event „load“, welches eine Funktion beinhaltet. In dieser Funktion greife ich per DOM mit dem .querySelector auf die CSS-Klasse .loader zu. Jetzt habe ich sie zwar aufgerufen, aber die Animation ist dauerhaft. Dann rufe ich auf der Klasse loader hidden auf und die Animation wird visibility hidden.

Bei diesem Versuch wurde jedoch .hidden als Klasse direkt aufgerufen und ausgeführt. Ich hatte nach langer Fehlersuche keinen blassen Schimmer, wieso die Ladeanimation trotz setTimeoutFunction nicht funktioniert. Ich wendete mich an Josef (den Dozenten der Vorlesung Front-End Entwicklung). Der Code veränderte sich daraufhin nur minimal. Der Unterschied ist einfach, dass die Funktion hideLoader direkt in der Funktion setTiemoutFunction extern aufgerufen und ausgeführt wird. Ebenfalls wird der Klasse .loader im HTML die Klasse .hidden angehängt nach den zwei Sekunden.

Ich habe den Timer nur zu Testzwecken auf 2 Sekunden gesetzt, damit man sieht, dass die Funktion auch wirklich funktioniert. Auf einer Live Seite im Internet würde ich die Funktion umschreiben bezogen auf die Ladezeit der Seite und die FadeOut Animation bezüglich der Länge um die Hälfte kürzen.

## Feature 2 – Toggle Dark Mode

Ich habe mir überlegt, welches Feature heute mehr Aufmerksamkeit denn je erhält. Ganz klar kam ich auf die Thematik Dark Mode. Ich habe mir zuerst überlegt, welche Features ich umfärben möchte und wie ich das Bewerkstelligen möchte. Ich kam auf die klassische Idee per Button. Habe auch überlegt einen Schieberegler zu verwenden, jedoch aufgrund der Benotungskriterien habe ich den Button gewählt. Er ist schneller erstellt und das CSS Styling spielt keine Rolle für die Benotung. Zu färbende Features waren für mich der Hintergrund und die Schrift. Auf mehr habe ich nicht geachtet, da ich für mich erstmal wissen wollte, ob ich das Feature überhaupt selbst programmiert bekomme. Ich kam schnell auf die Idee des Arbeitens und Ersetzten mit CSS-Styling. Ich habe zuerst einen Button mit einem onclick Event in HTML erstellt und die Styling-Klasse der Buttons übernommen. Ich habe nun auf der CSS-Seite die Dark Mode-Farben plus zzgl. Styling ersetzt aus den Originalklassen oder IDs, die das CSS-Styling definieren. Dann habe ich mich an den zu bewertenden Teil gemacht. Ich habe eine Funktion toggleDarkMode definiert, welche vom Button per Klick aufgerufen wird. In der Funktion wird die Variable darkMode definiert, mit der ich auf die Klasse dark-mode-button zugreife und die Schriftfarbe ändere. Ebenfalls greife ich auf die Klasse .wrapper zu und ersetze diese durch meinen veränderten Style für den Dark Mode.

Jetzt stoße ich aber auf ein Problem! Bei erneutem betätigen des Knopfes wird die Hintergrundfarbe nicht geändert, sondern nur die Schrift. Ich habe versucht die Funktion umzuschreiben und per jQuery per .one die Betätigung nur einmal ausführen zu können. Ich kam nur zu weiteren Fehlern also suchte ich nach einer anderen Lösung. Es muss ein zweiter Button her für die Aktivierung des Light Mode. Damit löse ich zwar das Problem des Bugs nicht komplett, allerdings biete ich eine Möglichkeit wieder den Light Mode zu verwenden ohne die Seite neu zu laden. Man betrachte also die Verwendung des Dark Mode als vorrübergehender Betatest.

Für die Aktivierung des Light Modes wird dann wiederrum das CSS-Styling per JavaScript angesprochen per onclick. Jetzt kann man also den Light Mode abrufen, ohne die Seite neu laden zu müssen.

## Feature 3 – Live-Clock

Als nächstes Feature hatte eine Live-Uhrzeit im Kopf. Warum ich das im Kopf hatte ist, da ich sehr viel in meiner Freizeit streame auf Twitch. Dort habe ich immer eine Uhr mitlaufen, welche mir die aktuelle Streaming-Zeit anzeigt. Ich hatte nun die Überlegung zwischen einem Session-Tracker der Nutzer um zu erfahren wie lange sich der Nutzer auf der Seite befindet oder einer Live-Uhrzeit. Auch aufgrund des Aufwandes in anderen Kursen habe ich mich für die spartanische Variante mit der Live-Uhrzeit entschieden.

Hier was der HTML Aufwand einfach ein <p>-Tag, welches ich per CSS gestylt habe. Der JavaScript Teil ist der aufwändigste bis jetzt geschriebenen Teil für Features des Assignments.

Für die aktualisierende Zeit muss ich eine Funktion schreiben, welche zuerst Variablen für das Datum, sowie die einzelnen Elemente der Zeit erstellt. Bspw. Hier Stunden, Minuten, Sekunden und der Live-Zeit selbst als „Date“! Die Zeitangaben habe ich dann mit den Methoden .getHours, etc. abgerufen um immer die genaue Live-Zeit zu erhalten. Dann habe ich auf mein <p>-Tag zugegriffen und die Zeit per .innerHTML mit den Variablen befüllt.

Aber wie aktualisiert sich jetzt die Zeit? Habe ich mich gefragt.

Dann kam mir die Idee wieder setTimeout zu verwenden wie beim Pre-Loader um die Zeit der Aktualisierung zu bestimmen. Die Funktion startTime() habe ich also dort aufgerufen und somit die Zeit der „Intervalle“ bestimmt.

Jetzt hatte ich allerdings noch ein Problem. Wenn eine Zahl nur einstellig war wurde keine null davor platziert und somit sah die Uhrzeit sehr komisch und ungewohnt aus. Schnell wurde mir klar, dass ich prüfen musste, ob die Zahlen jeweils einstellig waren um davor mit einer if-Bedingung zusätzlich eine „0“ zu platzieren.

Ich konstruierte als die Funktion liveTimeCheckUnder10(). Hier prüfte ich also stumpf ob die Zahl < 10 ist. Wenn das der Fall ist, wird eine null angehängt. Wenn sie größer ist, springt der Code nicht in die Bedingung, da sie nicht erfüllt ist.

Der Code funktionierte allerdings so nicht. Ich habe also den Debugger in Chrome verwendet um zu prüfen, ob die neue zahl überhaupt übergeben wird. Es war leider nicht der Fall. Dann ist mir aufgefallen ich habe den Wert nicht returned. Zeile Code hinzugefügt, nochmal geladen und dann gemerkt, dass es immer noch nicht funktioniert. Ich habe nach Überlegezeit angefangen im Internet per YouTube nachzuforschen. Ich kam auf ein Ergebnis. Die Person hat noch die neue Zeit (Minuten, Sekunden) in der alten Variablen aufgerufen per Methodenaufruf und hier auf die Variable Minuten und Sekunden übergeben. Abgetippt vom Video und neu geladen und schon hat das Ganze funktioniert.

## Feature 4 – Mail-Alert

Da mich das Feature 3 sehr viel Zeit gekostet hat, wollte ich nun noch zwei kleinere Features einbauen. Ich entschied mich in einem Feature für einen Alert. Ich habe mir überlegt, an welcher Stelle ich den Alert einbringen möchte. Ich habe mich für das Editieren der E-Mail entschieden. Ich habe das HTML umgebaut. Habe die Mail als <button> gekennzeichnet und mit einem onclick-Event versehen. Dann habe ich das CSS-Styling, damit man direkt sieht es passt nicht ins Layout orange gefärbt. Dann habe ich die Funktion mailAlert() in JavaScript geschrieben. Im Alert wird beschrieben, dass die Mail Adresse nicht funktioniert und man doch bitte die Besitzer der Seite telefonisch erreichen soll.

Warum habe ich mich aber für die Mail entschieden? Ich habe mich für die Mail entschieden, da bei technischen Problemen dieser Code einfach einkommentiert werden kann und man mit dem Alert die Aufmerksamkeit der Nutzer auf den Alert mit der Nachricht lenkt. So wissen die Nutzer, dass sie zwar nicht per Mail die Besitzer erreichen können, jedoch immer noch die Chance erhalten die Besitzer zu erreichen.

Das CSS-Styling würde ich für eine live-Seite jedoch ändern. Ich würde die Farbe des Buttons unsichtbar machen und den Text unterstrichen, sodass der <button> nicht auffällt.

## Feature 5 – Back-Up-Prank

Für das letzte Feature habe ich mir etwas Besonderes einfallen lassen. Ich habe mich gefragt, welche Feature denn unfassbar nützlich ist, ich jedoch seiner eigentlichen Funktion enteignen möchte. Ich habe mir Videos mit Usability freundlichen Features angesehen auf YouTube und dabei bin ich auf den Back-up Button gekommen. Zu sagen ist, dass dieser Button das Leben des Benutzers enorm vereinfacht. Da ich jedoch die Idee eines Pranks im Kopf hatte, dachte ich mir, ich werde diesen Button auf einer festen Position unterhalb des Footers anbringen. Bei der Überlegung wie ich am Besten die Funktion des Buttons gestalten möchte ist mir die Überlegung mit einer Art Text-Animation gekommen. Ich hatte zuerst die Idee einfach auf eine troll-Meme Seite weiterzuleiten. Dann dachte ich es muss eine On-Page Animation sein. Also kam ich auf die Idee einen Textinhalt zu ändern. Ich erstellte also ein <h3>-Tag und gab diesem die ID „pranked“. Ich habe in der Zwischenzeit das Styling des Buttons verändert. Dann habe ich im JavaScript Teil in einer Funktion auf die ID des <h3>-Tag zugegriffen und den Wert des Tags mit .innerHTML geändert mit der Aufschrift, dass der Nutzer geprankt wurde und doch bitte selber nach oben scrollen soll.

Da ich bei JavaScript schon seit längerem immer den Fehler mache, onclick direkt auf den Button anzuwenden, habe ich diesen Fehler wieder gemacht und mich gewundert, warum wiedermal das Feature nicht funktioniert. Ich ging systematisch vor und fing bei meinen bekannten Fehlerquellen an. Ich habe dann schnell meinen Fehler gefunden und entsprechend das den Button mit dem onlick versehen. Per Live-Server die Seite aktualisiert und der Prank-Button erfüllt jetzt seine Funktion.