Assignment 1

DHBW Mosbach, Frontend-Entwicklung, Herr Slezak, HSS 2020, Alischa Thomas

Vorkenntnisse und Vorbereitung

Meine Vorkenntnisse im Themenbereich Programmieren mit JavaScript (JS) belaufen sich auf ein theoretisches Grundverständnis von Programmieren im Allgemeinen und praktischen Grundlagenkenntnissen mit HTML und CSS. Deshalb arbeitete ich vor der Bearbeitung dieses Assignments nochmal den Foliensatz zu Vorlesung 1 durch. Des Weiteren beschäftigte ich mich nochmal mit den Übungen aus dem ersten Foliensatz. Darüber hinaus arbeitete ich den interaktiven Kurs *JavaScript Fundamentals for ES6* auf der Plattform *Pluralsight* durch. Das Ziel dabei war ein praktisches Verständnis der Syntax und Logik von JavaScript zu erarbeiten. Vor dem Einstieg ging ich nochmal systematisch die Anforderungen an das Assignment in Moodle durch. Daraus identifizierte ich für mich nochmals klar das Ziel des Assignments für mich und den Dozierenden. Ich überlegte mir was ich im Rahmen dieses Assignments erlernen möchte und welches Vorgehen am Besten geeignet ist um diese Ziele zu erreichen. Aus Erfahrungen mit der Konfrontation neuer Programmieraufgaben weiß ich, dass ich mich in der Vergangenheit schnell erschlagen fühlte und daher auch vorzeitig an meine Grenzen stieß. Deshalb schrieb ich die Anforderungen und meinen Plan erstmal schriftlich nieder um somit den Fokus während des Arbeitsprozesses wahren zu können.

- Zielvorgabe: Manipulation eines Templates mit JS: im Vordergrund an UI Elementen, im Hintergrund, DOM-Manipulation (5 Interaktionselemente), Dokumentation inkl. Persönliche Reflexion, Abgabe versioniertes Git Repository
- Meine Ziele: Ajax, Animation, Canvas, Buttoninteraktion
- Bedingungen: ES6 Standard verwenden, kein jQuery
- Bewertung: JS-Logik

Um nicht den Überblick zu verlieren wurde das Assignment Schritt für Schritt aufgebaut. Dies diente dazu sich intensiv mit den einzelnen Bestandteilen auseinandersetzen zu können und sich nicht gleich überfordert zu fühlen. Dabei wurde das Ziel gesetzt sich mit jedem Schritt in der Komplexität zu steigern. Das Git-Repository wurde mit GitHub umgesetzt. Auf das Repository kann unter folgendem Link zugriffen werden: *https://github.com/althomi/assignment1* der Code in der Umgebung von Webstorm (IntelliJ) geschrieben.

1 Button Manipulation

Idee

Das Template weist bereits 4 Buttons mit Stylingattributen auf. Das Ziel der Buttonmanipulation ist es das Erscheinungsbild im User Interface beim Abfeuern eines Events zu verändern.

Theoretischer Hintergrund

Die Änderung des User Interfaces dient in der Praxis z.B. einem entscheidungsfreundlichen Design oder der besseren Orientierung für NutzerInnen. Eine Entscheidung kann zum Beispiel unterstütz werden, wenn der Button die Farbe bei einem Mouseover ändert. Dadurch wird die Klickbarkeit des Buttons hervorgehoben. Bei der Orientierung kann es helfen, wenn bereits geklickte Buttons farblich gekennzeichnet werden wie zum Beispiel auf der Suchergebnisliste von *Google*.

Umgesetzt wird diese Manipulation mit den beiden Methoden *getElementbyID()* (*HTML DOM getElementById() Method*, n.d.) und *addEventListener()*. Die erste Methode greift auf das HTML-Element zu. Die zweite Methode weist ihm ein Event zu, auf das reagiert wird. Dabei gibt es unterschiedlichste Events aus den Bereichen User Interface Events, Focus and Blur Events, Mouse Events, Keyboard Events, Form Events, Mutation Events and Observers, HTML5 Events, CSS Events (dfteam7, 2019). Bei dieser Button Manipulation werden Mouse Events umgesetzt. Spezifisch für Klickevents gibt es alternativ noch die *Onklick-Methode*. Beide Methoden wären bei dieser Manipulation anwendbar (*javascript—AddEventListener vs onclick*, n.d.). Da die *addEventListener-Methode* allerdings flexibler und breiter aufgestellt ist, wird diese im Folgenden angewendet.

Praktische Umsetzung Programmieren

Bevor mit dem Schreiben von JavaScript begonnen werden kann, müssen die HTML-Elemente der Buttons mit einer ID versehen werden. Über diese ID erlangt man Zugriff auf das Element in dem JS-Dokument mit der Methode *getElementByID()*. Die zu manipulierenden Buttons wurden über diese Methode einer Variablen zugewiesen. Im nächsten Schritt wird der Variable über die Methode addEventListener() das Event zugewiesen, auf das reagiert werden soll. In diesem Zuge wird außerdem die Funktion definiert, die beim Abfeuern des Events ausgeführt wird. Diese Manipulation bezieht sich hauptsächlich auf die Veränderung der Füllfarbe (*AddEventListener and changing background color*, 2018). Dazu ruft man an der zuvor erstellten Buttonvariable die Stylemethoden auf. Die Farbe wurde passend zur Website mit dem *Adobe Color Tool* ausgewählt (*Farbpalette, das Farbschema für Künstler | Adobe Color*, n.d.).

Um den Style der Buttons manipulieren zu können wurden zuerst die Standardstylings von

Buttons in CSS recherchiert (*CSS Buttons*, n.d.). Wir das Standardstyling nicht beachtet kann es zu Fehlern in der Manipulation kommen. Möchte man dem Rahmen beim Abfeuern eines Events eine Farbe geben wie bei Button4, muss dem Button zusätzlich zu der Farbe ein Borderstyle im CSS-Dokument zugewiesen werden, da dieser im Standardstyling auf *none* steht. Andernfalls würde man keine Veränderung durch das Event sehen können.
In dieser Manipulation werden verschiedene Funktionen ausgetestet um die Arbeit mit Buttons zu üben:

- Button1: Eventtyp „click“, aufgerufene Methode: style.backgroundcolor
    - Ergebnis: Bei einem Klick verändert sich die Füllfarbe des Buttons
- Button2: Eventtyp “mouseover”, aufgerufene Methode: style.backgroundcolor
    - Ergebnis: Bei Hovern über den Button verändert sich seine Füllfarbe permanent
- Button3: Eventtyp “mouseover”, und “mouseout”, aufgerufene Methode: style.backgroundcolor
    - Ergebnis: Bei Hovern über Button verändert sich die Füllfarbe des Buttons temporär. Sie wird auf den Originalzustand zurückgesetzt sobald der Cursor die Buttonfläche wieder verlässt.
- Button4: Eventtyp “mouseover” und “mouseout”, aufgerufene Methode: style.borderstyle, style.bordercolor, style.color
    - Ergebnis: Der Rahmen, die Füllfarbe und Schriftfarbe ändern sich beim Hovern über den Button. Sie werden auf den Originalzustand zurückgesetzt sobald der Cursor die Buttonfläche wieder verlässt.

Persönliche Reflexion

Probleme und Hindernisse

Bei dieser ersten Manipulation bezogen sich die größten Probleme auf den Einstieg. Es fiel mir schwer anzufangen weil ich nicht genau wusste wo und wie. Beim Arbeiten mit HTML wird einem zum Beispiel bereits eine grobe Struktur vorgegeben, das ist hier nicht der Fall. Was mir hier weitergeholfen hat war die Sichtung des Codes anderer auf *Stackoverflow*. Dadurch fing ich dann einfach an zu programmieren und hatte schon schnell meine ersten Code-Snippets erstellt.
Zu Beginn bereitete mir die Arbeit mit dem integrierten Version Control Tool von Webstorm starke Probleme und kostete mich viel Zeit. Durch intensive Recherche und mehrere erstellte und doch wieder gelöschte Repositories, sowie einem Austausch mit meinem Dozierenden

entschied ich mich dazu Github über Terminal und die zugehörige Desktop App zu verwenden. Dadurch habe ich gelernt den für mich am besten geeigneten Weg zu wählen. Des Weiteren wurde mir bewusst, dass eine Alles in einem Lösung nicht immer die Optimalste ist, da diese manchmal weniger Kontrolle bedeutet.

Zwischenfazit

Die Manipulation eines Buttons auf der Oberfläche, dem User Interface, empfand ich als sehr gut geeigneten Einstieg in dieses Assignment. Der JavaScript Code anderer Developer ist an diesem Komplexitätspunkt gut nachzuvollziehen wodurch ein Einstieg erleichtert wurde. Ich habe viele verschiedene Arten von Buttonmanipulationen ausgetestet und konnte für mich die Vielfalt auf dieser niedrigen Komplexitätsebene entdecken. Dieses erste Interaktionselement dient im Verlauf des Assignments weiteren Manipulationen (Manipulation 3 Ajax Abruf und 4 Ajax Präsentation auf UI).

2 Animation mit Canvas

Idee

Die Bühne des Templates bietet viel Platz. Hier soll ein Satz einige Sekunden nach Öffnen der Website erscheinen. Die einzelnen Worte werden dabei zeitverzögert ausgesteuert.

Theoretischer Hintergrund

Mit JavaScript können animierte Inhalte in Kopplung an das HTML-Canvas Tag ausgesteuert werden. Das Canvas-Tag stellt einen leeren Container auf der Website dar, in den mit Hilfe von JS Inhalte „gezeichnet“ werden können. Dies betrifft Wörter, Formen, aber auch ganze Bilder (*HTML Canvas*, n.d.). Das Event, das hierbei i.d.R. die Animation auslöst, ist das Laden der Seite. Das Canvas kann in verschiedenen Kontexten verwendet werden. Deshalb wird dieser nach dem Zugriff auf das HTML-Element mit *getElementByID()* mit der *getConntext()-Methode* festgelegt. Danach stehen alle Methoden dieses Kontextes zur Verfügung. Die meist genutzten Basisfunktionen sind dabei *moveTo(), lineTo, stroke(), beginPath(), arc(), font(), fillText(), strokeText(), drawImage()* (*HTML Canvas*, n.d.). Diese ermöglichen die Basisfunktionen im Canvas.

Neben dem Canvas wird in dieser Manipulation außerdem die *setTimeout()-Methode* verwendet. Diese dient dazu Inhalte erst nach einer bestimmten Zeit auf der Website anzeigen zu lassen (*Window setTimeout() Method*, n.d.). In diese Methode kann eine Funktion eingebettet werden, die eine bestimmte Aktion nach der definierten Zeit ausführt (*javascript—Delay script*

loading, n.d.). Diese Methode kann für unterschiedlichste Elemente angezeigt werden. Ursprünglich ist sie dafür gedacht Alertboxen anzuzeigen. Sie kann aber auch für andere Elemente wie das Canvas angewandt werden (*javascript—SetTimeout with canvas*, n.d.).

Praktische Umsetzung Programmieren

Für die Umsetzung wurde dem HTML-Dokument ein Canvas Tag inklusive Größenangaben definiert. Im JS-Dokument werden drei Animationen auf dem gleichen Canvas durchgeführt. Diese sind über drei Funktionen definiert. In jeder Animationsfunktion wird zuerst die Methode setTimeout eingenestet. Innerhalb dieser wird das Canvaselement definiert. Dazu wird über die Methode *getElementByID()* auf das Canvas-Element zugegriffen und über *getContext()* der 2d-Kontext festgelegt. Anschließend werden die Methoden font(), fillStyle, filltext() an der zuvor initialisierten Canvas-Variablen aufgerufen. Diese definieren Schriftgröße und -art, Füllfarbe, Textinhalt und Position im Canvas. Anschließend wird die Funktion aufgerufen. Zum Ende wird die Zeitverzögerung in Millisekunden angegeben. Diese ist aufsteigend gestaffelt damit die Textinhalte nacheinander auf der Website animiert werden. Da die drei Funktion einige gleiche Zeilen an Code enthalten wird eine allgemeine Funktion geschrieben, an die dann für die jeweilige Animation spezifische Parameter übergeben werden. Dies betrifft den Textinhalt und dessen Position. Der *Timeout* wird innerhalb der einzelnen Funktionen definiert. Somit wird nun der Text I NEED YOU in großen roten Lettern auf der Bühne des Templates mit einer Zeitverzögerung von einer Sekunde für I, zwei Sekunden für NEED und drei Sekunden für YOU in der Schriftart *Changa One* ausgesteuert.

Persönliche Reflexion

Probleme und Hindernisse

Die Erstellung des Canvas an sich stellte keine Probleme dar. Allerdings bedurfte es einiger Recherche bezüglich der Positionierung. Das Canvas bietet zwei Arten der Position an: die des Canvas an sich und die der Inhalte. Erstere wird über CSS-Eigenschaften, zweitere über JS gesteuert (*css—How to position canvas using relative/absolute positioning*, n.d.). Ein weiteres Problem stellte das Styling des Textes dar. Nach einer kurzen Recherche stellte ich fest, dass es verschieden Methoden für die Outlines (*strokeText())* und die Füllung des Textes (*fillText())* gibt (Chinnathambi, n.d.) (*HTML Canvas Text*, n.d.).

Nachdem das Canvas umgesetzt wurde fiel beim Neuladen des Inhaltes in einem neuen Tab auf, dass der Loop nicht die eigebettete Schrift von Anfang an abgriff und die Animation des ersten Buchstabens *I* zweimal aussteuerte. Somit erschien einmal ein *I* in der Fallbackschrift und eins in der neu definierten. Nach einiger Eigenrecherche bedurfte eines eines Tech Reviews mit Kollegen aus meinem Betrieb. Wir fanden gemeinsam heraus, dass die Einbindung

der neuen Schrift in das HTML-Dokument dem Canvas-Element nicht ausreicht. Die Schrift muss vorher schon einmal aufgerufen worden sein (*html—Drawing text to <canvas> with @font-face does not work at the first time*, n.d.) . Deshalb wurde zusätzlich das HTML-Element *h2* mit der Schrift *Changa One* im zugehörigen CSS-Styling versehen. Somit wird diese beim Laden der Seite vor der Erstellung des Canvas aufgerufen und kann von diesem dann mit eingebunden werden.

Zwischenfazit

Die zweite Manipulatoin des Assignments stellte mich vor komplexere Herausforderungen. Letztens Endes konnte ich diese durch Codebeispiele von *Stackoverflow* und mit gemeinsamer Hilfe meiner Kollegen lösen. Die Größte dabei war es die Methode *setTimeout()*zu integrieren und eine verschachtelte Funktion zu erstellen. Nach dem Bewältigen der Aufgabe habe ich nun nicht nur gelernt wie ich Animationen mit Hilfe des Canvas-Tag einfügen kann, sondern auch wie ich selbst Funktionen schreiben und miteinander verbinden kann. Darüber hinaus konnte ich den Stil meines Codes für die Zukunft verbessern indem ich gelernt habe wie ich Funktionen so schreibe, dass ich sie an anderer Stelle wiederverwenden kann. Das im vorigen Abschnitt beschriebene Problem beim Laden der Schrift hat mich gelehrt, dass auch erfahrene Programmierer nicht alles wissen und ich somit auch nicht alles wissen muss. Es ist in Ordnung andere Personen aus dem Feld um Hilfe zu bitten, da es sehr spezifische Hürden geben kann.

3 Ajax Abruf

Idee

Mit Hilfe von Ajax sollen Daten von einem anderen Server abgerufen werden. Dies geschieht im Hintergrund ohne Kenntnisnahme von NutzerInnen.

Theoretischer Hintergrund

Ajax (Asynchronous JavaScript and XML) ermöglicht ein asynchrones Senden und Empfangen von Daten ohne einen Page Reload auszuführen (*AJAX Send an XMLHttpRequest To a Server*, n.d.). Ein Beispiel dafür ist eine Suchleiste auf einer Website. Mit Ajax kann, während NutzerInnen ihren Suchbegriff eingeben, eine Datenbank auf einem Server durchsuchen und den Suchbegriff entsprechend der gefundenen Begriffe vorausfüllen. Somit wird asynchron eine Datenabfrage an den Server geschickt und passende Daten empfangen und ausgesteuert. Die Kommunikation mit dem Server läuft dabei folgendermaßen ab: Ein Event wird im Browser abgefeuert z.B. Klick. Dieses erstellt ein XMLHTTPRequestObject, welches einen

HTTPRequest an den Server sendet. Dieser wird vom Server geprüft und eine Antwort wird an den Browser gesendet. Dieser verarbeitet die Antwort mit Hilfe von JavaScript und updated entsprechend den Seiteninhalt. Wichtig zu beachten ist, dass die Dateien, auf die man zugreifen möchte und die Websitedokumente auf dem gleichen Server liegen. Denn Ajax erlaubt keinen cross-domain Zugriff. Es kommt hierbei zu Kollisionen mit CORS (Cross Origin Ressource Sharing). Dieses kontrolliert den Zugriff auf den Server von anderen Zugriffsquellen aus und verbietet diese in der Regel.

Praktische Umsetzung Programmieren

Zuerst wird, wie in der Theorie beschrieben, ein neues XMLHTTPRequestObject erstellt. Anschließend wird die Methode *open* an diesem Objekt aufgerufen (*AJAX Send an XMLHttpRequest To a Server*, n.d.). Diese spezifiziert den HTTPRequest. Innerhalb dieser wird die Methode *GET* aufgerufen, sowie die zugehörige *URL*. Damit wird die Art des HTTPRequests festgelegt. Es wird etwas abgerufen von der angegebenen URL *https://learn-webcode.github.io/json-example/animals-1.json*. Im nächsten Schritt wird die Methode *onload* an dem XMLHTTPRequestObject aufgerufen. In dieser ist eine Callback-Methode definiert, die ausgeführt wird, wenn der HTTPRequest erfolgreich war (*XMLHttpRequestEventTarget.onload*, n.d.). Mit Hilfe des Aufrufes von *responseText* innerhalb der Callback-Funktion wird der Text ausgegeben, der auf dem Server abgerufen wird. Dieser werden anschließend in der Konsole ausgegeben. Somit wird der Ajax Request folgendermaßend definiert:

- es wird etwas abgerufen von einer anderen Domain
- dieses Abrufen passiert, wenn die HTML-Seite lädt und gibt die Daten, die von der anderen Domain abgefragt werden in der Konsole aus

Abschließend wird durch das Aufrufen der Methode *send* dieser HTTPRequest abgesendet. Durch das Öffnen der Konsole im Developer Tool kann durch die Callback-Funktion überprüft werden ob der Request erfolgreich war.
(Als Basis für die Umsetzung dieser Manipulation diente folgendes Tutorial: (*JSON and AJAX Tutorial: With Real Examples—YouTube*, n.d.)

Persönliche Reflexion

Probleme und Hindernisse

Diese Manipulation stellte für mich bei der Bearbeitung einen deutlich höheren Arbeitsaufwand dar. Neben der intensiven Auseinandersetzung mit dem Thema stand ich außerdem vor dem in der Theorie beschriebenen Zugriffproblem mit CORS. Ich hatte also keinen Zugriff auf andere Server und war ratlos wie ich diesen bekommen würde.

Nach einiger Recherche entdeckte ich ein hilfreiches Tutorial zum Umgang mit Ajax (*JSON and AJAX Tutorial: With Real Examples—YouTube*, n.d.)). Allerdings schien auch die dort verlinkte URL, die einen Zugriff versprach, nicht zu funktionieren. Somit war ich mit meinem Latein am Ende und nahm das Thema als zweites Problem mit in den Tech Review mit meinen Kollegen. Wir fanden heraus, dass ich das Problem richtig erkannt hatte, fragten uns aber weshalb es in dem Tutorial funktionierte und nicht bei mir. Nach genauem Betrachten entdeckten wir, dass die URL in dem Video eine andere war als die verlinkte. Die in dem Video war frei zugänglich für mich, wodurch mein Ajax Request dann doch noch erfolgreich wurde.

Zwischenfazit

Bevor ich die Manipulation anging sah ich mir umfangreiches Material zum Thema *Ajax* online an. Dabei nutze ich für die Theorie *MDM WebDocs* und für die Praxis sah ich mir einige *YouTube*-Tutorials zu dem Thema an. Im Gegensatz zu den ersten beiden Manipulationen fiel es mir schwerer die praktische Umsetzung zu verstehen und benötigte deutlich mehr Recherche. Für mich persönlich waren die Vorgänge im Hintergrund zu Beginn schwer zu greifen und sehr abstrakt. Dies stellte aber auch den Hauptgrund das Thema in Angriff zu nehmen. In der Vorlesung hatte ich bereits den gleichen Eindruck und wollte diese Unsicherheit durch dieses Assignment lösen. Nach dieser Manipulation mit Ajax habe ich nun den Eindruck grundlegende Funktionsweise und Methoden verstanden zu haben und nun erfolgreich HTTPRequest senden zu können. Allerdings müsste ich mich für eine künftige Verwendung noch einmal intensiv mit dem Thema CORS auseinandersetzen, da mir dieses auch wesentliche Probleme bereitet hat.

4 Ajax Präsentation auf UI

Idee

In dieser Manipulation wird an Manipulation1: Button Manipulation und Manipulation 3: Ajax Abruf angeknüpft. Die Daten, die beim Ajax Abruf angefordert werden, sollen nach einem an einem Button ausgeführten Klick Event auf der Benutzeroberfläche angezeigt werden. Somit können NutzerInnen Inhalte auf der Website angezeigt werden sobald sie eine Aktion ausführen wie z.B. einen Button zu klicken.

Theoretischer Hintergrund

Der theoretische Hintergrund zu Ajax wurde bereits bei Manipulation 3 beschrieben. In dieser Manipulation werden zusätzliche Funktionen, Methoden und Manipulationen am HTML-Dokument vorgenommen, die im folgenden praktischen Teil nachzuvollziehen sind.

Praktische Umsetzung Programmieren

Im ersten Schritt werden Button3, der Ajax Button, und ein im HTML-Dokument neu angelegtes Div-Element, das sich direkt unter diesem Button befindet mit ihrer ID über die Methode getElementByID() in dem JS-Dokument aufgerufen. Der Button wird benötigt, da an diesem das Klick-Event stattfinden wird, also den HTTPRequest aussteuert. Die abgerufenen Daten über diesen Request werden dann in dem neuen Div-Element angezeigt. Im nächsten Schritt wird der HTTPRequest in die Funktion des EventListeners, die an dem Button aufgerufen wird, eingebettet. Die addEventListener-Funktion ist dabei für das Klickevent definiert. Anschließend wird die Methode preventDefault() an dem Event aufgerufen um zu vermeiden, dass die bisher definierte Standardaktion ausgeführt wird (*Event.preventDefault()*, n.d.). Anschließend wird der HTTPRequest mit dem Typ *open* und der Methode *GET* und der in Manipulation 3 bereits verwendeten URL definiert.
Anschließend wird, wie in Manipulation 3, mit der Methode *onload* festgelegt, dass bei Aussteuerung des Requests Daten abgerufen werden. Im Gegensatz zu Manipulation 3 sollen diese aber nun NutzerInnen präsentiert werden. Dazu wird eine weitere Methode *JSON.parse* angewendet (*JSON.parse()*, n.d.) . Diese liest die abgerufene JSON-Datei aus wodurch der Inhalt zu normal leserlichem Inhalt entpackt wird. Diese ausgelesenen Daten werden in einer Variablen gespeichert, die mit der selbst definierten Funktion *createHTML* als HTML-Element gespeichert werden.
Die Funktion *createHTML* kreiert ein Div-Element, das für jedes Objekt in der JSON-Datei ein Paragraph-Element erstellt und es mit dem entpackten JSON-Objekt füllt. Anschließend wird die Positionierung dieses neu erstellen HTML-Element mit der Methode *insertAdjacentHTML()* festgelegt, indem diese an dem vorher unter dem Ajax Button definierten HTML-Element aufgerufen wird. Mit dem Attribut *beforeend* wird das neu erstellte JSON-HTML unter dem vorher festgelegten Ajax-Div auf der Benutzeroberfläche angezeigt. Somit ist der HTTPRequest mit folgenden Merkmalen definiert:

- reagiert auf das Klickevent an Button3
- vermeidet die Standardaktion des Buttons
- greift auf die definierte URL zu und zieht davon JSON-Daten
- liest JSON-Datei aus und kreiert ein HTML-Element
- zeigt JSON-Inhalte auf der Benutzeroberfläche an

Nun wird der HTTPRequest wie in Manipulation 3 mit der Methode *send* abgeschickt.

Persönliche Reflexion

Probleme und Hindernisse

Als Grundlage diente auch hier das in Manipulation 3 angegebene *YouTube*-Tutorial. Allerdings waren einige Veränderungen an der *createHTML*-Funktion notwendig, damit diese funktionierte. Hierbei unterstützen mich ebenfalls meine Kollegen mit hilfreichen Tipps und Verbesserungsvorschlägen für meinen Code, besonders bei dem *for-loop* innerhalb der Funktion. Dieser war ständig fehlerhaft, allerdings konnte ich den Fehler nicht selbständig zurückverfolgen. Durch ein kleines praktisches Tutorial meiner Kollegen im Thema *Debuggin mit dem Google Chrome Dev Tool* konnte ich den Fehler dann schnell beheben. Es lag daran, dass ich normale Kommata statt Semikolon innerhalb der *For-Schleife* verwendete.

Zwischenfazit

Diese Manipulation hat mir besonders gut gefallen, da ich meine Schritte zuvor mit einbinden konnte. Das gab mir das Gefühl etwas programmiert zu haben, das komplexer ist und wirklich in der Realität angewandt werden kann, sowie einen deutlichen Mehrwert bringt. Dies ist die komplexeste Manipulation dieses Assignments und hat mich lange beschäftigt. Der Code besteht aus deutlich mehr Bestandteilen, die in komplexerer Art und Weise zusammenarbeiten. Doch durch die tiefe Arbeit an den vorherigen Manipulationen konnte ich bereits viel Wissen auf diese anwenden. Ohne die Vorarbeit an den ersten dreien wäre ich sicher erschlagen gewesen von der vierten. Das zeigt mir, dass man auch komplexere Probleme und Aufgaben mit JS bewältigen kann indem man sie in seine einzelnen Teile destrukturiert und angeht.

5 Klick Zähler

Idee

Beim zweiten wiederholten Klicken auf Button1 soll dieser die Füllfarbe wechseln.

Theoretischer Hintergrund

Ein solcher Klick-Counter wird eingesetzt um z.B. auf der SERP von *Google* bereits besuchte Links lilafarben anzeigen zu lassen. Dies erleichert NutzterInnen die Orientierung und vermeidet eine hohe Bounce-Rate, die durch versehentliches Wiederbesuchen und schnelles Verlassen von Websites entstehen würde. Dazu wird eine *If-Bedingung* an das Klickevent angehangen, die die Änderung der Farbe bei Übersteigen eines mitlaufenden Counters ausführt.

Praktische Umsetzung

Im ersten Schritt wird eine Variable erstellt, die auf einen bestimmten Anfangswert gesetzt wird; in diesem Fall 1. Anschließend wird die Methode *addEventListener()* an Button1 aufgerufen. Diese Methode addiert die zuvor initialisierte Variable bei jedem Aufruf um 1. Außerdem wird in einer *If-Bedingung* festgelegt, dass der Button seine Füllfarbe ändert sobald er das zweite Mal angeklickt, die Funktion durch das Event also das zweite Mal ausgesteuert wurde. Die Füllfarbe wird wie bei Manipulation 1 durch die Methode style. backgroundcolor geändert.

Persönliche Reflexion

Probleme und Hindernisse

Bei dieser Manipulation gab es keine Probleme und Hindernisse. Alle Probleme, die hier hätten auftauchen können, begegneten mir bereits zuvor und ich wusste nun damit umzugehen bzw. konnte sie von vorneherein vermeiden.

Zwischenfazit

Die Idee zu dieser Manipulation lieferte mir die Aufgabenbeschreibung zu Assignment1 auf *Moodle*. Sie schlug als eine Option einen *page counter* vor. Beim Ansehen des *YouTube*-Videos von Manipulation 3 wurde ebenfalls auf einen *click counter* angesprochen. Noch bevor in dem Tutorial gezeigt werden konnte wie es umgesetzt wurde, stoppte ich das Tutorial und wollte es auf eigene Faust versuchen. Sofort hatte ich eine genaue Vorstellung davon wie der Code funktionieren musste und konnte ihn selbständig erfolgreich umsetzen. Ohne Tutorial und Onlinerecherche konnte ich selbst meinen eigenen Klick-Counter auf einen Button anwenden. Das gab mir das Gefühl mich mittlerweile gut in die Denkweise von JS eingelebt zu haben und die Basislogik so verinnerlicht zu haben, dass ich jetzt leichter Probleme angehen und einfache Inhalte selbst programmieren kann.

Gesamtfazit

In diesem Assignment habe ich mich selbst vor Herausforderungen wie z.B. dem Umgang mit Ajax gestellt, wollte aber auch strukturiert Basiskenntnisse erlernen wie z.B. das Verschachteln von Funktionen, For-Schleifen und If-Bedingungen. Durch die Aufbereitung der Dokumentation habe ich außerdem eine gute Zusammenfassung meines erlangten Wissens und kann im weiteren Verlauf meines Studiums darauf zurückgreifen.
Auch, wenn es zeitlich sehr intensiv war, hat es sich doch gelohnt Inhalte so tief und detailliert aufzuarbeiten.

Quellen

AddEventListener and changing background color. (2018, May 25). The FreeCodeCamp Forum. https://www.freecodecamp.org/forum/t/addeventlistener-and-changing-background-color/193023

AJAX Send an XMLHttpRequest To a Server. (n.d.). Retrieved April 13, 2020, from https://www.w3schools.com/xml/ajax_xmlhttprequest_send.asp

Chinnathambi, K. (n.d.). *Drawing Text on the Canvas*. Kirupa.Com. Retrieved April 5, 2020, from https://www.kirupa.com/canvas/drawing_text.htm

CSS Buttons. (n.d.). Retrieved April 5, 2020, from https://www.w3schools.com/css/css3_buttons.asp

css—How to position canvas using relative/absolute positioning. (n.d.). Stack Overflow. Retrieved April 5, 2020, from https://stackoverflow.com/questions/17265803/how-to-position-canvas-using-relative-absolute-positioning

dfteam7. (2019, August 1). *JavaScript Event Types—8 Essential Types to shape your JS Concepts!* DataFlair. https://data-flair.training/blogs/javascript-event-types/

Event.preventDefault(). (n.d.). MDN Web Docs. Retrieved April 13, 2020, from https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault

Farbpalette, das Farbschema für Künstler | Adobe Color. (n.d.). Retrieved April 5, 2020, from https://color.adobe.com/de/explore

HTML Canvas. (n.d.). Retrieved April 5, 2020, from https://www.w3schools.com/html/html5_canvas.asp

HTML Canvas Text. (n.d.). Retrieved April 5, 2020, from https://www.w3schools.com/graphics/canvas_text.asp

HTML DOM getElementById() Method. (n.d.). Retrieved April 5, 2020, from https://www.w3schools.com/jsref/met_document_getelementbyid.asp

html—Drawing text to <canvas> with @font-face does not work at the first time. (n.d.). Stack Overflow. Retrieved April 13, 2020, from https://stackoverflow.com/questions/2756575/drawing-text-to-canvas-with-font-face-does-not-work-at-the-first-time

javascript—AddEventListener vs onclick. (n.d.). Stack Overflow. Retrieved April 5, 2020, from https://stackoverflow.com/questions/6348494/addeventlistener-vs-onclick

javascript—Delay script loading. (n.d.). Stack Overflow. Retrieved April 5, 2020, from https://stackoverflow.com/questions/9611714/delay-script-loading

javascript—SetTimeout with canvas. (n.d.). Stack Overflow. Retrieved April 5, 2020, from https://stackoverflow.com/questions/12087874/settimeout-with-canvas

JSON and AJAX Tutorial: With Real Examples—YouTube. (n.d.). Retrieved April 13, 2020, from https://www.youtube.com/watch?v=rJesac0_Ftw

JSON.parse(). (n.d.). MDN Web Docs. Retrieved April 13, 2020, from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

Window setTimeout() Method. (n.d.). Retrieved April 5, 2020, from https://www.w3schools.com/jsref/met_win_settimeout.asp

XMLHttpRequestEventTarget.onload. (n.d.). MDN Web Docs. Retrieved April 13, 2020, from https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequestEventTarget/onload