Dokumentation Assignment 3

Lifestyleblog Webcomponents mit stencil.js

09.05.2020

Autorinnen

Alischa Thomas	Katharina Barth	Nina Eberle	Larissa Eirich
Matr.-Nr: 3019616	Matr.-Nr: 1334995	Matr.-Nr:7134284	Matr.-Nr: 6916710

DHBW Mosbach

Studiengang Onlinemedien

LV Frontend-Entwicklung

Modul T2 in ON19-2

HSS 2020

Inhaltsverzeichnis

Vorbereitung: alle Autorinnen

Auswahl Thema

Zu Beginn der Bearbeitung war relativ schnell klar, dass das Thema für unser gemeinsames drittes Assignment eine Website sein sollte, da wir alle schon Erfahrung mit dem Gestalten einer Website hatten. Anfangs standen für das Design-System unserer Website eine Onlinezeitung oder einen Blog zur Auswahl. Es wurde sich für einen Lifestyleblog entschieden, da dieser viele verschiedene Möglichkeiten bietet Komponenten zu erstellen und somit jedes Teammitglied zu sich passende Aufgaben innerhalb des Assignments finden kann.
Wir entschieden uns für einen Lifestyleblog. Um Inspiration hinsichtlich der Aufteilung und des Designs zu erlangen, haben wir folgende Beispielseiten angeschaut: *Sallys Blog* und die Webseite der *NewYork Times* (siehe Anhang). Wir entschieden uns dafür eine Art Fusion aus den beiden als Vorlage für unsere Webseite zu verwenden. Der grundlegende Rahmen ist an einen Blog angelehnt und die einzelnen Artikel, wie jene auf der News-Seite einer Zeitung, dargestellt.

Festlegung Komponenten

Im zweiten Schritt entschieden wir durch ein Brainstorming darüber, welche Komponenten essentiell für unser Design-System sind. Hierfür sammelte zunächst jedes Teammitglied, auf unterschiedlichsten Webseiten Inspiration und legte dann die für ihn wichtigsten Komponenten in einem gemeinsamen Dokument (siehe Anhang) ab. Als nächster Schritt folgte die Festlegung der Komponenten. Das gesamte Team traf eine finale Auswahl von acht Komponenten, welche wir in *Stencil.js* für unser Design-System erstellt haben.
Als feststand welche Komponenten umgesetzt werden sollten, gab jedes Teammitglied seine zwei präferierten Komponenten an und die Aufgabenverteilung wurde in einer Tabelle namentlich festgehalten. So wusste jedes Teammitglied genau, welche Komponente von welcher Person erstellt wird. Durch die Screenshots der Inspiration-Seiten, auf denen die einzelnen Komponenten abgebildet sind, existierte eine grobe Richtung des Designs. Jedoch hat zusätzlich jedes Gruppenmitglied einen Wireframe artigen Entwurf (siehe Anhang) seiner Komponente erstellt. Des Weiteren hat Katharina (siehe Anhang) ein grobes Layout der gesamten Website erstellt, sodass die gesamte Gruppe in der Lage war sich einen vorläufigen Eindruck vom Gesamtbild zu verschaffen. Ziel der Website ist es ein gebrauchsfertiges Mockup aus Komponenten für einen Lifestyle-Blog zu erstellen.

Technische Vorbereitung

Um mit der Erstellung beginnen zu können wurde ein Repository auf *gitHub* erstellt. Danach wurde das Stencil-Projekt aufgesetzt und alle Gruppenmitglieder haben sich für ihre Komponenten einen eigenen Branch erstellt. Alle Komponenten wurden mit einem Shadow-DOM erstellt, damit die Manipulation durch Dritten ausgeschlossen ist. Im Folgenden werden alle Komponenten von der jeweiligen Entwicklerin genau erläutert. Die Erläuterung ist untergliedert in die Themen Idee, Funktionsweise und persönliche Reflexion der Erstellung.

Komponenten

Button Login und Registrierung: Alischa Thomas

Idee

Zur Vorbereitung auf die danach folgende Komponente *Login-Formular* werden zwei Buttons erstellt. Dabei öffnet der Login-Button den Loginbereich der Website. Der Registrierungs-Button führt auf eine hypothetische Seite, auf der sich LeserInnen für den Blog registrieren können.

Praktische Umsetzung

Der Login-Button und Registrierungs-Button werden als zwei separate Komponenten erstellt. Dazu werden beide mit Stylings im jeweils zugehörigen CSS-Dokument ausgestattet. Das TSX-Dokument bildet einen Shadow-tree inklusive Funktionalität für die Komponenten. Über diese beiden Bestandteile werden beide Buttons als *customized Tag* in das Dokument eingebunden (shadow-root) (*Styling Components - Stencil*, n.d.).

Um die Komponenten der Buttons zu erstellen werden die Decorators *Component, Event, h, EventEmitter* (*Stencil | Props, State & Tons of Decorators*, n.d.) verwendet. Diese werden vom *@stencil/core* Paket importiert (*Decorators - Stencil*, n.d.). In *Component* wird der Tagname für den Aufruf im HTML-Dokument festgelegt, das Dokument als Styledokument zugewiesen und das TSX-Dokument als Shadow-DOM festgelegt (*Styling Components - Stencil*, n.d.). Anschließend wird die Klasse für den Export vorbereitet. Ihr werden die benötigten Decorators und Funktionen zugewiesen, sowie der Shadow-tree aufgebaut. Dieser beinhaltet alles, was in das daraus resultierende customized Tag übertragen werden soll. Der verwendete Decorator *@Event()* sorgt dafür, dass ein *EventEmitter* aufgrund eines Klicks an dem Element (Button) ausgeführt wird (*Events - Stencil*, n.d.). Diese Kombination hört auf das Event des Klicks und führt dann die dafür definierten Funktionen aus. Dazu sind zweierlei Dinge notwendig: die Funktion wird definiert und im Shadow-tree einem HTML-Tag zugewiesen, hier dem <button>-Element. Mit der Funktion *handleClick()* wird in der Konsole ein String ausgegeben. Damit kann überprüft werden, ob der *Eventlistener* erfolgreich war. Jetzt liegt ein Button vor, der durch einen Klick einen definierten Text in der Konsole ausgibt.

Die exakt gleiche Vorgehensweise trifft auf den Button für die Registrierung zu. Dieser wird als separater Shadow-root in das HTML-Dokument durch den dafür in der Komponente definierten customized Tag eingebaut. Der Inhalt des TSX-Dokumentes ist von der Logik her gleich. Der einzige Unterschied liegt im Inhalt des eingegebenen Textes, da sich dieser nun auf den Registrierungs-Button bezieht. Die Betitelung des Buttons unterscheidet sich damit seine Funktion für NutzerInnen eindeutig wird.
Somit können diese Komponenten als klickbare Login-/ und Registrierungs-Buttons überall auf der Seite und auf anderen Seiten eingebaut werden. Sie steuern beide beim Klicken des Buttons einen passenden Text in der Konsole aus.

Persönliche Reflexion

Trotz der simplen Button-Komponenten kostete der Einstieg in die Thematik von *stencil.js* einen extrem hohen Rechercheaufwand, da die Tutorials aus der Vorlesung zu dem Zeitpunkt der Bearbeitung der Aufgabe noch nicht zur Verfügung standen. Darüber hinaus ist die Onlinedokumentation von *stencil.js* selbst zwar in den Grundzügen vorhanden, weist aber nur den theoretischen Einsatz von grundlegenden Code Snippets auf. Die Dokumentation zeigt leider auch nicht den Zusammenhang der verschiedenen zu verwendenden Dokumente auf. Die Onlinerecherche ergab außerdem extrem wenige Resultate, die aufgrund des jungen Entwicklungszeitpunkt von *stencil.js* ein hohes Komplexitätsniveau aufweisen. Somit gestaltete sich das Einstiegsniveau auch für einfache Buttons sehr hoch.
Unter Betracht dieses komplexen Niveaus erwiesen sich diese als Einstieg in *stencil.js* jedoch als einfachste Alternative. Dadurch konnte ich den Grundaufbau von TSX-Dokumenten kennenlernen und den Zusammenhang verschiedener Dokumente wie .CSS .HTML und .TSX. Außerdem konnte ich zwei Standard-Decorators kennenlernen und die Logik hinter Ihnen an einfachen Funktion wie *console.log()* austesten. Gesamt forderte diese Aufgabe bereits von Beginn an viel Zeit und Konzentration, lernte mich aber auch eine gute und umfangreiche Onlinedokumentation wertzuschätzen und vertiefte mein Verständnis für die Logik von Komponenten in *stencil.js* in ihren Grundzügen.

Login-Formular: Alischa Thomas

Idee

Das Login-Formular selbst stellt eine neue, selbständige Komponente dar, in die die zuvor erstellten Button-Komponenten eingebaut werden. In diesem Formular können sich Leserinnen des Blogs mit ihrem Namen und Mailadresse anmelden, ein vergessenes Passwort neu anfordern oder zur Registrierung wechseln. Die tatsächlichen Funktionen des Anmeldens, Anfordern und Registrierens wurden hierbei nicht umgesetzt, lediglich die Darstellung und Interaktion auf der Oberfläche.

Praktische Umsetzung

Wie bei den eben erstellten Button-Komponenten wird ein im TSX-Dokument definiertes customized Tag im HTML-Dokument eingebunden. Dieses wird im TSX-Dokument mit Funktionen und Inhalt gefüllt (shadow-tree). Die Komponente wird außerdem mit einem CSS-Dokument verbunden und als Shadow-tree deklariert. Auch für diese Komponente werden die Decorators *@Prop()* und *@Event()* verwendet. Diese geben der Login-Box eine Überschrift als Datentyp *String*, welcher aus dem customized Tag im HTML-Dokument übergeben wird und sorgen unter anderem dafür, dass sich die Login-Box durch Auslösen eines Klickevents öffnet.

Dieses Klickevent wird durch Funktionen definiert, die an der eingebetteten Loginbutton-Komponente im Shadow-tree aufgerufen werden. Die beiden dafür benötigten Funktionen arbeiten miteinander, sowie mit dem in dieser Komponente zusätzlich verwendeten Decorator *@State()*. Dieser hat einen bestimmten festgelegten Status (click=false, open=false). Wird dieser Status geändert z.B. durch einen Klick auf ein Element, wird der Render-Block im Shadow-tree wiederholt ausgeführt (*State Decorator-Stencil*, n.d.). Durch den Default-Status ist die Login-Box geschlossen, nur der Button wird angezeigt. Bei einem Klickevent auf den Button ändert sich der Status des *click* auf *true* wodurch sich auch der Status von *open* auf *true* ändert und die Loginbox geöffnet wird. Der Prozess läuft zusammengefasst folgendermaßen ab: Es kommt zu einem Klick auf den Button, der Status des in den Funktionen hinterlegten Decorators ändert sich und der Render-Block wird ausgeführt. Somit öffnet und vice versa schließt sich die Login-box. Das Schließen läuft dabei über den gleichen Prozess ab, in dem allerdings der Status wieder auf den Ausgangswert *click=false*, *open=false* zurückgesetzt wird. Somit ist der Loginbutton erneut bereit ein Klickevent zu empfangen, das die öffnende Funktion auslöst.

Da die Decorator und Funktionen diesen Prozess anstoßen, aber nicht definieren, muss hinterlegt werden was sich inhaltlich verändert. Dies passiert im Render-Block. Innerhalb dieses werden verschiedene Styles an die Elemente des Shadow-trees angehängt. Es gibt ein hinterlegtes Styling im CSS-Dokument, das ausgeführt wird, wenn die Box geöffnet ist und eins, wenn die Box geschlossen ist. Der geöffnete Status fragt das default-Styling ab, der geschlossene ein spezifisch definiertes, das die Login-Box verschwinden lässt.
Der vollständige Shadow-tree im Render-Block enthält somit eine Loginbox mit einer Überschrift, zwei Formularen zur NutzerInneneingabe der Mailadresse und des Passwortes, einen Login-button um den Login-Prozess anzustoßen, einen Link um das Passwort neu anzufordern und einen Registrierungs-Button, der Nutzerinnen auf eine externe Seite zur Neuregistrierung führen soll. Diese sind über HTML-Tags bzw. customized tags eingebunden. Das erste Element ist der eventempfangende Login-Button selbst. Die Buttons werden innerhalb des Shadow-trees der Login-Komponente mit Hilfe ihrer customized Tags *genestet* und somit zu ihren Kindelementen gemacht (*Javascript - How Nested Components Works in*

Stencil?, n.d.). Damit sich diese innerhalb der Elternkomponente an der richtigen Stelle befinden, werden ihnen zusätzliche CSS-Styles in dem CSS-Dokument der Elternkomponente gegeben. Der Login-Button, der Passwort-Link und der Registrierungs-Button sind mit einer Funktion ausgestattet, die einen Text in der Konsole darüber ausgibt, ob die Ausführung des Klickevents erfolgreich war. Dies dient der Fehlervorbeugung und simuliert die Weiterleitung auf eine in der Theorie folgende Seite.

Anhand der Tags im Shadow-tre werden die eben erwähnten CSS-Stylings bei jeder Ausführung des Render-Blocks durch Auslösen des Klickevents ausgeführt. Folglich wird bei jeder Ausführung des Blocks zwischen den beiden zugewiesenen Styles anhand jedes Tags die Erscheinung geändert, also der Status gewechselt (*Getting Started With Stencil - A Web Component Generator*, n.d.).

Nun besitzt die Website zwei Login-Buttons und einen Registrierungs-Button, von denen einer für das Öffnen einer Loginbox verantwortlich ist, in der nochmals verschiedene HTML-Elemente eingebettet sind, sowie eine Login-Komponenten und eine Registrierungskomponente. Die beiden überflüssigen Buttons für Login und Registrierung können durch das zusätzliche CSS-Styling *hidden*, aufgerufen am customized Tag im HTML-Dokument, versteckt werden. Theoretisch gesehen könnten die Tags vollständig aus dem HTML-Dokument entfernt werden, da sie nicht mehr benötigt werden. Aus Vollständigkeitsgründen für die Erfüllung des Assignments werden diese im Dokument beibehalten und versteckt. Somit liegt nun ein Login-Button vor, der aufgrund eines Klicks eine Loginbox mit Formularen und zugehörigen Buttons (ebenfalls Komponenten) anzeigt und durch einen erneuten Klick wieder schließt.

Persönliche Reflexion

Der Einstieg in den Einbau der Login-Komponente fiel durch die Vorarbeit an den Button-Komponenten leicht. Das anfängliche Bedenken über das *Nesten* der bereits erstellten Komponenten löste sich ebenfalls schnell in Luft auf, da die Logik dahinter durch meine Vorkenntnisse im Programmieren leicht nachzuvollziehen war.

Der Teil der Komponente, der mir Schwierigkeiten bereitete war das Öffnen und Schließen der Login-Box. Die Logik hinter der Zusammenarbeit von verschiedenen Funktionen, Decorators und CSS-Styles ist komplex und aufgrund der nur wenig vorhandenen Onlinedokumentation schwierig nachzuvollziehen. Deshalb den Code meiner Gruppenmitglieder, ,die bereits Ähnliches eingebettet hatten als Orientierungshilfe und konnte mit wenigen zusätzlichen Tutorials innerhalb der Gruppe die Funktion des Öffnens und Schließens auch in meiner Komponente einbetten. Dadurch habe ich gelernt, dass es beim Programmieren manchmal hilfreich sein kann sich länger mit einem kleinen Codeteil zu beschäftigen und diesen tiefgehend zu verstehen anstatt viele Codebeispiele z.B. bei der Onlinerecherche nur oberflächlich zu begutachten.

Nach dieser intensiven Auseinandersetzung mit dem Thema habe ich sehr gut verstanden wie die einzelne Codebereiche innerhalb des TSX-Dokumentes miteinander agieren und Daten übergeben können. Somit konnte ich aus der Erstellung der Button-Komponenten herausziehen wie die Dokumente generell miteinander in Verbindung stehen und wie der generelle Aufbau einer Komponente aussieht. Zusätzlich konnte ich anhand der Erstellung der Login-Komponente sehen wie Komponenten an anderer Stelle wiederverwendet werden, sowie erkennen wie die Inhalte der TSX-Datei auf tieferer Mikroebene miteinander verwoben sind und sich gegenseitig bedingen.

Header: Katharina Barth

Idee

Da die Grundidee unserer Webseite ein Blog ist, ist ein Header unabdingbar. Diesen sieht man in der Regel sowohl auf der Haupt- als auch auf jeder Unterseite, weswegen er sich sehr gut als Komponente eignet. Da wir uns bei der Searchbar und dem Login-Button für eigenständige Komponenten entschieden haben, beinhaltet der Header lediglich das Logo und fünf Links, die man individuell benennen und verlinken kann.

Praktische Umsetzung

Zu Beginn dieser Komponente wurde das Basis-Setup in einer separaten Datei aufgesetzt. Diese fängt also mit einem Import des Component-Decorators aus dem *@stencil/core* Paket (*Decorators - Stencil*, n.d.) an und erstellt eine *Componen*t, in der der Tag-Name, das passende Stylesheet und das Assets-Verzeichnis initialisiert werden. Daraufhin wird eine Klasse exportiert, in diesem Fall eine namens HeaderKatharina, die zunächst eine leere render-Funktion enthält. In ihrem return-Wert wurde daraufhin ein klassischer Header mit Hilfe einer *unordered list* und *anchors* aufgebaut, und mit dem dazugehörigen Stylesheet bearbeitet. Um die return-Werte wiedergeben zu können, wurde in diesem Zusammenhang das *hyperscrpt* (h) mit importiert. Da man die Benennung der Reiter im Header und auch ihre Verlinkungen individuell einstellen können sollte, war es nötig *Props* einzubinden und damit Variablen zu deklarieren, die an den entsprechenden Stellen im Header aufgerufen werden. Da im Header auch ein Logo auftauchen sollte, wurde auch dafür eine Variable angelegt, die mit Hilfe von *getAssetPath*, was ebenfalls in den Import eingebaut wurde, das Logo an der passenden Stelle aufruft. Dieses kann ebenfalls von Außen der Header-Komponente übergeben werden.

Da der Header am Anfang noch responsiv gestaltet werden sollte, wurde eine State Property hinzugefügt, die den Boolean-Wert einer Variable bei Veränderungen prüfen soll, sodass die Komponente neu gerendert werden kann. Geplant war hier ab einer Breite von 750px ein Burgermenü. Die zugehörigen Funktionalitäten finden sich auskommentiert noch im Code.

Persönliche Reflexion

Auch ich habe mit der Einarbeitung in Stencil vor dem Upload der Tutorials begonnen, weswegen ich sehr viel Zeit zum Einlesen aufwenden musste, um überhaupt mit einer Komponente starten zu können. Der Header hat sich aber als gute Einstiegs-Komponente erwiesen, da er sehr wenig Logik hinter sich stehen hat, sondern hauptsächlich aus Style- bzw. HTML-Elementen besteht. Das heißt, ich konnte mich hier auf nur wenige Decorator konzentrieren und diese so besser kennenlernen. Auch war es schwer mich von der mir bisher bekannten JavaScript-Syntax zu trennen, da vieles so mit Stencil nicht funktionierte, bzw. nicht angenommen wurde. So war es für mich am Anfang sehr unklar, warum manche Sachen nicht funktionieren und musste zu Beginn bei jeder Zeile Code alles nochmal umdenken und die richtige Syntax für mein Vorhaben recherchieren (oder auch bei beinen Gruppenmitgliedern "spicken"). Ich denke, dass dies vor allem an der für Anfänger ungeeigneten Dokumentation von Stencil lag. Eine weitere Schwierigkeit, die ich zuvor beim Programmieren nicht in diesem Ausmaß kannte, war das Stylen meiner Komponente (auch später beim Slider). Ich hatte das Gefühl, dass das CSS, das ich kenne in Stencil andere Auswirkungen auf die verschiedenen Elemente hatte. So musste ich auch hier viel Zeit investieren, um die Komponente so zu gestalten, wie ich sie gerne hätte, obwohl mir so etwas normalerweise recht leicht fällt.
Des Weiteren war ich es aus JavaScript gewohnt, dass Probleme die ich beim Programmieren hatte, schon einmal bei einer anderen Person vorkamen, weswegen es auch viele Lösungshilfen in diversen Communities gibt. Bei Stencil musste man oft aus verschiedensten Quellen Informationen herausfiltern um auf eine Lösung zu kommen, was wegen meiner geringen Vorkenntnisse oft "Trial and Error" war. Deswegen musste ich mich zwar stark mit Stencil auseinandersetzen, habe aber dadurch meine Programmierkenntnisse deutlich verbessert. Außerdem konnte ich dadurch eine gute Basis für meine zweite Komponente schaffen.

Slider: Katharina Barth

Idee

Für einen kleinen Hingucker auf der Webseite haben wir uns für eine Slider entschieden. Dieser geht über die volle Breite (bzw. 75%) und beinhaltet drei Bilder der Größe (960 x 360)px, die man sowohl weiter, als auch zurück klicken kann.

Praktische Umsetzung

Das Starter-Setup wurder bei der Slider-Komponente von der Header-Komponente übernommen. Die benötigten Decorator dafür waren: *Component, h, Prop*, und *State* (*Stencil | Props, State & Tons of Decorators*, n.d.). Die benötigten Variablen, die mit Hilfe von *@Prop* erstellt wurden, erlauben es später in der index.html die gewünschten Bilder in den Slider einzubauen. Diese werden zunächst in einen Array gepusht, durch den mit if-Bedingungen die verschiedenen Bilder in einer bestimmten Reihenfolge abgerufen werden. Dazu wurden zwei

Buttons in Form von Pfeilen erstellt, die jeweils ein *onclick*-Event zugewiesen bekommen haben. Hier kann man noch erwähnen, dass der Button über die ganze Bildhöhe geht, damit man (vor Allem auf einem Mobile Device) nicht exakt den Pfeil treffen muss, sondern einfach auf das rechte bzw. linke Ende des Bildes tippen kann. Da auch hier die Komponente immer wieder neu gerendert werden muss, wurde die *State Property* gebraucht, mit der man eine Variable deklariert, die den Wert *true* oder *false* haben kann. Der Wert wird durch das Klicken der erstellten Buttons verändert. Durch die Abfrage der Variable beim Rendern, wird ein anderer Index des zuvor erstellten Arrays angezeigt, wodurch man durch die Bilder klicken kann. Am Ende der Abfrage, wird die Variable wieder auf *false* gesetzt. So wird bei einem Buttonklick das Rendern der Komponente immer wieder neu aktiviert. Mit dem Gedanken, dass man nicht nur drei, sondern später eventuell mehr Bilder im Slider haben will, wurde eine weitere Variable erstellt, die die Länge des Arrays speichert. So ist die letzte Indexnummer dynamisch und der Übergang vom letzten zum ersten Bild (der Slider sollte wie ein Karussell wirken) muss nicht manuell angepasst werden.

Persönliche Reflexion

Nach dem Header ging der Basis-Aufbau des Sliders sehr schnell. Hier war es die Logik, die mir mit Stencil Schwierigkeiten bereitet hat. Auch hier war dementsprechend Vorarbeit nötig, um mit den Decorator Event, EventEmitter und Method überhaupt arbeiten zu können. Hierbei waren vor allem YouTube Tutorials hilfreich. Im Verlauf des Aufbaus des Sliders, konnte ich diese aber sogar ganz weglassen. Ein weiteres Problem, das beim Slider aufgekommen ist, war die Benennung der Parameter für die Übergabe der Bilddateinamen.
Die Variablen *image1* bis *image5,* die ich zu Beginn so genannt hatte, erlaubten mir nicht die Parameter in der HTML-Datei an meine Komponente zu übergeben. Dass das Problem tatsächlich an der Benennung lag, wurde mir erst sehr spät klar, auch wenn ich bis jetzt nicht genau weiß warum. So habe ich hier sehr viel Zeit damit verbracht, die Fehler in der Logik meines Codes zu suchen, obwohl diese von Anfang an stimmte.
Meine größten Hindernisse (bei beiden Komponenten) waren also letztendlich die für mich nicht so intuitive Syntax und das Stylen der erstellten Komponenten. Ich konnte aber auf jeden Fall viel aus dieser Aufgabe lernen und mein Können verbessern.

Footer: Larissa Eirich

Idee

Der abschließende Teil jeder Webseite ist wohl der Footer, so auch in unserem Blog. Aus unseren vielfältigen Inspirationsquellen ist ein minimalistischer Footer entstanden. Er wird durch eine schmale horizontale Linie vom Rest den Inhaltes optisch getrennt. Darunter befinden sich die Elemente des Footers, auf der linken Seite die Social-Media-Icons und auf der Rechten die Links zu relevanten Unterseiten unseres Blogs. Die eingebundenen

Funktionen führen jedoch zu einer Art Platzhalter Links, da sie lediglich der Veranschaulichung dienen.

Praktische Umsetzung

Begonnen wurde die praktische Umsetzung bei meiner Person durch die Aufsetzung des *Stencil.js* Projektes. In das bereits bestehende Repository auf *gitHub* wurde die Struktur von *Stencil.js* für Komponenten initialisiert (*Styling Components - Stencil*, n.d.). Um die Grundstruktur einer Stencil-Komponenten zu verwenden wurde die Testkomponente kopiert und an den entsprechenden Stellen umbenannt. In der ersten Zeile der TSX-Datei werden die Schlüsselwörter *Component*, um einen eigenen Component zu erstellen, Prop, um Attribute aus dem HTML-Dokument zu importieren und h, für die Verwendung von regulärer HTML-Syntax innerhalb des eigenen Komponenten importiert. Die Klasse FooterLarissa ist für die Funktionalität des Footers zuständig. Die sechs Props am Anfang der Funktion sorgen dafür, dass der Text und die URL der jeweiligen Links im HTML-Dokument als Attribut des Komponenten-Tags übergeben werden können. Dies hat zur Folge, dass man die Footer-Komponenten auf verschiedensten Unterseiten verwenden kann und die Links individuell anpassbar sind. Die jeweils ersten Props sind mit einem Ausrufungszeichen gekennzeichnet, sodass der Verwender gezwungen wird mindestens einen Link und dessen URL bei Einbindung der Komponente mitzugeben. Darauf folgt im Dokument eine Funktion, in der die Social-Media-Icons in einem zweidimensionalen Array abgelegt werden (*Getting Started With Stencil - A Web Component Generator*, n.d.). Auf der ersten Ebene werden drei Slots für die jeweiligen Social-Media-Foren erstellt und auf der zweiten Ebene dann die entsprechenden Information wie Name, URL und Pfad zum Bild/Icon abgelegt. Durch die Render-Methode wird die semantische Struktur des Komponenten zurück gegeben. Beginnend mit einer äußeren Wrapper-Klasse worauf eine Klasse folgt, welche für die Rastereinteilung zuständig ist. Danach folgt ein Div-Element welches die Trennlinie darstellt. Im linken Teil des Rasters werden in einem weiteren Div-Element die Icons durch eine Map-Methode nacheinander aus dem obigen Array in ein Link-Element gerendert. Schließlich werden auf der rechten Seite die Links aus den obigen Props in eine Liste importiert. Somit sind alle benötigten Bestandteile.

Persönliche Reflexion

Wie bereits zu Beginn erwähnt habe ich die Aufgabe übernommen das Projekt aufzusetzen und habe ich mich zuerst mit der Materie auseinander gesetzt, im nachhinein wäre es vielleicht einfacher gewesen zur selben Zeit daran zu arbeiten um sich besser austauschen zu können. Die *Stencil.js*-Dokumentation war mir anfangs noch zu abstrakt, als dass sie mir mit meinem Verständnis weiter geholfen hätte. An sich ist die Logik des Footers nicht sonderlich kompliziert. Allerdings gibt es wenig Referenzprojekte zu *Stencil.js* im Allgemeinen, sodass ich mich erstmal mit der allgemeinen Struktur von *Stencil.js* auseinander gesetzt habe um das Prinzip zu verstehen. Ein weiteres Problem, das dadurch auftauchte, waren die verschiedenen

Versionen von *Stencil.js*. Meistens hatte ich die passenden Codebeispiele als Tutorial gefunden jedoch kam ich durch diese nicht weiter, weil manche Syntaxen von älteren Versionen nicht mehr unterstützt wurden. Auch die Tatsache, dass wir als Gruppe lediglich mit einzelnen Webkomponenten gearbeitet haben und nicht mit der Struktur einer Stencil-App ist anfangs sehr erschwerend gewesen. Oft war man noch nicht in der Lage abzuschätzen welche Code-Bestandteile sich auch außerhalb einer App mit kleinen Anpassungen wieder verwenden lassen. Nachdem wir das erste Assignment in JavaScript bereits geschrieben hatten, war mir oft klar welche HTML-Elemente ich verwenden möchte und wie ich das Ergebnis erreichen kann, das ich haben möchte. Allerdings habe ich oft Stunden damit verbracht die Syntax die ich im Kopf hatte so zu schreiben, dass ich auch mit *Stencil.js* erreichen kann was ich mir vorgestellt hatte. Nachdem ich meine anfänglichen Schwierigkeiten überwunden hatte und die Links im Footer nicht nur vorhanden sondern auch klickbar waren, ergab sich die nächste Hürde. Das Design bzw. die dahinterliegende CSS-Struktur. Denn dazu gibt es keinerlei Dokumentation. Man verbringt sehr viel Zeit damit Dinge an die dafür vorhergesehene Stelle zu bringen. Durch Ausprobieren kann man sich irgendwann erschließen welches CSS dem anderen übergeordnet ist. Das Rahmenwerk von StencilJs ist mir aber bis zum heutigen Zeitpunkt unerschlossen geblieben, an manchen Stellen verhält es sich gegensätzlich zu jeder Logik und an gewissen Punkten kann man dem leider nicht vollständig entgegenwirken, weil man die Punkte an denen man die Anpassungen vornehmen sollte, nicht lokalisieren kann. Trotz allem war der Footer ein Erfolgserlebnis nach seiner Fertigstellung.

Searchmask: Larissa Eirich

Idee

Die Idee kam auch hier bei der Inspirationssuche zustande. Eine Suchmaske ist bei einem Lifestyle-Blog, der eine Vielzahl an verschiedenen Beiträgen zu einer großen Bandbreite an Thematiken bieten soll, ebenso essentiell wie eine Navigationsleiste um sich auf der Website zurecht zu finden. Die Suchmaske ist im ungenutzten Zustand relativ unscheinbar durch ein Lupensymbol dargestellt. Erst durch den Klick auf dieses, um eine Suchanfrage auszuführen, fährt sich ein voll funktionsfähiges Suchfeld aus in das man den Suchbegriff eingeben kann. Dies hat den Vorteil, dass man mit der Einbindung dieser Komponente wertvollen Platz sparen kann.

Praktische Umsetzung

Auch hier ist die Grundlage des eigen erstellten Komponenten eine TSX-Datei, importiert werden zusätzlich noch der *State*, *Event* und *EventEmitter-Operator*, da es nötig ist den Zustand der einzelnen Objekte zu überwachen und darauf hin eine Funktion aufzurufen. Dafür werden zwei boolesche Variablen mit dem Operator State versehen. Die erste wird als true

definiert damit die Lupe standardmäßig angezeigt wird. Der zweite Boolean ist dafür zuständig das Suchfeld vorerst nicht anzuzeigen. Danach werden zwei Eventemitter mit dem Operator Event deklariert, die dafür sorgen, dass die entsprechende Funktion ausgeführt wird, wenn das entsprechende Event angesprochen wurde.
In dieser Konstruktion ist es sinnvoller zuerst die HTML-Struktur zu erläutern. Diese besteht aus einem äußeren Wrapper der je nach Zustand durch eine andere CSS-Klasse angesprochen wird da die Lupe eine deutlich geringere Breite als das Suchfeld hat. Darunter folgt ein Button in dem die Lupe enthalten ist, wird dieser angeklickt ruft er die Methode toggleComponent auf. Diese Methode ändert den ersten Boolean auf false, dies hat zur Folge, dass durch den Eventemitter eventToggle die Lupe ausgeblendet wird in dem die CSS-Klasse geändert wird (StencilJS fundamentals. n.d.). Außerdem wird innerhalb der oben genannten Funktion, eine weitere aufgerufen die dafür zuständig ist das Suchfeld und den dazugehörigen Schließen-Button anzuzeigen, auch dies geschieht innerhalb der zweiten Funktion durch CSS-Klassen. Zuletzt wird auf dem Schließen-Button wieder die erste Funktion aufgerufen, welche dann wieder dafür sorgt, dass bei einem Klick auf den Button das Suchfeld und der Schließen-Button ausgeblendet werden und die Suchlupe wieder angezeigt wird. Die booleschen Werte sind also wieder auf die Ausgangsposition zurückgesetzt. Somit ist die Komponente nun voll funktionsfähig und einsatzbereit.

Persönliche Reflexion

Die Searchmask war meine zweite Komponente und die Erstellung dementsprechend einfacher. Auch hier gab es die Probleme, welche ich Mittlerweile als *Stencil.js* typisch bezeichnen würde. Ich wusste von Beginn an wie ich meine Funktion umsetzen will, allerdings gingen auch hier wieder einige Stunden drauf bis ich herausgefunden hatte wie ich in *Stencil.js* das erreiche, was ich haben möchte. Als ich in der geringfügig vorhandenen Dokumentation die passende Syntax gefunden habe, war es innerhalb von wenigen Minuten erledigt. Das CSS war anfangs auch noch relativ einfach zu gestalten, abgesehen von den Hierarchien die meist nicht logisch nachzuvollziehen sind, kann man mit viel investierter Zeit und Ehrgeiz ein ansprechendes Ergebnis erreichen. Allerdings befanden sich meine Komponenten zu diesem Zeitpunkt noch alleine auf der Website, dementsprechend gab es kein CSS mit dem sie kollidieren konnten.
Umso größer sind am Ende die Erfolgserlebnisse die man hat wenn das Endprodukt endlich Form annimmt. Dieses Projekt hat mich sowohl in technischer Hinsicht als auch auf Recherche-Ebene beziehungsweise die Art wie man aus wenigen Informationen viel lernen und produzieren kann weiter gebracht. Abschließend möchte ich noch zur Geltung bringen das *Stencil.js* mit seinem Konzept eine echte Erleichterung sein kann wenn man eine vollständige Webpräsenz aufbauen möchte.

Artikel-Teaser: Nina Eberle

Idee

Das Teaser-Element hat die Funktion, durch die Einbindung eines Links, auf einen Artikel zu gelangen. Bestandteile dieses Elements sind das Teaser Image, die Headline, das Vorschaubild der jeweiligen Webseite, das Thema des Artikels, der Teaser-Text, das Datum und ein *"read-more"* Link, der durch einen Pfeil, auf der rechten Seite des Links, ergänzt wird. Verlinkt ist allerdings der gesamte Artikel Teaser, da es auf diese Weise einfacher ist zu dem Artikel zu gelangen und man nicht lange nach der verlinkten Headline oder nach dem *"more"*-Link suchen muss. Im Frontend werden mehrere Artikel-Teaser nebeneinander dargestellt, da auf diese Weise ein größere Auswahl zur Verfügung gestellt wird.

Praktische Umsetzung

Im HTML wird das Host-Element mit *article-nina* angegeben. Innerhalb dieses Elements wird die Headline, das Thema, der Text, das Datum und der *"read-more"*-Text zusätzlich gerendert. Die einzelnen Artikel-Teaser werden mehrmals in der index.html untereinander aufgelistet und im CSS angepasst, sodass sich diese im Frontend nebeneinander verhalten.

Zu Beginn meiner Komponenten wurde der Content-, Prob und h Decorator in der Tsx-Datei von "*@stencil/core"* importiert. Bei den Meta-Informationen, *@component*, wird die Komponente mit dem Tag *article-nina*, der Style-URL *article-nina.css* und dem shadow-DOM noch einmal zusammengefasst.

Bei dieser Komponente wurden die Properties des Artikel-Teasers mit dem Property-decorator angelegt, die durch die Klasse *ArticleNina* exportiert werden. Bei fast allen angelegten Properties handelt es sich um Strings, da vor allem Text an den Teaser übergeben werden soll. Bei dem Teaser-Image und dem *"read-more"*-Pfeil wiederum wurde der Name der Bilddatei genannt, die sich im Ordner *assets* befinden. Es wird hier die Property *teaser* deklariert, welche *public* ist und man auf diese Weise auch von anderen Klasse aus auf diese zugreifen kann. Durch den Array mit der Typisierung *any* werden hier innerhalb von diesem die Namen der Teilelemente des Teasers aufgelistet und bei den verwendeten Images, mit den imageUrls bzw. Pfad der Datei, hinterlegt. Bei der *render* Methode wird das Ganze zurückgegeben und die vorher definierten Properties werden hier über *this.* aufgerufen.

Aufgebaut wurde die Komponente zunächst mit einem Rahmen in Form eines Divs, der die Klasse *article* besitzt und das Element zunächst umschließt. Da ich den Teaser komplett verlinken wollte, habe ich deshalb am Anfang bzw. am äußersten Div den Link eingebaut um die zusätzlichen Inhalte des Artikels ebenfalls zu verlinken. In diesem Fall wird zur Veranschaulichung der Funktionalität auf die DHBW Webseite weitergeleitet. Dies kann natürlich individuell angepasst werden.

Der Div mit der Klasse *article-card* wurde verwendet, um den Inhalt des Artikels noch einmal zusammenzufassen. Oberhalb des Elements befindet sich das Teaser-Image in Form eines Placeholders, das die Id *title-img* besitzt und mit dem Property image und mit der Javascript Funktion *this.image* aufgerufen wird. In dem Array wurde das Property *Image Url* in Form eines Strings angeben um den Pfad des Bild-Elements nachzuverfolgen.
Unterhalb des Images befindet sich ein weiterer Div mit der Klasse *upper-article*. Dieser wurde eingesetzt, um die oberer Reihe, in welcher mehrere Elemente mit einem *span* nebeneinander gesetzt werden, zu kennzeichen. Dementsprechend wurde der *span* mit der Klasse *article-brand* und dem *span* mit der Klasse *article-theme* eingefügt. *This* ist der Verweis auf das Objekt, dass die Javascript-Funktion aufgerufen hat. Somit wird also mit *this.brandname* das Property *brandname* aufgerufen, bei dem es sich um einen String handelt. Bei dem Aufruf dieser Funktion wird dieses Argument zusammen mit dem Zeiger auf das angegebene Objekt, das eben die Funktion aufrief, übergeben.
Dem Teaser-Thema, das beschreibt, um welche Kategorie sich der Artikel handelt, wird hier die Klasse *article-theme* zugewiesen. Hier wird das Property *theme* mit dem *this* Verweis aufgerufen, bei welchem es sich ebenfalls um einen String handelt. Die Headline des Artikels Elements, mit der die Überschrift des Teasers gemeint ist, wird in einem Div mit der Klasse *article-headline* gesetzt. Mit *this.headline* wird dieses Property aufgerufen. Unterhalb der Headline ist der Teaser-Text platziert, der dem Leser einen Einblick in den eigentlichen Artikel geben soll. Auch hier befindet sich dieses Element wieder in einem Div, welchem die Klasse *article-text* zugewiesen wurde. Das Property mit dem String *text* wird über *this.text* aufgerufen. Durch den Div mit der Klasse *under-article* wird die untere Reihe des Teasers markiert, die mehrere Elemente enthält. Eines dieser Elemente ist das Erscheinungsdatum des Artikels. In dem Span mit der Klasse *article-date* wird dieses Datum, mit dem String *date* aufgerufen. Ebenfalls in diesem *span* enthalten ist das Property *read-more*, welches noch einmal mit einem *href* Attribut verlinkt wurde, nur wird hier der Text beim Hovern über das *more* unterstrichen. Der Pfeil rechts von dem Text ist ein Pfeil platziert, der es nochmal verdeutlicht, dass durch das “more” mehr von dem Artikel zu lesen ist. Dieser Pfeil, bei welchem es sich um ein Bild mit der Id *arrow-img*, handelt, wird mit dem Property *arrow* und der this Funktion aufgerufen.

Persönliche Reflexion

Das erste Mal mit StencilJS zu arbeiten empfand ich als große Herausforderung. Da ich auch bei anderen Programmen wie Javascript oder Typescript anfänglich große Schwierigkeiten hatte, mit dem Programm zurecht zu kommen, fiel es mir bei Stencil ebenfalls nicht leicht. Auch wenn Stencil auf Javascript aufbaut und es deshalb keine drastische Umstellung gibt, fand ich das Hineindenken ziemlich kompliziert. Durch das Anschauen zahlreicher Videos und Lesen von Dokumentationen konnte ich zwar die meisten Probleme lösen, blieb aber dennoch bei dem ein oder anderen Problem hängen, welches ich nur durch die Hilfe meiner

Gruppenmitglieder lösen konnte.
Da mein Artikel- Element relativ simpel aufgebaut ist und es neben einem Link keine Funktion besitzt, war das Erstellen dieses im Vergleich zu meinem Cookiebanner-Element relativ einfach. Zwar dauerte dieses länger, da es meine erste Komponente war mit der ich mich beschäftigt habe und ich mich erst hineinfinden musste, allerdings ist sie einfacher strukturiert. Ich bin mir sicher, man hätte diese Komponente noch einfacher aufbauen und mit weiteren Funktionalitäten versehen können. Eine Idee, die ich jedoch aufgrund der fehlenden Zeit nicht umsetzen konnte, war ein Slider der einzelnen Teaser-Elementen einzubauen, sodass diese nicht statisch platziert sind, sondern man zwischen mehreren Elementen vor-und zurückspringen kann.

Cookiebanner: Nina Eberle

Idee

Die zweite meiner Komponenten zeigt einen Cookiebanner in der unteren Seite des Displays mit dem Text “Wir haben Cookies auf deinem Computer platziert. Unsere Cookies werden verwendet, um Inhalte und Werbung zu personalisieren.”, sowie mit dem zu erkennbaren Link mit dem Text “ Mehr Erfahren”. Durch das Hovern über den Link ändert sich die Farbe zu einem helleren Grau und der Cursor ändert sich zu einem Pointer. Durch Klicken auf den Link wird man zur DHBW Webseite weitergeleitet, da es sich bei diesem Link nur um ein Beispiel handelt und gegebenenfalls individuell angepasst werden kann. Auf der rechten Seite des Banners sieht man einen “OK”-Button, welcher ebenfalls seine Hintergrundfarbe und den Curser, beim Hovern über diesen, ändert. Durch Betätigen des Buttons schließt sich der Cookiebanner und kann erst beim erneuten Laden bzw. beim Aufrufen der Seite wieder angezeigt werden.

Praktische Umsetzung

Importiert werden zunächst der Component-, der Property- der State- der Event- und der EventEmitter Decorator. Die Komponente wird mit Hilfe von @Component, dem component decorator zusammengefasst. Der Tag hierbei lautet *cookiebanner-nina*, die StyleUrl wird übernommen von *cookiebanner-nina.css* und es wird der shadow DOM verwendet. Exportiert wird hier die Klasse *CookiebannerNina*, die die Properties *text*, *linktext*, *link* und *button*, sowie den State *schliessen* und das Event buttonSchliessen, beinhaltet. Bei den definierten Properties handelt es sich lediglich um Strings. Bei *State() schliessen* wird ein *boolean* mit dem Wert *true* verwendet, sodass sich der Banner schließen lässt.
Bei dem *Event() buttonSchliessen* wird ein *EventEmitter* verwendet. Dieses wird hier verwendet, da es anderen Code-Bestandteilen über das Eintreten des bestimmten Ereignisses, nämlich über das Schließen des Banners, informieren will.

Auch in dieser Komponente wird ein *public* Property mit dem *interface Array* erstellt, in welchem alle Typen angelegt werden können. Innerhalb dieses Arrays werden die Strings *cookiebanner-text*, *cookiebanner-link-text*, *cookiebanner-link* mit der URL, die auf die DHBW Webseite verlinkt und *cookiebanner-button* angelegt.
Um den Cookiebanner schließen zu können benötigt man hier die Methode *eventSchließen*, welche keinen Rückgabewert zurückgibt und deshalb auf *void* gesetzt wird. In der Funktion *this.schliessen* wird die *state property* aufgerufen, die den Banner schließt. Das Event *buttonSchliessen* wird ausgeführt, sobald der *OK*-Button betätigt wird. Da der *boolean* bei dem *state property* auf *true* gesetzt wurde, und er durch auslösen des Events nun auf *visible* steht, wird die Funktion *this.schliessen* ausgeführt. Innerhalb der render()-Methode wird der Aufbau und der Abruf der Funktionen zurückgegeben. Umschlossen wird die Komponente zunächst durch einen Div mit der Id *footer-cookie*.
In dieser Methode wird zunächst dem Div die Id *footer-cookie* zugewiesen. Hinter *class* wird sich mit *this.schliessen* auf das *state* property *schliessen* (*boolean* ist auf *true)* bezogen. In diesem Fall ist das Property auf *true* gesetzt und eingesetzt wird also die Klasse *footer-cookie*. Wird der boolean auf *false* gesetzt, hat man die Klasse *inactive*. Man besitzt in dieser Zeile also eine If-Bedingung (Ternary Expression) in dem Klassennamen . Wenn man also diesen Button mit dem Event *buttonSchliessen* klickt, setzt man das *state* Property *schliessen* auf *false* und so wird in der Klasse *inactive* “eingesetzt”. Solange also *schliessen* auf *true* ist, wird das eingesetzt was vor dem Doppelpunkt steht, also *footer-cookie*.
Durch das Einbinden eines *spans* wird die Beschreibung des Banners und der Button nebeneinander platziert. Die Beschreibung, die die Id *description* besitzt, wird durch die *this.text* und die *this.link* Funktion zurückgeben. Dem Button, durch den sich der Banner schließen lässt, wird die Id *accept* zugewiesen. Durch das *onClick* event wird die Funktion ausgelöst, sodass der Banner geschlossen wird. Mit der diesem *onClick* Event wird also die Methode *eventSchliessen* aufgerufen.

Persönliche Reflexion

Da ich diese Komponente im Anschluss geschrieben habe, fiel mir der Aufbau und der Umgang mit den einzelnen Properties deutlich leichter. Durch die Orientierung an meiner vorherigen Komponente oder an denen meiner Gruppe, konnte ich den Aufbau meines Cookiebanners relativ einfach lösen. Allerdings wurde bei dieser Komponente mehr benötigt, was die Logik anging. Bei dem Schreiben der Funktionen, z.B. bei dem *buttonSchliessen* Event, welches regelt, dass der Cookiebanner geschlossen wird, musste ich einiges recherchieren. Da ich jedoch bei meiner Internetrecherche nur auf wenige brauchbare Informationen gestoßen bin und oft auch keine nützlichen Antworten bekam, die mich weiter brachte, wusste ich manchmal nicht, wie ich an manchen Stellen weiter machen sollte.

Allerdings konnte ich mit Hilfe meiner Gruppenmitglieder das ein oder andere Problem schnell lösen.

Nachbereitung: alle Autorinnen

Zu der Nachbereitung gehörte, dass wir regelmäßig Git Konflikte gelöst haben. Dabei haben wir nicht nur Merge Konflikte gelöst, sondern haben uns auch bei den ein oder anderen Schwierigkeiten mit Git sofort ausgetauscht und Probleme gemeinsam behoben. Zur Nachbereitung zählte ebenfalls, dass wir unsere eigenen Komponenten zusammenbringen mussten. Durch das Mergen der Komponenten trat das Problem auf, dass diese nicht alle richtig zueinander positioniert waren. Im Vergleich zu dem Entwurf, welcher Katharina in XD ursprünglich angefertigt hatte, konnten wir feststellen, dass einiges noch nicht optimal platziert war.Aus diesem Grund mussten diese verändert und auf ein passendes Format angepasst werden. Ein weiterer Teil der Nachbereitung war es, die jeweiligen Dokumentationen der Komponenten, als auch die gemeinsame Dokumentation zu schreiben. Hierbei haben wir uns in mehreren Meetings ausgetauscht, unsere Idee in Form von Stichpunkten gesammelt und uns aufeinander abgestimmt. Gemeinsam wurde diese Ideen in Sätzen ausformuliert und gegenseitig verbessert. Die Formatierung des Dokumentes wurde von Alischa Thomas übernommen.

Gruppenfazit: alle Autorinnen

Alles in Allem war diese Gruppenarbeit eine positive Erfahrung für jede von uns. Dennoch empfanden wir manche Aspekte während der Gruppenarbeit eher negativ oder hinderlich. So entstand zum Beispiel ein einziges Chaos, als alle Komponenten zusammengefügt werden sollten, da jeder in der Gruppe seine bevorzugten Konzepte in CSS verwendet hatte. In StencilJs ein solches Chaos zu entwirren war sehr zeitaufwändig und wir alle bekamen Zweifel an den eigenen CSS-Kenntnissen. An sich empfinden wir stencil.js als eine schöne Idee und sehen klar seine Vorteile beim Entwickeln von Webseiten, jedoch fehlt unserer Meinung nach in der grundsätzlich vorhandenen Dokumentation der Bezug zu anderen Dokumenten, wie CSS. Auch fehlte uns hierbei die Unterstützung bei Fehlerbehebungen. Viele Fehler wurden nicht als solche angemerkt, weder in der Konsole, noch in den jeweiligen Dateien. Diese Tatsache in Kombination mit unseren geringen Vorkenntnissen hat das Beheben der Fehler oft sehr in die Länge gezogen.

Desweiteren kamen für uns die Tutorials des Dozenten etwas zu spät. Da gerade am Ende des Semesters viele Aufgaben aufeinanderfallen und Deadlines verschiedener Module sich häufen, wollten wir diese Gruppenarbeit so früh wie möglich anfangen und umsetzen, sodass wir beim Erscheinen des ersten Tutorials schon, vom Zusammenfügen unserer Komponenten abgesehen, fast fertig waren.

Unabhängig vom eigentlichen Programmieren sind wir sehr zufrieden mit unserem Projekt. Die Aufgabenverteilung und das Zeitmanagement waren kein Problem. Unsere eigenen gesetzten Deadlines hielten wir alle ein, sodass wir vor der Abgabe noch genug Zeit hatten, um auftretende Probleme zu lösen. Auch konnten wir uns dadurch ohne Zeitdruck der Dokumentation widmen. Zusätzlich zum organisierten Zeitmanagement haben die anfangs erstellten Wireframes in *Adobe XD* bei der Orientierung im Big Picture des Projektes geholfen. So sind wir der Gefahr aus dem Weg gegangen sind, dass unsere Komponenten am Ende optisch nicht zueinander passen oder die Größenverhältnisse nicht stimmen. Vor allem das Teamwork hat sehr gut funktioniert. Jede von uns war bei Nachfrage sofort bereit den eigenen Code zu erklären und Hilfe zu leisten. So konnten wir alle unsere Probleme gemeinsam schnell und effizient lösen. Lediglich die Anpassung unserer Webseite an verschiedene Formate blieb offen, da unsere Prioritäten wegen der Schwierigkeiten, die beim Programmieren aufkamen, andere waren. So waren für uns zum Beispiel die Funktionalitäten der einzelnen Komponenten wichtiger.

Doch wie zu Beginn in diesem Kapitel beschrieben, sind wir trotz Probleme und Komplikationen zufrieden mit unserer Arbeit und konnten viel daraus lernen. Ein wichtiger Punkt ist hierbei das zielorientierte Arbeiten. Da es für spezifische oder gezielte Vorhaben eigentlich nie All-In-One-Lösungen online gab, musste man auf verschiedensten Seiten recherchieren. Um aus diesen Quellen dann auf eine Lösung des eigenen Problems zu kommen, mussten wir schnell lernen, dass wir vorab genau wissen müssen, was wir von unseren Komponenten erwarten. Allgemein können wir von uns behaupten, unsere jeweiligen Programmierkenntnisse verbessert zu haben, gerade in Bezug auf Code-Duplikationen, die stencil.js oft einfach nicht erlaubt.

Letztendlich würden wir alle aktuell stencil.js nicht weiter verwenden. Denn zum einen schätzen wir unsere momentanen Programmierkenntnisse als zu gering ein und weil es Alternativen gibt, die wir leichter in der Syntax und in der Umsetzung empfinden. Und zum anderen, weil unsere Unternehmen andere Frameworks benutzen. Dennoch gibt es uns ein gutes Gefühl etwas zusammen erschaffen zu haben, das man auch in Zukunft wiederverwenden kann.

Quellen: alle Autorinnen

Decorators—Stencil. (n.d.). Retrieved April 29, 2020, from https://stenciljs.com/docs/component

Events—Stencil. (n.d.). Retrieved April 29, 2020, from https://stenciljs.com/docs/events

Getting Started With Stencil—A Web Component Generator. (n.d.). Retrieved April 30, 2020, from https://www.youtube.com/watch?v=K1CvJI4DGPU&t=800s

javascript—How Nested Components Works in Stencil? (n.d.). Stack Overflow. Retrieved April 29, 2020, https://stackoverflow.com/questions/48474915/how-nested-components-works-in-stencil

State Decorator-Stencil. (n.d.). State Decorator. Retrieved April 30, 2020, from file:///Users/ThomasA/Zotero/storage/V4ESC4BS/state.html

Stencil | Props, State & Tons of Decorators. (n.d.). Retrieved April 29, 2020, from https://www.youtube.com/watch?v=g4jsOCUD4lA&list=PLYG3A_NWKhq4jqOkpNXhFrQs3i7Gizr_k&index=13&t=0s

Styling Components—Stencil. (n.d.). Retrieved April 29, 2020, from https://stenciljs.com/docs/styling

StencilJS fundamentals. (n.d.). Retrieved May 07, 2020, from https://www.youtube.com/watch?v=SCB3X2ApYWc&t=601s

Anhang: alle Autorinnen

Referenz Onlineblog—Sallys Blog

https://sallys-blog.de/

Referenz Onlinezeitung—New York Times Buisness

https://www.nytimes.com/section/business

Referenz Onlinezeitung—New York Times Readercenter

https://www.nytimes.com/2020/03/30/reader-center/afghanistan-girls-taliban.html

Online-Dokument—Vorbereitung Komponenten

https://drive.google.com/open?id=1jmUlrokuiRL2oqJ9NcG3BSpcsNhf2fGUmx8Snmws-JM

Online-Dokument—Wireframe Komponenten

Alischa: https://xd.adobe.com/view/f42e4329-dace-42e0-7eec-e2880252123f-821f

Larissa: https://xd.adobe.com/view/9ad3bde1-eda8-42b3-4f85-06d61005e715-ece2

Nina: https://xd.adobe.com/view/44c83bc3-375b-4457-6c25-30577fdb6dfa-32f9/

Online-Dokument—Gesamt Wireframe

Katharina: https://xd.adobe.com/view/2af6e59f-d178-4fe4-7287-5ff215b814a6-5cc7/