# Gruppendokumentation Marco, Christian, Katharina, Pascal

# Dokumentation Marco Scotellaro

## Web-Komponente 1 – Footer

Zu Beginn habe ich zuerst die Komponente mit „@Component“ definiert. Da ich hier in dieser Komponente den Footer darstellen wollte, benannte ich die Komponente „my-footer“. Die styleUrl habe ich ebenfalls „my-footer“ genannt. Ich habe den shadow auf true gesetzt, da ich hier in dieser Komponente ein abgegrenztes Styling unabhängig vom Styling der HTML Seite haben wollte.

Die zu exportierende Klasse heißt MyFooter. In ihr wird der Footer gerendert und gibt das Div mit dem Impressum als Schrift und Anker zurück, welches dann den Inhalt der Komponente bilden.

Hier kam ich zu einem großen Problem, welches mich im fixen etwas Zeit gekostet hat. Ich habe das Rendering mit einem div gebaut, welches ein <a>-Tag und ein <p>-Tag beinhaltet. Das <p>-Tag besitzt den Inhalt „Impressum“ und soll durch das <a>-Tag zum Impressum leiten. Jedoch kam man, wenn man die Seite per Klick aufrufen wollte nur auf eine Fehlerseite und die eigentliche Seite des Impressums wurde nicht gefunden.

Daher wurde zuerst eine leere Variable initialisiert. Diese Variable verändert sich, da folgendes Problem besteht. Wenn der User der Seite auf der Startseite ist, benötigt er einen anderen Link, wie wenn sich der User auf einer Unterseite befindet. Wenn der User auf der Startseite ist, dann kommt der User zum Impressum, da nur einmal HTML vorangestellt wird. Wenn der User sich allerdings auf einer Unterseite befindet wird zusätzlich ein HTML vorangestellt und der User gelangt nicht mehr auf das Impressum. Daher benötige ich die if-Bedingung. Ich frage hier ab, ob sich der User auf der Startseite befindet mit dem „window.location.href=="html"“. Dann soll der Link leer sein + der Link zum Impressum wie im Code Zeile 18 veranschaulicht ist. Wenn der User allerdings auf einer Unterseite ist, soll die Variable link auf die Startseite zurückdeuten und dann erst das Impressum aufrufen. Dadurch umgeht man das Problem, dass dem Link zweimal HTML vorangestellt wird.

Daraufhin habe ich den Footer gestylt per CSS. Ich habe den Footer fixed am unteren Bildschirmrand angebracht. Wir haben uns in der Gruppe zuvor darauf geeinigt die Farben Schwarz, Weiß und Rot zu verwenden. Daher ist die Farbwahl des Footers auf schwarz gefallen und die Schrift dazu im Kontrast weiß. Ich habe die Schrift auf die vertikale Höhe platziert und auf die rechte Seite, sodass das Wort zirka rechts auf einer Höhe mit dem äußersten Wort der Nav-Bar ist.

Der Grund für diese Komponente ist, dass man von jeder Seite auf das Impressum zugreifen kann.

## Web-Komponente 2 – Flexdiv

Bei der zweiten Webkomponente habe ich ebenfalls zu Beginn wieder die Webkomponente mit @Component definiert. Der Tag und die styleUrl wurde auf das flexdiv angepasst und ebenfalls aus den gleichen Gründen wie bei der 1. Webkomponente wurde wieder der shadow auf true gesetzt.

Dieses Mal ist die Klasse etwas kleiner als bei der 1. Webkomponente. Die Webkomponente besitzt ebenfalls wieder eine zu exportierende Klasse, welche dieses Mal den Namen MyFlexdiv besitzt. In ihr wird das zu rendernde Element beschrieben. Das Element besteht aus einem Div, welches die Inhalte der Unterseiten strukturieren soll in einer Flexbox. Ich habe hier im <div>-Tag ein <slot> eingebaut. Dieser Slot sorgt dafür, dass der Inhalt selbst eingesetzt werden kann.

Entscheidend für die Webkomponente ist allerdings das CSS der Komponente. Ich habe hier ein „display: flex;“ eingebaut. Die Richtung sollte in einer Reihe sein, da die Inhalte in Divs nebeneinander angeordnet werden sollen und immer am Beginn der Flexbox angeordnet sein sollen.

## Zusätzliches nach der Erstellung

Wir haben uns als Gruppe darauf geeinigt die Überarbeitung und die Strukturierung der Seiten (sofern benötigt) aufzuteilen. Ich für meinen Teil arbeitete dann mit der Startseite und der Unterseite Geschlossenheit.

Angefangen mit der Startseite habe den Inhalt der Startseite nochmal in ein Div mit der Klasse „.contentAlign“ gepackt. Diese Klasse platzierte den Inhalt der Seite mittig per Flexbox-Struktur und platzierte den Content der Seite in die Mitte des Bildschirms. Ebenfalls hatte die <my-headline> ein Margin, welches ich um einen Minus-Wert verändert habe, sodass der gesamte Content in der Mitte platziert ist.

Als nächstes kümmerte ich mich um die Anordnung auf der Unterseite Geschlossenheit. Ich habe hier auch eine Beispielstruktur für die Verwendung der Elemente in der richtigen Reihenfolge erarbeitet und versucht das Ganze im HTML-Code anschaulich darzustellen. Die bereits erstellten Tags aller Kommilitonen aus der Gruppe habe ich in der richtigen Reihenfolge eingebaut. Diese waren bereits in einem Template-Versuch vorhanden, welchen wir verwarfen, da es für die Aufgabe nicht relevant war verworfen haben. Die Struktur bezüglich der Flexbox hat allerdings nicht funktioniert. Die Inhalte wurden stupide untereinander angezeigt. Also habe ich manuell die Divs erstellt, worin die Inhalte eingepflegt werden müssen um von der Flexbox organisiert zu werden. Das funktionierte soweit auch, jedoch war kein Abstand zwischen den Elementen vorhanden. Ich habe also den Divs Abstände gegeben und ebenfalls eine feste Breite, da die Breite von Div zu Div ansonsten unterschiedlich war.

Den Inhalt für die Startseite habe ich mir selbst ausgedacht. Den Inhalt für das Gesetz der Geschlossenheit habe ich von folgenden Websites kopiert:

-https://www.klickkomplizen.de/blog/print-design/gesetz-der-geschlossenheit/

-https://www.toushenne.de/design/gestaltgesetze-der-wahrnehmung.html

-https://www.webmasterpro.de/design/article/gesetz-der-geschlossenheit.html

# Reflexion

Vor der Vorlesung hatte ich noch nie etwas von dem Framework Stencil gehört. Für mich war es in der Vorlesung sehr schwer den Überblick zu behalten über die Bereiche JavaScript, TypeScript und Stencil. Daher habe ich zu Beginn unseres internen Assignment-Starts erneut die Vorlesung angeschaut und das Vorlesungsskript zu Stencil durchgelesen. Für mich war vor allem die Schwierigkeit zu verstehen wie die Komponenten zusammenhängen, warum eine Komponente so erstellt wird, wie sie erstellt wird und wieso man überhaupt solche Webkomponenten benötigt.

Als ich mich dann selbst rangemacht hatte an die Erstellung meiner Komponenten habe ich zuerst etwas Zeit gebraucht um zu verstehen wie ich eine Komponente überhaupt funktionstüchtig bekomme. Daher habe ich auch für die Erstellung der ersten Komponente viel mehr Zeit benötigt als für die Erstellung der 2. Komponente.

Besonders gut gelaufen für meinen Teil ist vor allem das Erarbeiten der Flexbox. Der Inhalt hat sich mit ein paar Nachbesserungen super organisieren lassen.

Als Recap zu meinem Wissenstand vor dem Assignment kann ich von mir behaupten, sehr viel gelernt zu haben. Ich kann jetzt eigene Komponente anlegen und verstehen den Sinn und Verwendungszweck von Stencil.

Für die Zukunft und zukünftige Projekte würde ich mich an die Erstellung eines Templates wagen und versuchen das Template funktionstüchtig zu bekommen. Sozusagen als Steigerung zur Erstellung von Webkomponenten. Ich habe mit unserem Dozenten eine Stunde versucht unser bisher fehlerhaftes Template zu fixen, jedoch rannte die Zeit weg und deshalb beschlossen wir ohne das Template an sich zu arbeiten. Ich bedanke mich also in dieser Ausarbeitung nochmal für die Hilfe bei unserer Arbeit, da Probleme und offene Fragen gut geklärt werden konnten.

# Dokumentation Christian Dänzer

Ziel dieser Web-App ist es, durch die Gestaltgesetze zu navigieren. Zu jedem wird mittels Beispiele und Begleittext das Gesetz erklärt.

## My-button.tsx

Der Tag wird dem Shadow-DOM mittels „shadow = true“ zugewiesen. Sinn des Buttons ist es, von der aktuellen auf die nächste Seite zukommen. Dadurch soll ein Rundlauf durch die Gestaltgesetze ermöglicht werden. Dazu ermittle ich zunächst die URL. Das passiert jeweils zu Beginn der Funktionen. In der Funktion „setButtonTagID()“, die beim rendern aufgerufen wird, begrenze ich sie durch regex ausschließlich auf den Dokumentnamen + Dateiendung. Durch einen control-flow weise ich dann die passende Zahl der Variable „buttonID“ zu. In der Funktion „goToNextPage()“, die durch „onclick“ aufgerufen wird, überschreibe ich dann die aktuelle URL, nachdem ich je nach buttonID den passenden Pfad angegeben habe. Hier arbeite ich wieder mit einer „.replace“-Funktion und regex. Wenn der User auf der letzten Seite des Rundgangs angekommen ist, erscheint ein Hinweis und er landet wieder auf der Startseite. Den Text des Buttons setze ich, ebenfalls wie das Ziel, variable. Dazu wird in der Funktion „setContent()“ je nach URL der passende Content generiert.

## My-Headline.tsx

Dieser Component wird mittels „shadow = true“ ebenfalls dem Shadow-DOM zugewiesen. Bei my-headline ermittle ich die URL über die Funktion „getURL()“, damit ich den Inhalt variabel korrekt anpassen kann. Befindet sich der User auf einer Seite eines Gestaltgesetzes, wird zusätzlich „Gesetz“ im Content angezeigt. Im Render-Prozess werden dann alle „ae“s zu „ä“s, bei Fortsetzung wird noch „guten“ angehängt und die passenden Artikel werden vergeben. Durch mehrere Control-Flows wird dadurch jede einzelne Headline korrekt vergeben, ohne, dass der User Text einfügen muss.

## My-subheadline.tsx

Auch dieser Component wird wieder dem Shadow-DOM zugewiesen. Ich habe einen Slot-Tag eingebaut, um bei der Erstellung eines Templates trotzdem noch Text variabel hinzufügen zu können. Außerdem kann der Content so ohne Properties zwischen den Tags eingefügt werden.

## Quellen

<https://angular.io/guide/template-syntax>

(zuletzt aufgerufen am 18.04.20)

<https://upmostly.com/tutorials/react-onclick-event-handling-with-examples>

(zuletzt aufgerufen am 18.04.20)

and of course: <https://stackoverflow.com/>

(zuletzt aufgerufen am 18.04.20)

# Reflexion

Ich habe bislang noch nicht mit StencilJS gearbeitet und musste mich gerade zu Beginn erst mal durch den Aufbau kämpfen. Die Arbeit mit den Components selbst war dann gar nicht mehr so schwer. Allerdings galt es auch hier wieder, sich mit den einzelnen Eigenschaften auseinanderzusetzen. Gerade die Interpolation-Notation hat mir bei der Gestaltung der Buttons sehr geholfen. Dadurch konnte ich sowohl die ID variabel übergeben, als auch die Funktion beim Klicken ausführen. Ansonsten habe ich diese Notation eigentlich in jedem Component für den Inhalt genutzt. Wenn ich irgendwas Sinnvolles mit regex machen kann, freut es mich auch immer. Ich habe hier noch Übungsbedarf und nutze solche Gelegenheiten dann sehr gerne, um meine Fähigkeiten zu verbessern.

Mein größtes Problem galt dem Methodenaufruf innerhalb anderer Funktionen. Da ich hier keine zufriedenstellende Lösung gefunden habe, bin ich ein Kompromiss eingegangen. Anstatt unterschiedliche Schritte in Funktionen auszulagern, habe ich sie in mehrfach in anderen Funktionen aufgerufen. Das hat mich zwar nicht ganz so zufrieden gestellt, aber dafür hat es funktioniert.

# Dokumentation Katharina Schmitt

# Eigenreflexion

Bevor meine Gruppe das Arbeiten begonnen hat, haben wir uns alle selbstständig nochmal Stencil angeschaut: <https://www.youtube.com/watch?v=MqMYaT1GlWY>, <https://www.youtube.com/watch?v=SCB3X2ApYWc>.

Das hat mir sehr geholfen, da ich am Anfang erstmal wenig verstanden habe. Nachdem ich mir aber nochmal näher angeschaut habe, wie Stencil funktioniert, habe ich schnell gemerkt, dass es gar nicht so schwer ist, wie ich im ersten Moment dachte. Unsere Gruppe hat sich dann getroffen und ein grobes Scribble unserer Startwebseite und den Unterwebseiten erstellt. Dabei haben wir auch direkt die verschiedenen Components definiert. Im nächsten Zug haben wir die Components unter den Mitgliedern aufgeteilt. Ich wollte gerne die Navigations-Bar machen und mich dafür entschieden. Außerdem haben wir darauf geachtet, dass jeder einen Component hat, der etwas anspruchsvoller ist und einen der leichter ist. Deshalb habe ich noch unseren „my-text“ Component und den „my-welcometext“ erstellt. Außerdem haben wir eine Schriftart und verschiedene Farben festgelegt, damit wir alle mit denselben arbeiten. Auf das genaue Design sind wir dann aber erst später eingegangen. Jeder war außerdem für den Inhalt von zwei Seiten zuständig. Ich habe hierbei die Seite Ähnlichkeit und Schicksal gemacht.

## My-text

Für diesen Component habe ich mir einen Ordner mit der tsx und css Datei erstellt. Für den Component brauche ich zunächst eine Property, die ein String ist, damit auf jeder Seite ein anderer Text eingefügt werden kann. Jedoch hat Marco mit dir hier nochmal etwas verändert, deshalb ist hier jetzt ein Boolean definiert. Danach erstelle ich als Component ein einfaches p-Element, das als Inhalt die davor definierte Property enthält. Auch diesen hat Marco mit dir noch einmal angepasst, da wir eigentlich Slots verwenden wollten. In der CSS-Datei habe ich nur die Schriftart angepasst und text-align. Dieser Component wir später dann in der „flexdiv“ verwendet.

## My-welcomeText

Dafür habe ich zunächst den „My-text“ Component kopiert. Hier kannst du eigentlich auch nochmal sehen, wie der Component „my-text“ eigentlich aussah. Hier habe ich aber ein div verwendet, damit der Absatz besser zu stylen ist. Danach habe ich das Sytling so gestaltet, dass ich unsere ausgewählte Schriftart angepasst habe und der Absatz mittig auf der Seite erscheint, in einem Block erscheint.

## My-nav bar

Da diese auf jeder Seite gleich sein soll und die Inhalte nicht angepasst werden sollen, habe ich keine Properties verwendet. Zunächst habe ich also eine ganz normale Navigationsbar über eine Liste erstellt und auch bereits ein Sytling gestaltet. Ich habe mich als erstes für schwarz als Hintergrundfarbe und weiße Schrift entschieden, die beim hovern über die einzelnen Elemente rot wird. In der Navigationsbar ist für mich dann das Problem aufgetreten, dass wenn man bereits auf einer Unterseite ist nicht mehr „weiterkommt“, da der hinterlegte Link dann nicht mehr passt. Also habe ich eine If-Funtkion erstellt. Diese sah zunächst folgendermaßen aus.

export class MyNav {

render() {

if (window.location.href=="html"){

    return <header>

      <nav>

        <ul class="navbar">

          <li><a href="html">Start</a></li>

          <li><a href="html\aehnlichkeit.html">Ähnlichkeit</a></li>

          <li><a href="html\geschlossenheit.html">Geschlossenheit</a></li>

          <li><a href="html\fortsetzung.html">Gute Fortsetzung</a></li>

          <li><a href="html\naehe.html">Nähe</a></li>

          <li><a href="html\praegnanz.html">Prägnanz</a></li>

          <li><a href="html\schicksal.html">Schicksal</a></li>

        </ul>

      </nav>

    </header>;

    }else{

      return <header>

      <nav>

        <ul class="navbar">

          <li><a href="..\html">Start</a></li>

          <li><a href="..\html\aehnlichkeit.html">Ähnlichkeit</a></li>

          <li><a href="..\html\geschlossenheit.html">Geschlossenheit</a></li>

          <li><a href="..\html\fortsetzung.html">Gute Fortsetzung</a></li>

          <li><a href="..\html\naehe.html">Nähe</a></li>

          <li><a href="..\html\praegnanz.html">Prägnanz</a></li>

          <li><a href="..\html\schicksal.html">Schicksal</a></li>

        </ul>

      </nav>

    </header>;

    }}

}

Ich habe in „If“ abgefragt, ob ich mich auf der Startseite befinde und dann dafür die Links wie ich sie zuvor hatte ausgegeben. Wenn ich mich aber nicht auf der Startseite befinde, soll der Link erst nochmal einen Schritt zurück. So können wir auch von jeder Unterseite eine andere aufrufen und gleichzeitig wird in der URL nicht der ganze Weg angezeigt, sondern nur die Startseite und die aktuelle Unterseite. Dadurch wird diese nicht extrem lang, wenn der User zwischen den Seiten wechselt und ist immer übersichtlich.

Christian hat mir dann noch den Tipp gegeben, dass ich einer Variable auch verschiedene Werte zuweisen kann. Dadurch konnte ich den Code nochmal kürzen, wie er dann auch aktuell im Script zu sehen ist. Dazu habe ich eine Variable „link“ erstellt, die in der If-Funktion den Anfang des Links enthält. Wenn ich mich also auf der Startseite befinde, ist die Variable leer auf einer Unterseite enthält sie aber „einszurück“. Dadurch kann ich danach nur einmal die Ausgabe der Navigtionsbar schreiben und vor den Link die Variable setzen.

Damit habe ich auch nochmal Marco geholfen und wir haben im Footer für das Impressum das gleiche Prinzip angewendet.

# Eigenreflexion

Nachdem wir alle unsere Components erstellt hatten, haben wir uns nochmal genauer über das Design unterhalten und das bereits vorhandene angeschaut. Wir haben uns dann abgestimmt und das Design vollendet. Zuerst wollten wir ein Template erstellen. Marco hat mit dir dann aber abgeklärt, dass wir es auch ohne machen können. Dann hat jeder die Seiten für die er zuständig war mithilfe der Components aufgebaut. Bevor ich mich mit dem Thema Stencil genauer auseinandergesetzt habe, kam mir die Aufgabe sehr schwer und erstmal unübersichtlich vor. Im Nachhinein kann ich aber sagen es ist einfacher als gedacht. Von allen drei Assignments viel mir dieses am leichtesten. Hier war eine wichtige Rolle unsere Zusammenarbeit, die sehr gut funktioniert hat. Wir haben uns gegenseitig unterstützt und auch auf kleine Fehler oder Verbesserungen hingewiesen, wodurch jeder von uns nochmal etwas dazugelernt hat.

# Dokumentation Pascal Feinauer

## My-Definitiontext.tsx

## Ich habe eine Komponente mit @Component erstellt und habe ich diese Komponente my-definitiontext benannt. Die styleUrl habe ich ebenfalls my-definitiontext genannt, damit ich beides einander zuordnen kann. Den shadow habe ich auf true gesetzt, wodurch die Komponente dem Shadow-DOM zugewiesen wurde.

## My-Imagebox-left, My-imagebox-middle, My-imagebox-right

Für die Mini-Templates aus Bild, Subheadline und Text habe ich drei Komponenten erstellt, da der Style unterschiedlich ist (Margin-left variiert). Den shadow habe ich auf true gesetzt, wodurch die Komponente dem Shadow-Dom zugeiwesen wird. Die Komponenten sind jeweils ein Div mit zugehöriger ID und Klasse. Außerdem enthalten sie ein Slot-Element. Das ist nötig, damit der Inhalt später noch variabel übergeben werden kann. Mit dem Slot-Tag wird dann im HTML ein Bild, Subheadline und Text mit den passenden Tags eingefügt. Die Bilder haben dann jeweils noch ein Styling bekommen, damit die Texte danach auf der gleichen Höhe sind und nicht je nach Bildgröße verrutschen.

# Reflexion

Ich habe vor der ersten Vorlesung noch nie was von StencilJS gehört und folglich auch noch nicht damit gearbeitet. Zu Beginn des Projekts erging es mir ähnlich wie beim 1. Assignment. Die Vorlesung allein, hat mir nicht viel geholfen, ich musste mir wieder mal Hilfe in Tutorials suchen um StencilJS generell zu verstehen. Trotz zahlreichen Videos, die ich mir angeschaut habe, komme ich mit StencilJS immer noch nicht wirklich zurecht und habe somit Unterstützung von meiner Gruppe benötigt. Als wir dann gegen Ende ein gemeinsames Meeting hatten, konnte ich StencilJS etwas mehr verstehen, bin aber meiner Meinung nach noch zeimlich beim Anfänger-Niveau.

Aufgrund einer ungewohnter Situation zuhause, hänge ich dieses Semester generell ziemlich hinterher, was auch ein Grund ist, wieso ich mich bei StencilJS noch auf Anfänger-Niveau bewege. Dadurch, dass sich die Situation zuhause wieder beruhigt hat und ich während des Projekts gemerkt habe, dass man mit StencilJS ziemlich coole Sachen machen kann, werde ich mich in zukünftige Projekte mehr reinsteigern.