

# Technologieanalyse mit Angular und WebGL zur Umsetzung eines Konfigurators für Mehrwegbecher

Studiengang Medieninformatik

## Bachelorarbeit

vorgelegt von

**Lukas Michael Müller**

geb. in Schotten

durchgeführt in bei

Yellow Tree - Agentur für Kommunikation & Design, Siegen

Referent der Arbeit:	Prof. Dr. Benjamin Einert
Korreferent der Arbeit:	Prof. Dr. Stefan Euler
Betreuer bei Yellow Tree:	Dipl.-Ing. Andreas Utsch

Friedberg, 2019



*Für Yellow Tree*



# Danksagung

Normalerweise haben eine ganze Reihe von Personen mehr oder wenig Anteil am Gelingen der Bachelorarbeit, denen man hier dankt: In der Regel zunächst den Referenten und Betreuern der Arbeit. Aber natürlich auch Personen, Firmen und Institutionen, die die Arbeit tatkräftig unterstützt haben. Sei es durch die Bereitstellung von spezieller Hard- oder Software oder nur durch ein gewissenhaftes Korrekturlesen.



# Selbstständigkeitserklärung

Ich erkläre, dass ich die eingereichte Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Friedberg, April 2019

Lukas Michael Müller





# Inhaltsverzeichnis

<b>Danksagung</b>	<b>i</b>
<b>Selbstständigkeitserklärung</b>	<b>iii</b>
<b>Inhaltsverzeichnis</b>	<b>v</b>
<b>Abbildungsverzeichnis</b>	<b>vii</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problemstellung . . . . .	1
1.3 Zielsetzung . . . . .	2
1.4 Vorgehen bei der Umsetzung . . . . .	2
1.5 Aufbau der Arbeit . . . . .	3
<b>2 Stand der Technik</b>	<b>5</b>
2.1 Bestehende Lösungsansätze . . . . .	5
2.1.1 Flyeralarm . . . . .	5
2.1.2 Spread Shirt . . . . .	6
2.1.3 Becher-bedrucken.de . . . . .	7
2.2 3D Online Konfiguratoren . . . . .	7
2.3 Webframeworks . . . . .	9
2.4 Flash und WebGL . . . . .	11
2.5 3D JavaScript Bibliotheken . . . . .	12
<b>3 Grundlagen</b>	<b>15</b>
3.1 Single Page Anwendungen . . . . .	15
3.2 Framework Angular . . . . .	16
3.2.1 Komponentenbasierte Programmierung . . . . .	16
3.2.2 Modulare Umsetzung . . . . .	17
3.2.3 Grundfunktionen des Frameworks . . . . .	19
3.2.4 Angular CLI . . . . .	20
3.2.5 Versionen . . . . .	20
3.3 Three.js . . . . .	21

3.3.1	Die Szene . . . . .	22
3.3.2	Modelle . . . . .	23
3.3.3	Events . . . . .	23
3.4	Responsive Webdesign . . . . .	24
3.5	Usability . . . . .	24
3.6	Performance . . . . .	25
<b>4</b>	<b>Anforderungen und Konzeption</b>	<b>27</b>
4.1	Funktionale Anforderungen . . . . .	27
4.2	Nicht Funktionale Anforderungen . . . . .	29
4.3	Konzeption . . . . .	31
4.3.1	Ablauf einer Konfiguration . . . . .	31
4.3.2	Benutzeroberfläche . . . . .	32
<b>5</b>	<b>Umsetzung &amp; Implementierung</b>	<b>35</b>
5.1	Entwicklungsumgebung . . . . .	35
5.2	Anlegen des Angular-Projekts . . . . .	36
5.3	Module des Konfigurators . . . . .	37
5.4	Komponenten des Konfigurators . . . . .	38
5.5	3D Szene mit Model . . . . .	41
5.6	Design Rendering . . . . .	44
5.7	Mobile Umsetzung . . . . .	45
5.8	Daten-Service . . . . .	46
5.9	Upload-Vorgang . . . . .	46
<b>6</b>	<b>Testphase &amp; Ergebnisse</b>	<b>51</b>
6.1	Testumgebung . . . . .	51
6.1.1	Unit-Tests . . . . .	51
6.2	Ergebnisse . . . . .	52
<b>7</b>	<b>Zusammenfassung</b>	<b>55</b>
<b>A</b>	<b>Lorem Ipsum</b>	<b>57</b>
	<b>Glossar</b>	<b>59</b>
	<b>Literaturverzeichnis</b>	<b>61</b>

# Abbildungsverzeichnis

2.1	Mehrwegbecher bestellen . . . . .	5
2.2	Tasse bestellen . . . . .	6
2.3	Becher bedrucken . . . . .	7
2.4	Audi Konfigurator . . . . .	8
2.5	Siemens Konfigurator . . . . .	9
2.6	Google Trends Statistik Frameworks . . . . .	10
2.7	google Trends Statistics 3D Bibliotheken . . . . .	13
3.1	Angular Modul Beispiel . . . . .	18
3.2	Angular CLI Warnung . . . . .	21
4.1	Flussdiagramm . . . . .	32
4.2	Komponentendiagramm . . . . .	33
4.3	Mockup . . . . .	34
5.1	Anlegen eines Projekts . . . . .	36
5.2	Übersicht der Module . . . . .	37
5.3	<i>Erstellung einer Angular-Komponente mit PhpStorm</i> . . . . .	38
5.4	Sidebar des Konfigurators . . . . .	41
5.5	3D Becher Model . . . . .	43
5.6	Konfigurator mit designtem Becher . . . . .	45
5.7	UploadComponent des Konfigurators . . . . .	47



# Kapitel 1

## Einleitung

Mit dem folgenden Kapitel soll eine Einführung in das Thema gegeben werden. Es wird darauf eingegangen, warum dieses Thema relevant ist. Außerdem wird das Problem beschrieben, sowie das Ziel festgelegt. Am Ende des Kapitels wird auf den Aufbau der Arbeit eingegangen.

### 1.1 Motivation

Der Kunde *Gizeh*<sup>1</sup> bedruckt individuelle Mehrwegbecher. Große Druckportale, wie *Flyeralarm*<sup>2</sup>, senden ihre Aufträge an *Gizeh*. Dieser produziert dann die Becher, sowie den späteren Druck. Ein webbasierter Konfigurator könnte der Firma ermöglichen zukünftig Direktbestellungen aufzunehmen. Damit hätte der Kunde eine einfache Möglichkeit sein Design in einer 3D Ansicht zu sehen und ganz einfach zu konfigurieren. So hat man bevor der Becher bestellt wird schon einmal gesehen, wie es aussehen wird.

Schon in anderen Branchen, wie zum Beispiel auf dem Automarkt, wird dieses Prinzip der 3D Konfiguratoren angewandt.

### 1.2 Problemstellung

Wenn man einen Mehrwegbecher beispielsweise über *Flyeralarm* bedrucken lassen will, sollte ein fertiges Design vorliegen. Dabei gibt es keine speziellen Vorgaben bzgl. der kubischen Form. Man kann lediglich ein Design im PDF-Format hochladen. Nebenbei wird man darauf hingewiesen, das aufgrund der kubischen Form das Design gestaucht wird. Somit werden beispielsweise Kreise eventuell nicht ganz rund dargestellt.

Dem Kunden wäre es somit ein Vorteil, eine Vorschau des bedruckten Bechers anschauen

---

<sup>1</sup>**Gizeh Verpackungen** ist eine international tätige deutsche Unternehmensgruppe die Kunststoffverpackungen entwirft, fertigt und dekoriert. Das Unternehmen, dessen Stammsitz sich im nordrhein-westfälischen Bergneustadt befindet, beschäftigt gegenwärtig insgesamt etwa 750 Mitarbeiter und erwirtschaftete 2017 einen Umsatz von etwa 120 Millionen Euro.

<sup>2</sup>Die **flyeralarm GmbH** ist eine Online-Druckerei mit Sitz in Würzburg. Das Unternehmen ist auf die Herstellung und den Vertrieb von Drucksachen spezialisiert. Das Druckereiunternehmen gehört zu 100 Prozent dem Gründer Thorsten Fischer und ist in 15 europäischen Ländern präsent.

zu können. Die Lösung könnte ein 3D Konfigurator für Mehrwegbecher sein. Es gibt schon einige 3D Konfiguratoren. Allerdings besteht bei diesem die Frage: Wie kann der designte Becher optimal dargestellt werden? Und wie kann eine eventuell notwendige Anpassung des Designs erfolgen? Gerade wenn man den Aspekt der Responsivität hinzunimmt findet man keine bestehende Lösung, die zufriedenstellend wäre. Dieser Aspekt wird in Grundlagen Kapitel 3 unter dem Punkt 3.4 *Responsive Webdesign* genauer erläutert.

### 1.3 Zielsetzung

Im Rahmen der Arbeit wird ein 3D Konfigurator für Mehrwegbecher entwickelt. Dazu wird das Framework *Angular* und *WebGL* verwendet<sup>3</sup>. Er soll die Funktionalität haben, ein Design in einer 3D Vorschau auf einen Becher zu rendern. Eine optionale Funktion wäre das Anpassen des Designs durch zuschneiden oder transformieren. Dabei soll der Konfigurator auch auf mobilen Geräten verwendbar sein. Es ist wichtig das die Anwendung schnell und einfach zu bedienen ist.

### 1.4 Vorgehen bei der Umsetzung

**Entwicklungsumgebung** Wie schon erwähnt wird *Angular* verwendet. Das Framework eignet sich besonders gut, da hiermit die Anwendung gut testbar und wartbar umsetzbar ist. Bei der Entwicklung werden die Stärken und Schwächen des Frameworks aufgezeigt. Es wird versucht eine möglichst performante Anwendung zu programmieren. Als Editor wird *PhpStorm*<sup>4</sup> verwendet. Er bietet eine gute Implementierung von Angular Projekten und hat weitere nützliche Funktionen, die das Entwickeln vereinfachen.

**Design** Da der Konfigurator später in eine bestehende Webseite eingebaut werden soll<sup>5</sup>, wird sich das Design am Stil der Webseite orientieren. Genauere Vorgaben dazu gibt es nicht. Deshalb wird eine Aufgabe sein ein innovatives Design zu finden. Es soll den Ansprüchen der *Usability* gerecht werden.

**3D Szene** Nahezu jeder Browser unterstützt WebGL. Die *three.js* Library bietet dem Entwickler eine Abstraktionsschicht über *WebGL*, um benötigte 3D Szenen bauen zu können. Mithilfe dieser JavaScript Library wird der Becher mit dem hochgeladenen Design gerendert. Auf dieses Thema wird in Punkt 2.5 *JavaScript Bibliotheken* noch einmal Bezug genommen.

**Testumgebung** Am Ende der Entwicklung wird die entwickelte Applikation mit der Testumgebung von Angular untersucht und anschließend ein Fazit daraus gezogen. Das Framework liefert von Haus aus eine gute Möglichkeit um Unit-Tests sowie End-to-End-Tests durchzuführen.

---

<sup>3</sup>Es wird später noch genauer auf die Umsetzung eingegangen. Im Grundlagenkapitel werden die einzelnen Technologien genauer beleuchtet

<sup>4</sup><https://www.jetbrains.com/phpstorm>

<sup>5</sup>Dies ist nicht Bestandteil der Arbeit. Zuerst muss Kunde dem Projekt noch zustimmen. Anschließend müssen noch eventuelle Änderungen vorgenommen werden.

## 1.5 Aufbau der Arbeit

In dem Kapitel 2 *Stand der Technik* wird zunächst auf vorhandene Lösungen bzw. Lösungsansätze eingegangen. Es soll deutlich werden, warum diese nicht ausreichend sind. Außerdem werden Beispiele aus anderen Branchen angeführt, die eine gute 3D Konfiguration für andere Produkte ermöglichen. Kapitel 3 *Grundlagen* erklärt grob die verwendeten Technologien. Durch Beispiele wird gezeigt, warum diese sinnvoll zur Umsetzung eines 3D Konfigurators sind. Es werden auch einige wichtige Begriffe erläutert, die wichtig zur Realisierung von modernen Webanwendungen sind.

Kapitel 4 *Anforderungen und Konzeption* beschreibt die Entwicklung des Projektes. Nachdem das Problem noch einmal genauer analysiert wird und die Anforderungen an den Konfigurator klar sind, soll anschließend das Konzept mithilfe von Diagrammen und Mockups erläutert werden. Dann wird die Programmierung des Konfigurators gezeigt und wie es zur Endlösung kam. Dabei wird auf wesentliche Bestandteile und Funktionen der Anwendung genauer eingegangen. Der Leser soll verstehen warum es zu bestimmten Teillösungen kam und wie die Technologien Angular und WebGL dazu verwendet werden konnten.

Zuguterletzt beschäftigen sich die Kapitel 6 *Ergebnisse* und 7 *Zusammenfassung* mit den Evaluation der Anwendung. Zunächst werden einige Unit- und End-to-End-Tests durchgeführt. Danach wird das Ergebnis der Tests erläutert und infolgedessen schließlich ein Fazit gefasst.





# Kapitel 2

## Stand der Technik

Im Folgenden werden vergleichbare Arbeiten, Projekte und deren Einsatzgebiete angeführt. Es wird zunächst auf vergleichbare Projekte und anschließend auf die in der Arbeit verwendeten Technologien eingegangen. Dabei wird analysiert, warum eine Technologieanalyse im Kontext dieser Arbeit sinnvoll ist.

### 2.1 Bestehende Lösungsansätze

#### 2.1.1 Flyeralarm

Auf der Internetseite *Flyeralarm* gibt es nicht nur die Möglichkeit Flyer zu drucken. Es gibt sehr viele Kategorien mit einigen Produkten. Eine der Kategorien sind Becher. Man hat auch die Möglichkeit Mehrwegbecher zu bedrucken (siehe Abbildung 2.1). Dazu muss

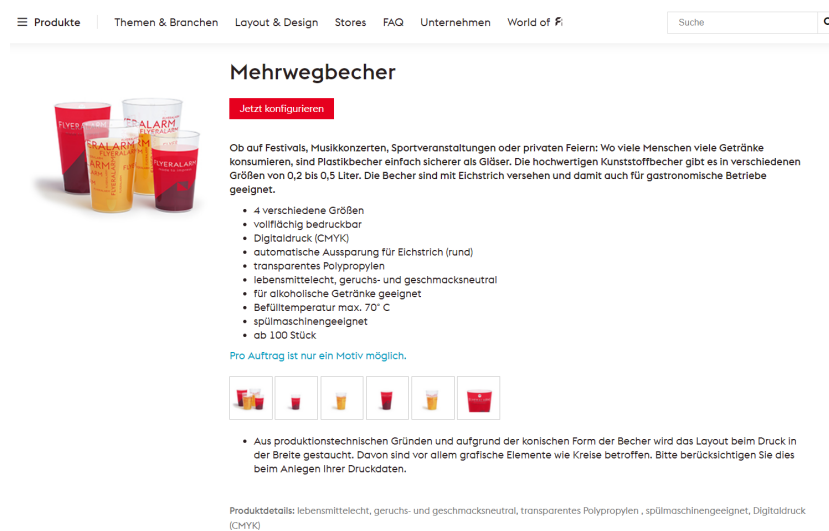


Abbildung 2.1: Bestellvorgang von Mehrwegbechern auf flyeralarm.com

## 2. STAND DER TECHNIK

man, nachdem man die gewünschte Größe gewählt hat, sein Design hochladen. Dabei sollte man das Datenblatt<sup>1</sup> berücksichtigen, welches die Vorgaben beschreibt wie zum Beispiel Sicherheitsabstand oder Druck Farbraum. Die Online-Druckerei schreibt auf ihrer Webseite: „Aus produktionstechnischen Gründen und aufgrund der konischen Form der Becher wird das Layout beim Druck in der Breite gestaucht. Davon sind vor allem grafische Elemente wie Kreise betroffen.“ [Fly] Eine Vorschau, wie das ganze am Ende aussehen wird, gibt es nicht. Im schlechtesten Fall hat man am Ende ein nicht zufriedenstellendes Ergebnis. Auch andere Seiten bieten ein ähnliches Angebot. Eine fertige Vorschau ist jedoch eher nicht zu finden.

### 2.1.2 Spread Shirt

Spreadshirt ist eigentlich eine Onlinedruckerei für T-Shirts. Sie selbst schreiben über sich Folgendes: „Seit 2002 liefert Dir Spreadshirt T-Shirt-Druck in bester Qualität. Was als Start-up-Idee in Leipzig begann, ist inzwischen ein weltweit erfolgreiches Print-on-Demand-Unternehmen, das Wert auf faire Handels- und Produktionswege legt, seine Verantwortung als internationaler Arbeitgeber ernst nimmt und seinen Mitarbeitern ein attraktives Arbeitsumfeld bietet.“ [spr] Zwar wird kein Druck von Mehrwegbechern angeboten, dafür aber der Druck

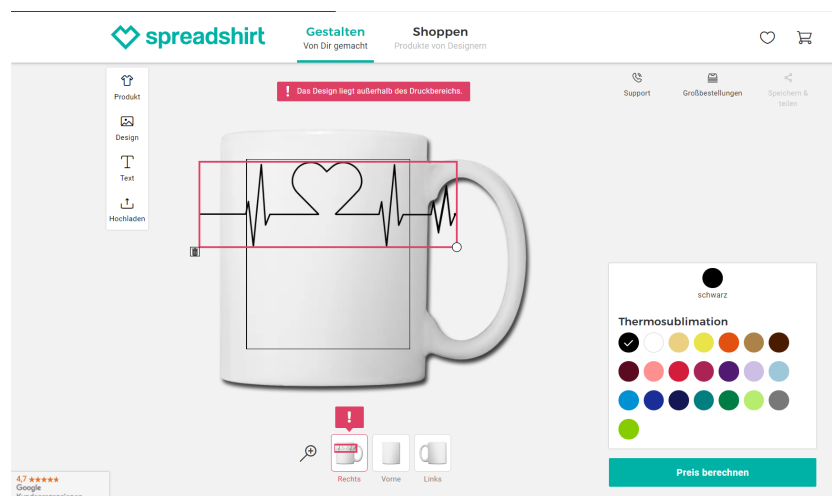


Abbildung 2.2: Gestaltung einer Tasse auf spread-shirt.com

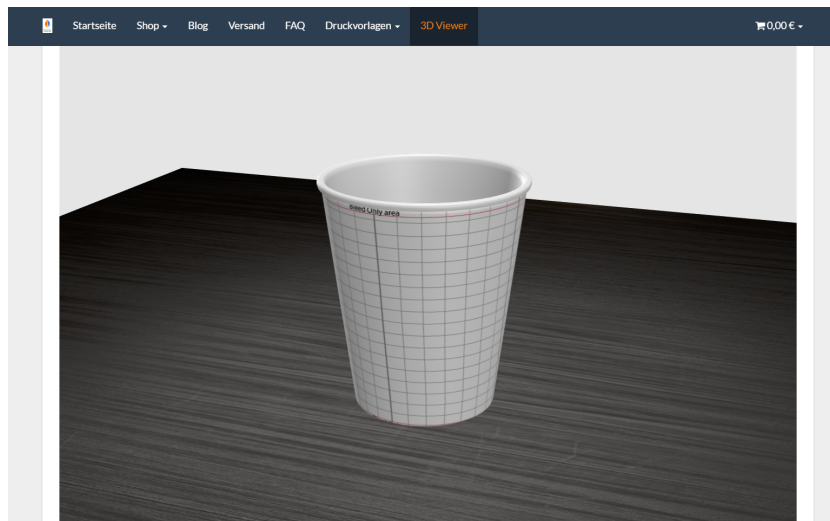
von Tassen. Der Kunde kann hier in einem Online-Konfigurator seine Tasse selbst gestalten, indem er Text oder Bildelemente auf die Tasse legt. Das ganze wird sogar in 3D angezeigt. Jedoch ist die Ansicht nicht flexibel sondern statisch. Man kann die Tasse lediglich von drei verschiedenen Blickwinkeln betrachten (rechts, vorne, links). In unserem Fall sollen fertige Designs dargestellt werden. Auch das ist beim Tassenkonfigurator von Spreadshirt schwierig. Wie in der Abbildung 2.2 zu sehen kann ein Objekt nur im Druckbereich dargestellt werden, nicht außerhalb des Bereichs. Damit ist ein Rundum-Druck nicht möglich.

<sup>1</sup>Das Datenblatt ist auf der Webseite zu finden und kann von dort heruntergeladen werden.

Trotzdem lässt sich sagen, dass dieser Konfigurator gut und übersichtlich gestaltet ist. Jedoch hat er nicht die Funktionalität, welche der Konfigurator für Gizeh haben sollte.

### 2.1.3 Becher-bedrucken.de

Ein weiteres Angebot für Becher gibt es auf *becher-bedrucken.de*. Dort gibt es einen 3D Konfigurator für verschiedene Becher. Der technische Ansatz ist schon sehr gut und kann bei der Entwicklung berücksichtigt werden. Es werden ähnliche Technologien verwendet, wie in dieser Arbeit. Jedoch sind die Designvorgaben ganz anders als die Druckvorgaben von Gizeh. Das hochgeladene Design ist so angepasst, das es bestmöglich auf dem Becher angezeigt werden kann. Die Darstellung in 3D ist schön anzusehen. Zumindest auf einem



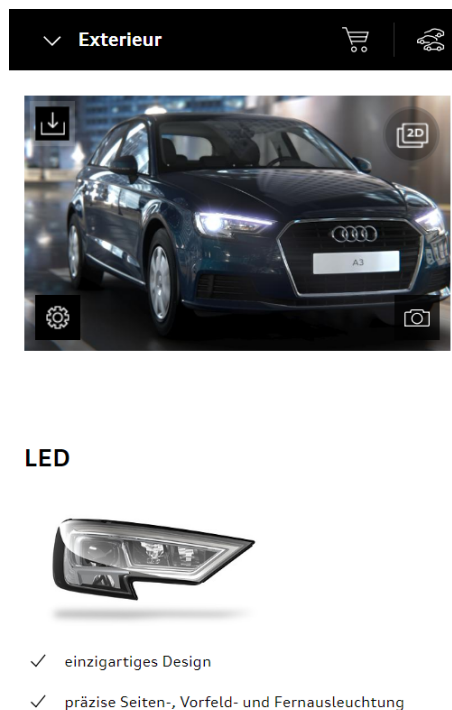
**Abbildung 2.3:** 3D Viewer eines Bechers auf *bedrucken-becher.de*

Gerät mit größerem Display. Eine Anpassung für mobile Geräte ist so gut wie gar nicht vorhanden. Technisch gesehen kann aber trotzdem an diesen Lösungsansatz angeknüpft werden.

## 2.2 3D Online Konfiguratoren

Heutzutage kann nahezu alles bedruckt werden. Die Vielzahl an Produkten ist groß. Dies haben wir bisher in diesem Kapitel erläutert, wie das konkret bei Bechern oder Tassen aussehen kann. Schwieriger zu finden sind allerdings 3D Konfiguratoren für diese Produkte. Oft gibt es höchstens eine 3D Pop-Up Ansicht, eine alte Lösung mit Flash o. ä. Eine responsive Lösung ist eher nicht zu finden.

In der Autoindustrie und anderen Branchen sind jedoch einige gute 3D Konfiguratoren umgesetzt. Wenn man eventuell die passenden Felgen sucht, bekommt man da einen übersichtlich gut gestalteten Konfigurator in 3D. Teilweise sind Konfiguratoren zu finden, welche responsiv sind. Einige Firmen bieten sogar an 3D Konfiguratoren für bestimmte Produkte umzusetzen.



**Abbildung 2.4:** *Neue 3D Ansicht von Audi*

Dabei handelt es sich meist um individuelle Lösungen. Das Folgende Beispiel soll zeigen, dass es durchaus möglich ist gute 3D Konfiguratoren zu entwickeln.

**Audi 3D Konfigurator** Letztes Jahr veröffentlichte Audi seinen neuen Konfigurator auf der Webseite. Er rendert die Fahrzeuge in Echtzeit. Die Anwendung ist auch für mobile Geräte optimiert (siehe Abbildung 2.4). So kann man den Konfigurator beispielsweise mit Toucheingaben steuern. Man kann sich das Fahrzeug ganz genau anschauen und an Details heranzoomen. Technisch eine sehr gute Umsetzung, die auch optisch etwas her macht. Für die Zukunft plant Audi sogar eine 4k-Darstellung. Dadurch bekommt der Benutzer eine noch realistischere Ansicht des Wagens.

**Siemens 3D Konfigurator** Immer mehr Firmen präsentieren neue 3D Konfiguratoren für ihre Produkte. So auch 2018 Siemens. Mit einem 3D Konfigurator ist es nun möglich Systemschaltschränke am 3D-Modell zu konfigurieren. Dabei wählt der Nutzer die verschiedenen Module, welche in den Schaltschrank verbaut werden sollen, in mehreren Schritten aus. Der Konfigurator prüft dabei, ob eine Kombination der Module möglich ist. Sogar eine Exportfunktion für CAD-Programme ist in den Konfigurator integriert. Die Kosten der Zusammenstellung werden auch in einer Übersicht dargestellt.

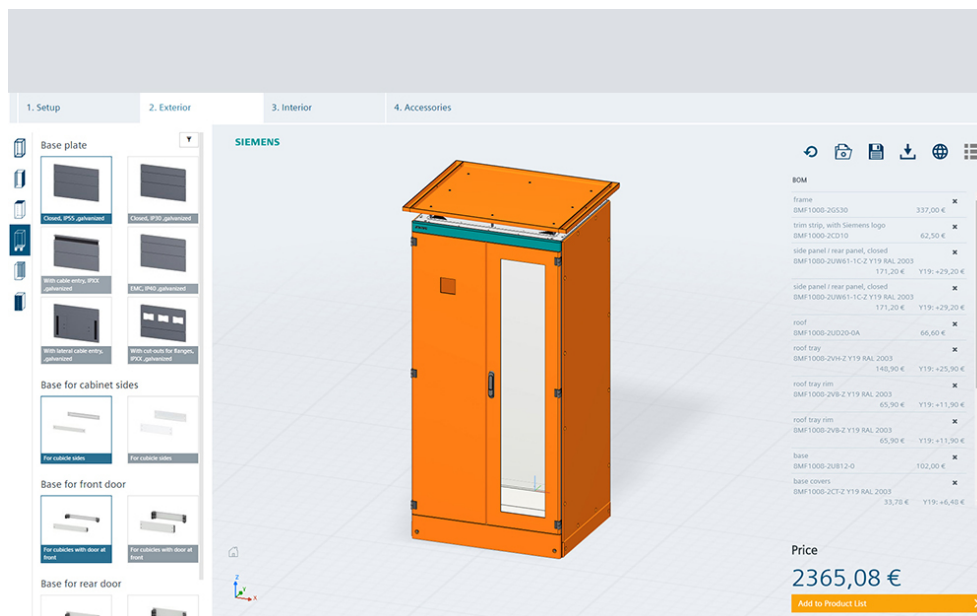


Abbildung 2.5: Neuer 3D Konfigurator von Siemens für Systemschaltschränke

**Trilux Limba 3D Konfigurator** TRILUX Simplify Your Light steht für den einfachsten und sichersten Weg zu einer maßgeschneiderten, energieeffizienten und zukunftsfähigen Lichtlösung. Das Unternehmen hat auf seiner Webseite zu verschiedenen Produkten einen Konfigurator. Es ist eine einfache und schnelle Hilfe bei der Suche nach der richtigen Beleuchtung. Dabei ist es möglich, das Produkt nach belieben anzupassen und zu konfigurieren. Am Ende lässt sich das ganze in einer PDF exportieren. Das hilft dem Kunden enorm, da er sich das Produkt besser vorstellen kann und es leichter fällt eine Entscheidung bezüglich der Auswahl zu treffen.

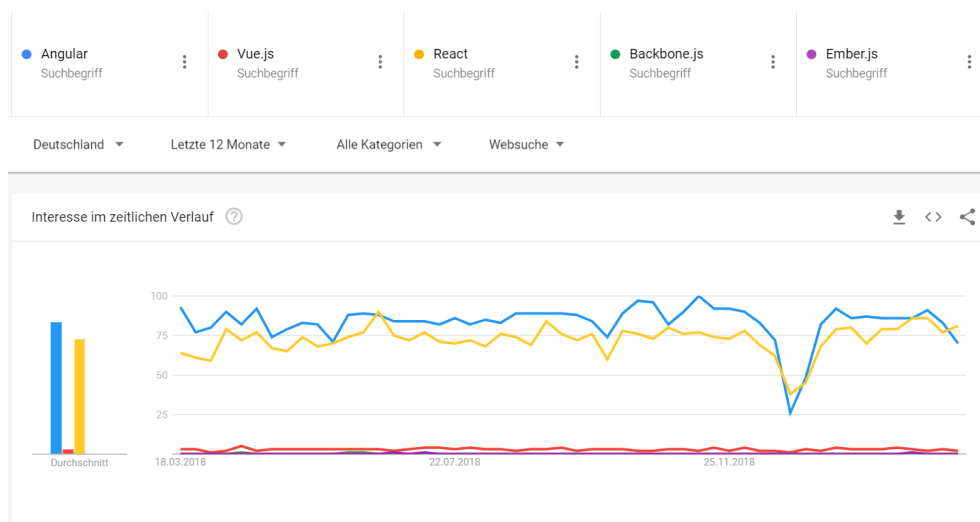
## 2.3 Webframeworks

Heutzutage werden Frameworks wie *Angular* oder *ReactJS* verwendet um performante und benutzerfreundliche Webanwendungen zu entwickeln. Es handelt sich dabei um JavaScript-Frontend-Frameworks. Man hat oft die Qual der Wahl, da in den letzten Jahren einige Frameworks entwickelt bzw. weiterentwickelt wurden. Die Anzahl der Angebote an JavaScript-Frameworks und -Libraries ist groß. Sie kommen oft beim Entwickeln von *Single Page Applications (SPA)*<sup>2</sup> zum Einsatz.

<sup>2</sup>Was Single Page Anwendungen sind wird im Kapitel Grundlagen genauer beschrieben.

## 2. STAND DER TECHNIK

**Angular** ist nichts anderes als ein JavaScript-Framework auf Basis von TypeScript<sup>3</sup>. Es wurde von Google entwickelt und ist ein Open-Source-Framework. Es unterstützt den Entwickler dabei, moderne Webanwendungen zu machen, die zum einen für Desktop und zum andern für Mobile optimiert worden sind. Wie in der Abbildung 2.6 zu sehen, ist *Angular* das meist genutzte und bekannteste Framework für SPAs. Ähnliches zeigt auch die Stack Overflow Entwicklerumfrage 2018: Bei den meist genutzten Bibliotheken und Frameworks liegt Angular mit 36,9% einen Platz vor React mit 27,8% [Sta]. Außerdem gibt es den Entwicklern



**Abbildung 2.6:** Google Trends Statistik. Suchanfragen von Frontend-Frameworks in 2018

Konventionen und Richtlinien an die Hand. Das ist gerade dann sinnvoll, wenn man in einem großen Team arbeitet. Oder bei großen Projekten für ein einfaches Handling und die Wartbarkeit des Codes sinnvoll wird. Ein Merkmal von Angular ist, dass es viele unterschiedliche Module gibt, die es ermöglichen, sehr effizient Anwendungen zu produzieren. Das Framework ist auch bekannt für seine tollen Animationen auf Basis von Web-Animation. Ein weiteres, wichtiges Modul ist das Routing-Modul. Im Grunde genommen sind Routings Grundbestandteil für *Single Page Anwendungen*. Mittels Routings kann ich festlegen welcher Teil der Anwendung nun angezeigt werden soll. Außerdem baut das Framework auf die komponentenbasierte Programmierung. In Kapitel 3 werden wir noch genauer auf Angular eingehen<sup>4</sup>.

**React** ist das meistgesuchte JavaScript Framework 2018 [Sta]. Es wurde 2013 von Facebook entwickelt und viele bekannte Unternehmen wie Netflix, Twitter oder PayPal verwenden das Framework. Ähnlich wie Angular setzt React auf modulare Komponentenarchitektur. Damit wird der Frontendcode leicht nachvollziehbar. „Das Ziel von React ist es, einfacheren Code schreiben zu können, dessen Bestandteile weniger miteinander verschränkt oder verwoben sind“ [Kö]

<sup>3</sup>Eine von Microsoft entwickelte Programmiersprache. TypeScript ist eine kompilierte und plattformübergreifende Sprache, die reine JavaScript-Dateien generiert.

<sup>4</sup>Die Dokumentation des Frameworks ist unter <https://angular.io/docs/> zu finden.

React ist kein Framework, es ist eine Bibliothek. Es ist ein sehr flexibles Werkzeug, dass in bestehende Anwendungen eingebaut werden kann, ohne den ganzen Code umzustrukturieren. Dem Entwickler wird keine Grundstruktur für seine Applikation gegeben, was ein wesentlicher Unterschied zu Angular ist. Ein weiteres Merkmal von React ist die virtuelle DOM. Diese garantiert die Synchronisierung der DOM indem bei Änderungen alles neu gerendert wird<sup>5</sup>.

**Vue.js** ist ein weiteres Frontend-Framework zur Entwicklung von *Single Page Anwendungen*. Es ist nicht das beliebteste Framework und trotzdem taucht es immer wieder auf (siehe Abbildung 2.6) und ist definitiv eine Alternative zu *Angular* oder *React*. Es setzt auch auf eine modulare Architektur, die in einzelne Komponenten zerteilt ist. Das Framework ist im Vergleich zu seinen Konkurrenten deutlich einfacher und hat eine flache Lernkurve. Dadurch ist es auch schneller und bringt dem Framework einen Vorteil gegenüber den Alternativen. Obwohl es kleiner ist, bringt es trotzdem alle wichtigen Funktionen wie Suchmaschinenoptimierung mit sich. Es kann auch in Kombination mit React verwendet werden. In Vue.js wird ohnehin das bekannte virtuelle DOM genutzt. Auch die Community ist stetig am wachsen.<sup>6</sup>

## 2.4 Flash und WebGL

In dem Artikel *HTML5/WebGL vs Flash in 3D Visualisation* schreibt der Autor folgendes über WebGL:

*„The development of improved 3D graphics in Web-based applications took a step forward recently, when programmers began building WebGL into the Mozilla Firefox nightly builds, and into WebKit, which is used by Google Chrome and Apple's Safari browser. WebGL is one of the most developed libraries which are supported by HTML5.“* [Bah].

Adobe Flash ist auch heute noch vielen ein Begriff. Es war darauf ausgerichtet interaktive 2D Grafiken im Web bereitzustellen. Mit der Veröffentlichung der Version 10 des Flash Players haben die Entwickler sogar eine z-Achse eingeführt. Sie ermöglicht eine 3D Darstellung und Transformation von Objekten. Diese Unterstützung für die Interaktion von Objekten der dritten Dimension wurde jedoch in begrenzter Weise bereitgestellt. Einer der aktuellsten Leistungstests zwischen dem Flash- und dem WebGL-Canvas-basierten 3D-Inhalt zeigt deutlich, dass der HTML5-Canvas beginnt, höhere Frameraten zu generieren und 3D-Inhalte im Web zu rendern. Die Testergebnisse zeigen, dass WebGL beim Rendern von 3D-Inhalten wesentlich schneller abschneidet und höhere Frameraten für die 3D-Animationen im Web bietet, während sie mit Flash verglichen werden. Weiter schreibt Senad Bahr in seinem Artikel: *„And actually, that is what the 3D graphics is all about, about altering the 3D model and see it change on time basis. This is something that can currently be achieved only with the usage of the HTML5 and WebGL engines on the web [...]“* [Bah] 3D-Modelle zu ändern und sie zu animieren kann also derzeit nur durch die Verwendung von HTML5

---

<sup>5</sup>Die vollständige Dokumentation ist unter <https://reactjs.org/docs/getting-started.html> zu finden

<sup>6</sup>Weitere Informationen sind in der offiziellen Dokumentation unter <https://vue.js.org/v2/guide> zu finden.

und WebGL im Web erreicht werden.

Weiter zeigen die Ergebnisse des Artikels deutlich, dass Flash-basierte 3D-Grafiken auf iOS-basierten Geräten nahezu nicht darstellbar sind, da Adobe mit seinem Flash-Plugin nie iOS unterstützt hat. Aufgrund der HTML5-Funktionalität und -Erreichbarkeit kann die Webseite auch auf einer Vielzahl von Geräten, einschließlich iOS- und Android-Geräten, wiedergegeben werden, ohne dass sich die Benutzer um das Vorhandensein der Plugins und die Versionierung des Players kümmern müssen. Webbrowser entwickeln sich stetig weiter. WebGL kann in über 95% aller Browser verwendet werden [Dev]. Was zuvor schon in Videospiele mit OpenGL möglich war, ist nun auch im Web möglich. Wie genau WebGL als Abstraktionsschicht für den grafischen Teil einer Anwendung verwendet werden kann, wird in Kapitel 2.5 erläutert.

Mit WebGL müssen die Benutzer nicht dazu aufgefordert werden, eines der Plugins zu installieren, um den 3D-Inhalt auf der Website zum Laufen zu bringen. Die einzige Voraussetzung ist, dass der Webbrowser das HTML5-Canvas-Element unterstützt, über das WebGL den Inhalt für den Benutzer darstellt.

Im Wesentlichen kann WebGL auf jeder Plattform und auf allen großen Systemen mit OpenGL-fähiger Grafikkarte und einem Browser ausgeführt werden, der WebGL unterstützt.

### 2.5 3D JavaScript Bibliotheken

Um eine 3D Szene mit WebGL darzustellen wird ein JavaScript Framework benötigt. Es erstellt und rendert eine Szene mit den 3D Objekten. Im Folgenden werden die zwei bekanntesten und weit verbreitetsten Bibliotheken vorgestellt.

**ThreeJS** *Three.js* ist es eine sehr seriöse WebGL-Bibliothek mit einer starken Community und vielen guten Beispielen. Es wird auch oft in kommerziellen Webanwendungen verwendet. Die erste Version des Frameworks tauchte 2010 auf und der Quellcode wird in einem Repository auf GitHub gehostet<sup>7</sup>. Three.js hat eine verständliche Struktur und große Anpassungsmöglichkeiten. Es ist eine Anwendungsprogrammierschnittstelle, mit der animierte 3D-Grafiken in einem Webbrowser erstellt und angezeigt werden. Neben der Dokumentation gibt es auch ein Wiki, was dem Entwickler zum schnellen Einstieg sehr helfen kann<sup>8</sup>. Schon beim erstellen einer einfachen Szene mit dem Framework fällt auf, dass es dem Prinzip der objektorientierten Programmierung folgt<sup>9</sup>. In dem Kapitel 1 wird noch genauer auf das Framework eingegangen.

**BabylonJS** *Babylon.js*<sup>10</sup> ist ein Framework, mit dem komplette 3D-Webanwendungen erstellen werden können. Babylon.js hat eine Community, die stetig wächst und auch aktiv zum Projekt beiträgt und immer mehr Funktionen hinzufügt. Das Framework hat alle notwendigen Werkzeuge, um 3D-Anwendungen umzusetzen. Sie können 3D-Objekte laden und zeichnen, diese 3D-Objekte verwalten, Spezialeffekte erstellen und verwalten, räumliche

---

<sup>7</sup><https://github.com/mrdoob/three.js/>

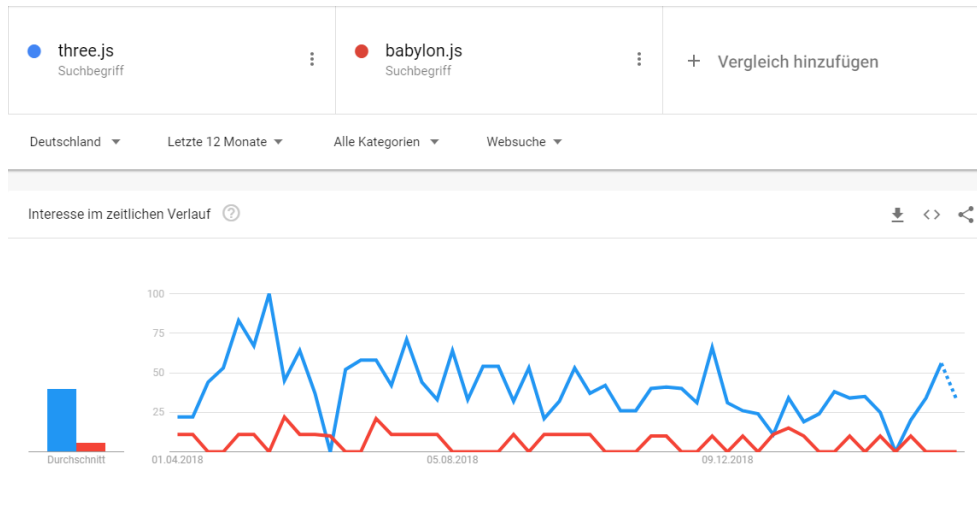
<sup>8</sup><https://github.com/mrdoob/Three.js/wiki/>

<sup>9</sup><https://threejs.org/docs/#manual/en/introduction/Creating-a-scene>

<sup>10</sup>Die offizielle Dokumentation ist unter <https://doc.babylonjs.com/> zu finden.



Sounds spielen und verwalten, Gameplay erstellen und vieles mehr. Babylons.js ist ein benutzerfreundliches



**Abbildung 2.7:** *Google Trends: Vergleich Three.js und Babylon.js in den letzten 12 Monaten*

Framework, da Sie diese Dinge mit den minimalen Codezeilen einrichten können. Es ist ein mit TypeScript entwickeltes JavaScript-Framework. (vgl. [MM]) Wie in Abbildung 2.7 zu sehen ist Babylon.js jedoch weniger gefragt als three.js. Das zeigt sich auch in den 3D Webanwendungen, wo auch meist three.js verwendet wird.

Dieses Kapitel hat gezeigt, dass im Bereich der 3D Konfiguratoren schon einiges möglich ist. Es zeigt, die in dieser Arbeit analysierten Technologien auch in Zukunft eine wichtige Rolle spielen. Fakt ist aber auch, dass eine optimale Lösung für einen responsiven 3D Konfigurator eher nicht zu finden ist. Gerade wenn man spezielle Produkte darstellen möchte, wie die Mehrwegbecher mit einem hochgeladenen Design, ist keine zufriedenstellende Lösung da.



# Kapitel 3

## Grundlagen

In dem Grundlagenkapitel geht es um das Basiswissen, auf dem die Arbeit aufbaut. Es wird näher auf die verwendeten Technologien eingegangen. Dabei werden die Frontend-Frameworks beleuchtet sowie die 3D Bibliotheken. Näher wird auch auf die Begriffe Responsive Webdesign, Usability und Performance eingegangen.

### 3.1 Single Page Anwendungen

Früher war es bei Webanwendungen wichtig, soviel wie möglich auf dem Server zu erledigen. Bei modernen Webframeworks gilt jedoch genau das Gegenteil. Es wird versucht möglichst viel clientseitig umzusetzen. *„Dies steigert die Benutzerfreundlichkeit und schafft die Möglichkeit der Anpassung an die Auflösungen und Formfaktoren der vielen unterschiedlichen klassischen und mobilen Plattformen.“* [Bah]. Im Folgenden wird nun kurz etwas genauer darauf eingegangen was André Krämer in seinem Videokurs sehr gut und einfach erklärt hat.

**Klassische Webanwendungen** Bevor man sich damit befasst was Single Page Anwendungen sind, sollte man sich das Prinzip einer klassischen Webanwendung anschauen. Nehmen wir beispielsweise an, ein Client sendet eine Anfrage an einen Server. Hierbei handelt es sich um einen Webbrowser, welcher eine Webadresse öffnen möchte. Der Zielsever übernimmt die Verarbeitung. In der Regel laufen nun Skripte auf dem Server, welche HTML rendern. Schließlich bekommt der Client auf seine Anfrage eine Antwort in Form eines HTML-Dokumentes und kann es dann anzeigen. Bei diesem Vorgang liegt die komplette Verarbeitung bei dem Server. Der Webbrowser dient lediglich der Darstellung und einfachen Eingabe. Das die Daten auf einem Server verarbeitet werden wird deutlich, wenn ein Seitenwechsel vorgenommen wird. Dabei wird kurze Zeit nichts angezeigt während die Seite komplett neu lädt.

Mit der Einführung von AJAX<sup>1</sup> in 2005 machte die Webentwicklung einen Schritt nach vorne. Nun war es nämlich möglich, Daten asynchron an den Server zu senden. Das heißt,

---

<sup>1</sup>Ajax bzw. AJAX steht als Akronym für „Asynchronous JavaScript and XML“. Die Technologie ermöglicht es, einzelne Teile einer Webseite bei Bedarf asynchron zu laden, so dass sie dynamisch wird. Der angezeigte Inhalt lässt sich gezielt so manipulieren, ohne die komplette Seite neu zu laden.

der Server verarbeitet zwar immer noch die Daten, sendet nun aber nur noch einen Teil des vorgerenderten Dokuments oder sogar nur einzelne Daten der Seite. Somit wird die Seite nicht wieder komplett neu geladen. Wenn man nun aber in einen komplett andern Bereich der Seite wechseln will, hilft AJAX nicht. Die Seite wird neu geladen und es dauert einen kurzen Moment bis das Ergebnis vom Webbrowser angezeigt werden kann. Um dieses Problem zu lösen, kommen *Single Page Anwendungen* zum Einsatz.

**Dezentralisierung der Webanwendung** Im Grunde arbeiten SPA's nach folgendem Prinzip: Der Webbrowser sendet eine Anfrage an den Server. Dieser verarbeitet, wie bei klassischen Webanwendungen nun die Anfrage. Anschließend wird im Webbrowser eine Benutzeroberfläche dargestellt. Nun kommt der entscheidende Unterschied zu klassischen Anwendungen. Bei einem Seitenwechsel oder anderen Interaktionen des Benutzers wird eine neue Benutzeroberfläche erzeugt, indem Teile der Seite ausgetauscht werden. Die Seite wird nie komplett neu geladen. Der Nutzer bleibt immer auf der einen Seite, zumindest fühlt es sich für den Benutzer so an. Dieses Prinzip kennt der Anwender von Desktop-Anwendungen. Man hat sogar die Möglichkeit die Anwendung (zumindest teilweise) offlinefähig zu machen. Die Grundlage einer solchen Anwendung ist das Routing. Abhängig vom Pfad werden bestimmte Bausteine der Anwendung angezeigt bzw. ausgeblendet. Wie wir im späteren Verlauf dieser Arbeit noch sehen werden, hat man tatsächlich nur eine HTML-Datei, welche abhängig vom Kontext immer andere Inhalte anzeigen kann. Zusammenfassend lässt sich also festhalten, das ein Ziel von SPAs ist, die Kommunikation zwischen Client und Server enorm zu reduzieren. Um dies zu erreichen wird die Anwendung dezentralisiert. Das heißt, es gibt nur ein HTML Dokument[Dom]. Durch den Einsatz von Frameworks, wie Angular, ist es möglich Inhalte zu aktualisieren oder zu einer anderen Seite zu wechseln, ohne die Seite über den Server neu zu laden. <sup>2</sup>

## 3.2 Framework Angular

Das Framework Angular ist sehr umfangreich. Im Folgenden werden einige Grundlagen erläutert, die zur Implementierung einer Anwendung mit Angular benötigt werden. Dabei soll ein Grundverständnis vermittelt werden, wie das Framework aufgebaut ist und wie es arbeitet.

### 3.2.1 Komponentenbasierte Programmierung

Angular ist ein Framework, das auf Komponenten setzt. Das Prinzip der komponentenbasierten Entwicklung kommt nicht nur bei Angular vor, sondern auch bei vielen anderen Programmiersprachen und Frameworks. Was in Angular Komponenten sind wird in dem Buch *Angular: Grundlagen, fortgeschrittene Techniken und Best Practices mit TypeScript - ab Angular 4, inklusive NativeScript und Redux* folgendermaßen beschrieben:

---

<sup>2</sup>Das Prinzip von Single Page Anwendungen wird zum Beispiel von Googles Gmail und Gmaps sowie von Twitter verwendet

*„Komponenten sind die Grundbausteine einer Angular Anwendung. Jede Anwendung ist aus vielen verschiedenen Komponenten zusammengesetzt, die jeweils eine bestimmte Aufgabe erfüllen. Eine Komponente beschreibt somit immer einen kleinen Teil der Anwendung, z. B. eine Seite oder ein einzelnes UI-Element.“*[\[WMKH\]](#)

Im Grunde ist eine Komponente also nichts anderes als ein selbstdefinierter HTML-Knoten, den man im HTML Kontext ganz normal nutzen kann. Man erzeugt einfach DOM-Elemente<sup>3</sup> und kann sie dann automatisch in Angular nutzen. Eine Komponente in Angular ist in drei Parts aufgeteilt: Logik, Vorlage und Style. Die Logik beschreibt die TypeScript Klassen, Eigenschaften, Methoden und so weiter. Sie beschreibt also, was zu tun ist, wenn zum Beispiel ein bestimmter Button geklickt wird. Die Vorlage (Template) ist ein HTML-Schnipsel. Dies kann einfach nur ein Button-Element sein oder auch eine HTML-Struktur mit mehreren HTML-Elementen. Sie beschreibt die Struktur bzw. den Aufbau der Komponente. Wie die Komponente aussehen soll wird in dem Style Part festgelegt. Dabei ist zu beachten, dass die Style-Definitionen nur innerhalb der Komponente gelten. Globale Styles der Anwendung werden separat definiert.

Der Startpunkt der Anwendung ist die `index.html`. Sie definiert die Applikationskomponente, der Ankerpunkt einer Angular-Anwendung, quasi eine Basiskomponente. In Dieser können nun weitere Kindskomponenten eingefügt werden, welche auch ineinander verschachtelt sein können. So entsteht eine komplette Struktur. Eine Komponentenvorlage kann also nicht nur HTML-Elemente enthalten, sondern auch Kindskomponenten. Entscheidend ist, dass Komponenten verschachtelt werden können. Dies ist das Grundprinzip der komponentenbasierten Programmierung (vgl. Angular Grundkurs[\[Ün\]](#)).

### 3.2.2 Modulare Umsetzung

Schauen wir uns nun an, wie Angular mit Modulen arbeitet und was genau Module sind. *Nikolas Poniros* hat das in seinem Buch *Angular für Dummies* so definiert:

*„Angular-Module helfen beim Gliedern einer Webanwendung in verschiedene Funktionsblöcke. Alle Angular-Bausteine, die logisch zu einem Funktionsblock gehören, werden mit dem entsprechenden Angular-Modul registriert. Die registrierten Bausteine gehören dann zum Angular-Modul. Angular-Module sind von der Denkweise her vergleichbar mit ECMA-Script-Modulen. ECMA-Script-Module kapseln TypeScript-Konstrukte wie Klassen und Funktionen und erlauben nur den Zugriff auf exportierte Konstrukte. Angular-Module kapseln Komponenten, Pipes und Direktiven<sup>4</sup>. Ein anderes Angular-Modul kann nur auf einen Baustein zugreifen, wenn es dieses exportiert.“*[\[Pon\]](#)

Angular bietet also die Möglichkeit modular zu arbeiten. Die einzelnen Elemente und Funktionalitäten können in Modulen gruppiert werden. Im Grunde ist ein Modul nichts

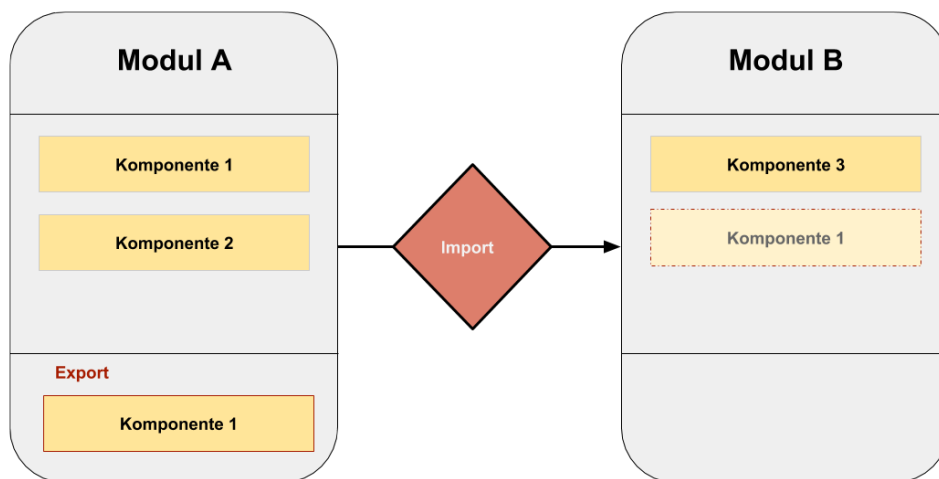
---

<sup>3</sup>Wenn eine Webseite geladen wird, erstellt der Browser ein Document Object Model der Seite. Das HTML-DOM-Modell ist als Baum von Objekten aufgebaut. Diese Objekte sind in der Regel einfache HTML-Tags

<sup>4</sup>Was genau es mit diesen Angular Bausteinen auf sich hat wird in Punkt 3.2.3 beleuchtet.

anderes als ein Container. Sie lassen sich besonders gut verwenden, wenn man mit mehreren Entwicklern zusammenarbeiten möchte. Module sind einzelne Bestandteile der Anwendung und lassen sich ganz einfach in bestehende Anwendungen einbauen. So kann ich ein Modul in mehreren Anwendungen wiederverwenden. Oder verschiedene Teammitglieder entwickeln jeweils ihre eigenen Module, welche anschließend in einer Anwendung zusammengefügt werden können. Ich kann ein Modul also auch in einem anderen Kontext wiederverwenden. Ein Modul darf allerdings nur in einem einzigen Modul initialisiert werden. Stattdessen wird das Modul bei mehrfacher Verwendung lediglich importiert. Folgendes Beispiel soll das Prinzip modularer Umsetzung verdeutlichen.

Das hier beschriebene Beispiel ist in Abbildung 3.1 zu sehen. Angenommen wir haben ein



**Abbildung 3.1:** Fallbeispiel mit zwei Angular Modulen mit Komponenten

*Modul A.* In diesem Modul sind nun *Komponente 1* und *Komponente 2* deklariert. Das heißt, es ist möglich in *Komponente 1* die *Komponente 2* zu verwenden und umgekehrt, weil beide in dem *Modul A* registriert sind. Jetzt haben wir ein weiteres *Modul B*. In diesem ist die *Komponente 3* registriert. Nun wäre es sehr praktisch, wenn *Komponente 3* ebenfalls *Komponente 1* aus dem anderen Modul verwenden könnte. Wie schon erwähnt darf ein Baustein in Angular nur in einem Modul registriert bzw. initialisiert werden. Deshalb muss das *Modul A* in *Modul B* importiert werden, damit auch dort die *Komponente 1* verwendet werden kann. Zusätzlich muss in *Modul A* die *Komponente 1* exportiert werden. Dadurch wird sie erst von außen frei zugänglich und somit in Bausteinen anderer Module nutzbar.

**Hauseigene Module** Angular hat auch ein paar hauseigene Module, die je nach Anwendungsfall importiert werden können. Das *BrowserModule* wird für Web-Anwendungen im Angular-Umfeld genutzt und verfügt über alle Funktionalitäten, mit dem wir in der Lage sind, Ereignisse innerhalb des Browsers abzufangen, DOM-Rendering durchführen zu können und

damit schließlich die lauffähige Angular-Anwendung im Browser realisieren zu können. Das *CommonModule* beinhaltet allgemeine Funktionen, die sehr, sehr häufig genutzt werden. Diese Funktionen liegen in Form von Direktiven und Pipes vor, womit ich bestimmen kann, ob ein HTML-Knoten angezeigt wird oder nicht oder auch Pipes, womit ich Ausgaben formatieren kann. Auch sprachabhängige Funktionalitäten sind in den *CommonModule* enthalten. Das *HttpModule* ist dafür da, Client-Server-Kommunikation zu betreiben. Das heißt, HTTP-Requests lassen sich mit Hilfe des *HttpModules* hervorragend realisieren, dafür werden Services zur Verfügung gestellt. Das *HttpModule* hat auch Services für Testings und Co. Für Formulare gibt es entweder das *FormsModule* oder das *ReactiveFormsModule*. Das hängt so ein wenig davon ab, wie man Formulare in der Anwendung gestalten will. Das *RouterModule* ist dafür da, um Komponenten-Routing zu realisieren. Das Routing ist die Grundlage für *Single-Page-Applications*. Das heißt, über das Routing kann man bestimmen, welche Komponente dargestellt werden muss, wenn ein bestimmter Pfad in der Anwendung besucht wird.

### 3.2.3 Grundfunktionen des Frameworks

Module und Komponenten in Angular haben wir nun schon etwas ausführlicher betrachtet. Nun wollen wir uns weitere Funktionen und Bestandteile von Angular anschauen<sup>5</sup>. Wie die einzelnen Bausteine umgesetzt und implementiert werden können sehen wir noch in Kapitel 4 Methodik am Fallbeispiel des 3D Konfigurators für Mehrwegbecher.

**Bindungen** „Bindungen sind im Wesentlichen die Brücke zwischen der Darstellungsschicht und der Logikschicht“ [Ün]. Man kann beispielsweise Variablen oder Eigenschaften aus TypeScript-Klassen in Vorlagen binden. Angenommen in der TypeScript-Klasse einer Komponente ist die Variable `title` deklariert. Um `title` nun in der Vorlage zu binden, schreibt man Folgendes: `<h1> {{title}} </h1>`. Natürlich geht das ganze noch deutlich komplexer. Dadurch wird die Komponente mit ihrem Content dynamisch.

**Direktiven** Direktiven sind ein wichtiger Bestandteil von Angular. Direktiven werden in Vorlagen genutzt, indem ich sie als Attribute auszeichne. Das heißt, Direktiven sind oft Attribute, die in ein HTML-Element hinzugefügt werden oder auch beispielsweise dadurch deklariert, dass an ein bestimmtes HTML-Element ein bestimmtes Attribut angehängt sein muss. Attribut-Direktiven machen eigentlich nichts anderes als eine Anpassung des Aussehens beziehungsweise eine Anpassung des Verhaltens eines Elements, das heißt, es manipuliert ein vorhandenes Element im Wesentlichen.

Analog dazu kann man auch strukturelle Direktiven benutzen. Das gibt es zum Beispiel `ngFor`, diese ist sehr praktisch bei der Entwicklung einer Angular-Anwendung. Sie benutzt nämlich das Element, auf das sie angewendet wurde, als Vorlage, iteriert durch eine Liste, zum Beispiel ein `<ul>`-Element, und packt diese Vorlage so oft in den DOM hinein, wie es Elemente in der Liste gibt.

---

<sup>5</sup>Auf die weiteren Bestandteile von Angular wird hier nicht im Details eingegangen. Genauere Dokumentationen dazu sind online unter [angular.io](http://angular.io) zu finden.

**Pipes** Angular bietet uns die Möglichkeit zur Nutzung von Pipes. Sie dienen dazu, eine Ausgabe zu manipulieren. Überwiegend werden Pipes in Vorlagen genutzt. Die Pipe manipuliert die Ausgabe und sorgt dafür, dass beispielsweise der Name in Großbuchstaben (uppercase) dargestellt wird. Analog lassen sich auch Pipes in Kette schalten. Das heißt, wenn man eine Ausgabe hat, kann man diese wiederum in andere Pipes hineinpacken.

**Services** Der Begriff Services in der Entwicklung ist breit gefächert. Jede Programmiersprache versteht unter Services in gewissen Maßen etwas anderes. In der Angular-Welt sind Services Logiken, die nicht abhängig von der View sind. Eine Komponente besteht aus Logik- und Darstellungsschicht. In der Logikschicht ist ganz explizit eine Logik vorhanden, die nur für diese eine Komponente da ist.

Ein Service selber hat auch Logiken, die aber nicht allein für eine Komponente gelten, sondern auch im Kontext in anderen Bausteine genutzt werden können. Ein einfaches Beispiel dafür: Client-Server-Kommunikation. Es ist möglich einen Service zu erstellen, der das gesamte Handling des Logins steuert. Dieser Service verarbeitet dann via Passwort und Username die Client-Server-Kommunikation für das Login und empfängt vom Server dann das User-Objekt. An der anderen Stelle kann es sein, dass ich zum Beispiel die Authentifizierung überprüfen möchte oder einfach nur den Namen des Users brauche. In diesem Fall würde man den gleichen Service in der anderen Komponente wieder nutzen und könnte dann entsprechend auf die Eigenschaften des Services zurückgreifen, die es zuvor schon geholt hat.

#### 3.2.4 Angular CLI

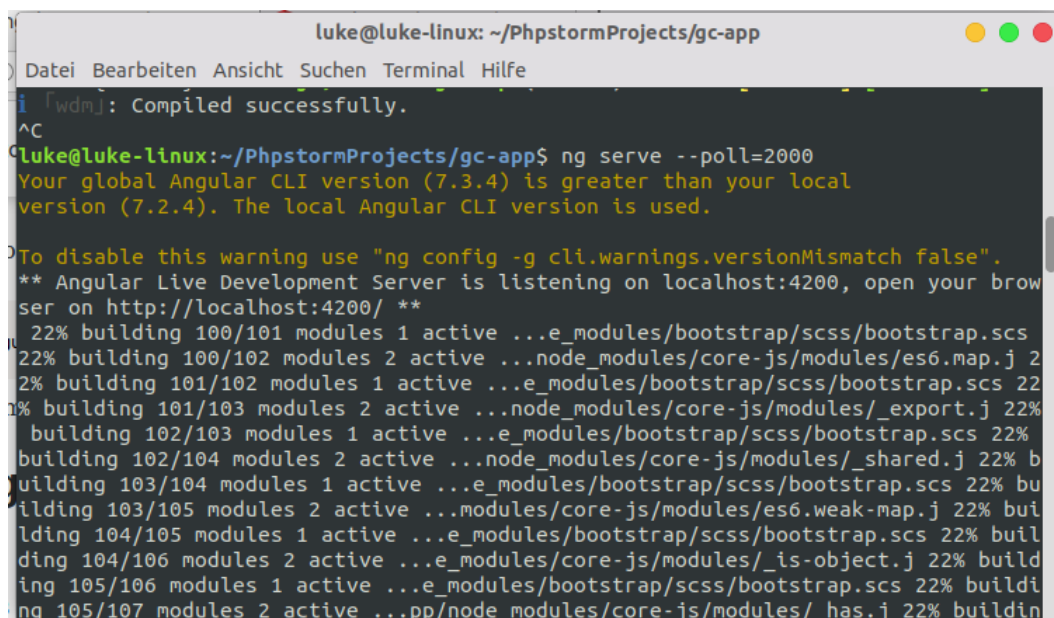
Die Angular CLI ist ein mächtiges Tool. Wie der Name schon sagt wird es über die Command Line gesteuert. Sie kann beispielsweise verwendet werden, um ein neues Angular-Projekt anzulegen. Die CLI legt dann im Hintergrund die benötigte Projektstruktur mit allen benötigten Dateien in dem gewünschten Verzeichnis ab. Anschließend ist die Anwendung sogar schon lauffähig. Man kann sie ganz bequem über die CLI starten, dabei wird standardmäßig der Developing-Server verwendet. Mit Webpack wird das ganze Projekt gebündelt. Eine sehr große Hilfe können die Code Generatoren sein. Sie erzeugen automatisch Komponenten, Module oder Services. Dabei werden automatisch alle `@import`-Anweisungen, Exports und so weiter vorgenommen. Wie das ganze konkret aussieht schauen wir uns bei der Umsetzung des Konfigurators noch einmal an. Angular bietet eine sehr gut verwendbare Testumgebung. Mit verschiedenen Modulen kann die Anwendung bis ins kleinste Detail getestet werden. Ganze User-Interaktionen können simuliert werden, auch auf verschiedenen Geräte-Klassifikationen. Wenn man die Anwendung veröffentlichen will, bietet das Framework über seine CLI eine build Funktion. Diese ermöglicht eine einfache und kompakte Lösung. Alles in allem ist Angular CLI ein sehr komfortables Tool, was einem das Erstellen einer Angular-Anwendung leichter macht.

#### 3.2.5 Versionen

Das Framework Angular setzt auf das System der semantischen Versionierung (*SEMVER*). Die erste Version des Frameworks war *AngularJS*. Schon die erste Version hatte das Ziel,



ein strukturiertes und übersichtliches Framework zu sein. Mit der *Version 2* wechselte die Programmiersprache von JavaScript zu TypeScript, welches von Microsoft entwickelt wurde. Das Framework wurde mit der Version 2 also komplett neu entwickelt. Es setzt aber größtenteils auf das alte Konzept von *AngularJS*. Da das Router-Modul schon die Version 3 hatte, wurde die nächste Version von Angular komplett auf *Version 4* angehoben, damit nun wieder alle Module auf der selben Version sind [Bö]. Mittlerweile ist die aktuelle Version des Framework *Angular 7*<sup>6</sup>. Die aktuelle Version bringt ein paar Bugfixes mit sich sowie mehr Flexibilität. So können beispielsweise über die *Angular CLI* ganz bequem alle Pakete automatisch aktualisiert werden (siehe [Ste]). Da die Anwendung auf verschiedenen Systemen entwickelt wird, kann es durchaus vorkommen, dass unterschiedliche Versionen von *Angular CLI* installiert sind. Beim starten der Anwendung erscheint eine Warnung, dass die Version des Projekts eine andere ist als die auf dem Rechner global installierte (wie in Abbildung zu sehen). Normalerweise kann



```

luke@luke-linux: ~/PhpstormProjects/gc-app
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
i [wdm]: Compiled successfully.
^C
luke@luke-linux:~/PhpstormProjects/gc-app$ ng serve --poll=2000
Your global Angular CLI version (7.3.4) is greater than your local
version (7.2.4). The local Angular CLI version is used.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".
** Angular Live Development Server is listening on localhost:4200, open your brow
ser on http://localhost:4200/ **
22% building 100/101 modules 1 active ...e_modules/bootstrap/scss/bootstrap.scs
22% building 100/102 modules 2 active ...node_modules/core-js/modules/es6.map.j 2
2% building 101/102 modules 1 active ...e_modules/bootstrap/scss/bootstrap.scs 22
1% building 101/103 modules 2 active ...node_modules/core-js/modules/_export.j 22%
building 102/103 modules 1 active ...e_modules/bootstrap/scss/bootstrap.scs 22%
building 102/104 modules 2 active ...node_modules/core-js/modules/_shared.j 22% b
uilding 103/104 modules 1 active ...e_modules/bootstrap/scss/bootstrap.scs 22% bui
lding 103/105 modules 2 active ...modules/core-js/modules/es6.weak-map.j 22% bui
lding 104/105 modules 1 active ...e_modules/bootstrap/scss/bootstrap.scs 22% buil
ding 104/106 modules 2 active ...e_modules/core-js/modules/_is-object.j 22% buildi
ng 105/106 modules 1 active ...e_modules/bootstrap/scss/bootstrap.scs 22% buildi
ng 105/107 modules 2 active ...pp/node modules/core-js/modules/ has.j 22% buildin

```

**Abbildung 3.2:** Warnung wegen Versionskonflikt der Angular CLI

man diese Meldung ignorieren und sogar ausblenden lassen. Es kann aber unter Umständen sinnvoll sein, wenn etwa eine neue Version neue Features mit sich bringt oder brauchbare Änderungen. Mit dem Befehl `ng update` ist es innerhalb der Version 7 sehr leicht ein Update vorzunehmen. Nach Ausführen des Befehls sollte die Anwendung mit der aktuellen Version lauffähig sein.

### 3.3 Three.js

Schon 2010 begann Ricardo Cabello Miguel die 3D-Bibliothek Three.js zu entwickeln. Mittlerweile ist er unter dem Namen Mr.doob bekannt. Als dann WebGL veröffentlicht wurde, portierte

<sup>6</sup>Stand März 2019

er *Three.js*, um es mit der neuen Technologie zu verwenden. Mr.doob bezeichnete es als „einfach zu implementieren“. Er hatte damals bereits zwei andere Renderer damit gebaut. Seit dieser Zeit hat *Three.js* an Leistung und Raffinesse zugenommen. Nun ist es zur beliebtesten Wahl für die Erstellung von 3D-Anwendungen mit WebGL geworden.

*Three.js* abstrahiert den Kontext von WebGL und stellt die 3D-Szene als Meshes, Materials und Lightings dar <sup>7</sup>. Aber *Three.js* ist mehr als nur eine Abstraktionsschicht für WebGL. Die *Three.js*-API wurde einfach konzipiert, sodass ein schneller Einstieg möglich ist. Funktionen können ziemlich einfach hinzugefügt und *Three.js* angepasst werden. Vordefinierte Objekte und Beispiele ermöglichen die Entwicklung von Modellierungsanwendungen bis hin zu Spielen. Auch Animationen sind einfach zu implementieren sowie Interaktionen in der Szene. Neben den Grundfunktionen der Bibliothek gibt es zahlreiche Beispiele und Extras, welche in eigene Projekte eingebunden werden können. Außerdem wurde *Three.js* so umgesetzt, dass eine hohe Leistung ohne Beeinträchtigung der Usability gewährleistet ist. Zudem gibt es umfangreiche Fehlerprüfungen, Ausnahmen und Konsolenwarnungen, um den Entwickler zu unterstützen.

#### 3.3.1 Die Szene

Die 3D-Bibliothek ist objektorientiert. Das heißt, Programmierer arbeiten mit JavaScript-Objekten, anstatt nur JavaScript-Funktionsaufrufe durchzuführen. Folgendes einfaches Beispiel, zeigt wie eine Szene in *Three.js* erstellt werden kann.

Listing 3.1: Erstellung einer Szene in *Three.js*

```
|| var scene = new THREE.Scene();
```

Über verschiedene Funktionen können dann die weiteren Bestandteile einer Szene hinzugefügt werden und anschließend mit dem Renderer gerendert werden. Da Angular auf TypeScript basiert, muss man bei der Implementierung einer Szene in Angular ein paar Dinge beachten. Diese werden in Kapitel 4 genauer erläutert. In diesem Teil werden zunächst die wesentlichen Bestandteile der *Three.js* Szene vorgestellt. Differenzierte Erklärungen zu *Three.js* sind in der offiziellen Dokumentation zu finden <sup>8</sup>.

**Kamera** Es gibt verschiedene Kamerateypen. Normalerweise wird immer die *PerspectiveCamera* verwendet. Deshalb wird in dieser Arbeit nicht weiter auf die anderen Typen eingegangen. Die Kamera wird auch als Objekt in der Szene angelegt. Dabei müssen verschiedene Eigenschaften wie Seitenverhältnis und Position der Kamera angegeben werden.

**Renderer** Damit eine Szene gerendert werden kann, wird der Renderer benötigt. Hierbei handelt es sich schlichtweg um einen WebGL-Renderer. Der wird dann in ein `<canvas>`-Element eingefügt um die Szene darin darzustellen.

---

<sup>7</sup>Das sind typische Bestandteile einer 3D-Szene die Grafik Programmierer kennen

<sup>8</sup><http://threejs.org/docs>

**Lighting** Damit Objekte der Szene sichtbar werden, muss ein *light* oder sogar mehrere hinzugefügt werden. Sonst ist die Szene einfach nur schwarz. Three.js hat verschiedene Belichtungstypen wie beispielsweise *AmbientLight* oder *PointLight*.

**Geometry** Three.js verfügt über leistungsstarke, benutzerfreundliche Objekte für die 3D-Mathematik, z. B. Matrizen, Projektionen und Vektoren. Man kann auch verschiedene Geometrien erzeugen wie zum Beispiel Würfel, Zylinder oder Kugeln. Im nächsten Kapitel werden wir uns noch anschauen wie man einen Zylinder erstellen kann.

**Materials** Materials in Three.js definieren das Aussehen von Objekten. Auch hier gibt es wieder verschiedene Materials, je nach Anwendungsfall<sup>9</sup>. Dabei hat man unzählige Möglichkeiten das Aussehen umzusetzen. Man kann auch Texturen auf das Objekt „mappen“. Auch ein aus der 3D-Modellierung bekanntes *UV-Mapping* ist möglich.

**Mesh** Ein Mesh fasst ein oder mehrere Objekte und Materials und so weiter zusammen. Am Ende wird das Mesh zur Szene hinzugefügt.

### 3.3.2 Modelle

Mit Three.js lassen sich ebenfalls Modelle aus 3D Programmen wie Maya oder Cinema4D darstellen. Dazu werden Loader, also Werkzeuge mit denen man Modelle laden kann, verwendet. Sie sind nicht Teil vom Kern des Framework. Sie können nach Bedarf in die Anwendung mit eingebunden werden. Teilweise sind die Loader noch nicht ganz ausgereift oder es gibt manchmal Importierungsprobleme. Einige bekannte Loader sind: *ColladaLoader*<sup>10</sup>, *OBJLoader*<sup>11</sup> und *JSONLoader*. Ein noch recht neuer Loader, der immer wieder auftaucht ist der *glTF-Loader*<sup>12</sup> der von den Machern von WebGL entwickelt wurde und laut den Entwicklern noch einfacher und performanter arbeiten soll.

### 3.3.3 Events

Three.js liefert einige Funktionen, die bestimmte Events regeln bzw. festlegen, was zu tun ist, wenn bestimmte Aktionen erfolgen. Zum Beispiel hat das `<canvas>`-Element eine feste Größe. Wenn das Fenster des Browsers sich verändert, sollte das Element sich jedoch anpassen. Das ist ganz einfach mit einem *EventListener* und der Funktion *resize* umsetzbar. Hierbei ergänzen sich *Three.js* und *Angular* sehr gut. In dem Framework wird das ganz einfach mit einem Dekorator realisiert. Mit *@HostListener* können Ereignisse in Angular gebunden werden. Sogar benutzerdefinierte Ereignisse können erstellt werden.

---

<sup>9</sup>Aus der 3D-Modellierung sind hier Begriffe wie Lambert oder Phong bekannt. Darauf bauen auch die verschiedenen Materials in der Bibliothek auf.

<sup>10</sup><http://www.khronos.org/collada/>

<sup>11</sup><http://www.martinreddy.net/gfx/3d/OBJ.spec>

<sup>12</sup><https://github.com/KhronosGroup/glTF/tree/master/specification/2.0>

## 3.4 Responsive Webdesign

Der erste Eindruck ist wichtig. Ein Besucher benötigt nur 50 Millisekunden, um sich eine Meinung über eine Website zu bilden. Wenn also die Gestaltung der Webseite keinen guten ersten Eindruck hinterlässt, werden viele potenzielle Kunden einfach gehen[?]. Laut einer Umfrage von *clutch.co* sollen in 2019 nahezu alle Webseiten kleinerer Unternehmen mobil freundlich sein. Es ist offensichtlich, dass immer mehr mobile Geräte verwendet werden. Deshalb liegt es auch auf der Hand Webseiten für diese Geräteklasse anzupassen. Laut einer Studie bevorzugen etwa 3/4 der Benutzer mobil freundliche Webseiten und würden sie auch wieder besuchen [?].

In seinem Artikel Responsive Web Design definiert Ethan Marcotte den Begriff mit 3 Säulen: „*Fluid grids, flexible images, and media queries*“ [Mar]. Er betont aber auch, dass es ein anderes Denken erfordert. „*It's a mechanical concept, the brainchild of a single person, based on finite, specific elements.*“ [Gar] Es gibt also keine klare Definition von Responsive Webdesign. Oder anders gesagt: „*I can tell you how to do Responsive Web Design. How we make things “responsive” is up to us. All of us.*“ [Gar].

Auch wenn es keine klare Definition von *responsive Webdesign* gibt, so ist klar was damit gemeint ist. Webseiten und Webanwendungen sind responsive, wenn sie für alle Geräteklassen angepasst und optimiert sind. Wie man das genau gestaltet, liegt bei jedem selbst.

## 3.5 Usability

Die ISO-Norm 9241 beschreibt eine allgemein gültige Definition von Usability:

*„Usability bezeichnet das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und mit Zufriedenheit zu erreichen.“* [DEI]

Hinter dem englische Begriff *Usability* verbirgt sich also ein umfassenderes Konzept, das allgemein unter *Benutzerfreundlichkeit* verstanden wird. Es beschreibt den Umfang, in dem ein System, ein Produkt oder eine Dienstleistung von bestimmten Benutzern verwendet werden kann, um bestimmte Ziele mit Effizienz und Zufriedenheit in einem bestimmten Nutzungskontext zu erreichen. Jedoch dreht sich nicht alles nur um das Produkt. „*Im Mittelpunkt steht aber immer der User mit seinen Wünschen und Bedürfnissen.*“ [Giz]

*„Software-Anwendungen oder Produkte weisen eine hohe Usability auf, wenn sie von den vorgesehenen Benutzern einfach erlernt und effizient verwendet werden können und diese damit ihre beabsichtigten Ziele und Aufgaben zufriedenstellend ausführen können. Dazu gehören nicht nur ein stimmiges User Interface, sondern auch die passenden Funktionen, um zum Ziel zu gelangen“* [RF]

**User Centered Design (UCD)** *„Hinter diesem Begriff verbirgt sich eine Vielzahl von Gestaltungsprozessen, die den späteren Benutzer ins Zentrum der Entwicklung stellen. Mit der Betonung des Designs wird zum Ausdruck gebracht, dass sowohl Interaktions- als auch Gestaltungsaspekte, also etwa die Gestaltung der Dialogabläufe, die Gestaltung der Form physischer Produkte und Bedienelemente, aber auch das grafische Design, wichtige Bestandteile für eine optimale Benutzung darstellen und von Beginn weg berücksichtigt werden müssen.“*  
[RF]

Wir fassen also zusammen, das *Usability* sich immer um den Benutzer dreht und einfach sowie effiziente Bedienung erfordert. Dabei muss die *Benutzerfreundlichkeit* aber immer im Kontext seiner Verwendung beurteilt werden.

## 3.6 Performance

*Die Leistung vieler Websites hängt von der Auslastung der Website zu Spitzenzeiten unter verschiedenen Bedingungen ab. Leistungstests werden normalerweise in einer vernünftig simulierten Umgebung mit Hilfe von Leistungstestwerkzeugen durchgeführt. Die Leistung einer Website hängt jedoch von verschiedenen Parametern ab, und jeder Parameter muss unter verschiedenen Belastungsniveaus getestet werden.* Aufgrund der Komplexität von Websites ist es nicht möglich, einen gemeinsamen Nenner für Leistungsparameter zum Testen der Website zu zeichnen. Verschiedene Teile der Website müssen mit unterschiedlichen Parametern unter verschiedenen Bedingungen und Belastungsniveaus getestet werden. In solchen Fällen muss die Website in viele Komponenten zerlegt werden, die das Verhalten verschiedener Geschäftskomponenten darstellen. Diese Geschäftskomponenten werden verschiedenen Objekten zugeordnet, die das Verhalten und die Struktur des Teils der Website wirklich darstellen. Diese Objekte werden Leistungstests mit verschiedenen Parametern und Belastungsniveaus unterzogen. In diesem Dokument wird der neue Testprozess angesprochen, bei dem das Konzept der Zerlegung des Verhaltens der Website in testbare Komponenten verwendet wird, die auf testbare Objekte abgebildet werden. Diese überprüfbaren Objekte werden Leistungstests unter verschiedenen Leistungsparametern und Belastungsniveaus unterzogen.

Mit diesem Kapitel wurden alle relevanten Grundlagen für die Umsetzung des Projekts erläutert. Der Leser sollte nun ein Grundwissen von Angular und WebGL haben und ein Grundverständnis von SPAs und komponentenbasierter Programmierung. Auch wurden die wichtigen Begriffe responsive Webdesign, Performance und Usability definiert.



## Kapitel 4

# Anforderungen und Konzeption

Unter funktionalen Anforderungen versteht man konkrete Funktionalitäten, welche die Anwendung leisten soll. Die Anwendung lässt sich in mehrere Module aufteilen, die dem User zu Verfügung stehen. In diesem Kapitel werden die Anforderungen erläutert. Anschließend wird auf das Konzept der Anwendung eingegangen, damit deutlich wird, wie der technische Aufbau aussehen soll.

### 4.1 Funktionale Anforderungen

Unter funktionalen Anforderungen versteht man konkrete Funktionalitäten, welche die Anwendung leisten soll. Die Anwendung lässt sich in mehrere Module aufteilen, die dem User zu Verfügung stehen. In diesem Kaptitel werden sowohl die funktionalen also auch die nicht funktionalen Anforderungen beschrieben. Es handelt sich zwar um ein echtes Projekt, wobei es sich vorerst um eine prototypische Entwicklung handelt. Als Anforderung vom Hersteller gilt nur, dass der Konfigurator einen Bestellvorgang für den Kunden und das Unternehmen vereinfacht. Dabei soll durch innovative Gestaltung und Programmierung des Konfigurators eine performante Lösung entwickelt werden. Das Ergebnis sollte zum bisherigen Webauftritt des Unternehmens passen. Nun wollen wir uns die Anforderungen im Details anschauen, bevor das Konzept der Webanwendung vorgestellt wird.

**3D Vorschau des Bechers** Zunächst ist wichtig, das die 3D Darstellung des Bechers umgesetzt wird. Dabei spielt das Design, welches der User hochladen wird, erst einmal keine Rolle. Schon beim Start wird Standardmäßig des 3D-Modell eines Bechers angezeigt. Von den Mehrwegbechern sind vier verschiedene Größen vorhanden. Dem Benutzer soll es möglich sein über das Konfigurationsmenü eine beliebige Größe auszuwählen. Ohne lange Wartezeiten muss sich das 3D-Modell des Bechers dementsprechend aktualisieren. Falls der User schon ein Design hochgeladen hat und die Größe nun ändern möchte, wird das Design gelöscht und muss erneut hochgeladen werden. Grund dafür sind die unterschiedlichen Designvorgaben für jede Größe. Man soll vor dem Löschvorgang allerdings einen Warnhinweis bekommen, um ungewolltes Löschen zu vermeiden.

#### 4. ANFORDERUNGEN UND KONZEPTION

---

Nr.	Bezeichnung	Beschreibung
1	Auswahl des Bechers	Der Benutzer kann zwischen verschiedenen Bechergrößen wählen.
2	Interaktionen	Der Benutzer kann beispielsweise den Becher transformieren oder an den Becher heranzoomen.
3	Upload des Designs	Der Benutzer kann nach sein Design entsprechende der Druckvorgaben hochlade.n
4	Zusatzoptionen	Der Benutzer kann bestimmte Zusatzoptionen auswählen.
5	Übersicht anzeigen	Der Benutzer kann sich eine Übersicht seiner Konfiguration ansehen.
6	Konfiguration speichern	Der Benutzer kann seinen konfigurierten Becher speichern.

**Tabelle 4.1:** Übersicht der funktionalen Anforderungen

**Interaktionen** Da es sich um eine 3D Ansicht handelt, sollte man auch interagieren können. Der Becher kann von Benutzer 360 Grad auf der X-Achse und auf der Y-Achse 180 Grad gedreht werden. Auch eine Zoom-Funktion soll implementiert werden. Damit kann der User näher an den Becher heranzoomen. Es sollte jedoch ein maximaler Zoom eingestellt werden, damit die Ansicht realistisch bleibt. Außerdem soll zur einfacheren Bedienung ein Button erstellt werden, der den Becher wieder in seine Ursprungsposition bringt. Diese Interaktionen sollten sowohl mit der Maus als auch mit Touch-Eingabe funktionieren.

**Upload des Designs** Normalerweise kann ein Kunde einen Mehrwegbecher über Flyeralarm<sup>1</sup> drucken lassen. Dabei muss man sich an die Design-Vorgaben halten, die im Datenblatt<sup>2</sup> definiert sind. Diese unterscheiden sich je nach Größe des Bechers, da es sich ja um ein vollflächigen Druck handelt. Um den Benutzer Arbeit zu ersparen, sollte er das Design genau nach den Vorgaben der Druckerei hochladen. Beim Upload wird die Datei auf das korrekte Seitenverhältnis geprüft. Ohne lange Wartezeit wird nun das Design auf den Becher gerendert und der Kunde kann sich das Ergebnis anschauen. Es sollte auch möglich sein das Design mit einem anderen zu ersetzen. Wie schon erwähnt wird das Design bei ändern der Bechergröße entfernt. Als Dateiformat sind ausschließlich png-Dateien zulässig, weil damit Transparenzen dargestellt werden können.

---

<sup>1</sup><https://www.flyeralarm.com/de/>

<sup>2</sup>siehe Anhang



**Zusatzoptionen wählen** Hierbei handelt es sich um optionale Zusatzfunktionen die nicht unbedingt im Rahmen des Bachelorprojektes umgesetzt werden. Eine Zusatzoption wäre das ein und ausblenden des Eichstrichs, der bei den Mehrwegbechern mit aufgedruckt wird. Eine weitere Zusatzangabe des Users wäre eine Eingabe der Stückzahl, die bestellt werden soll. Das wäre sinnvoll wenn der Konfigurator in Zukunft in den Bestellvorgang eingebaut werden soll. Andernfalls hätte der Kunde zumindest die Möglichkeit eine Preisvorschau in der Übersicht zu sehen.

**Übersicht anzeigen** Wie gerade erwähnt soll es eine Übersicht geben. In Form einer Tabelle soll dem Kunden nun noch einmal seine Auswahl zusammengefasst werden. Er kann dort die ausgewählte Bechergröße, das hochgeladene Design (den Dateinamen), eventuell die Stückzahl, Preisvorschau und so weiter, sehen. Optional kann eine Druck-Funktion implementiert werden, die es dem Benutzer ermöglicht die Übersicht inklusive einer Abbildung des Bechers mit dem Design zu drucken bzw. in einer pdf-Datei zu speichern.

**Konfiguration speichern** Hierbei handelt es sich wieder um eine optionale Funktion, die wahrscheinlich nicht mehr im Rahmen der Bachelorarbeit umgesetzt werden kann. Wenn ein Benutzer einen Becher konfiguriert hat, ist es durchaus denkbar das er zu späteren Zeiten noch einmal darauf zurückgreifen möchte. Oder er möchte verschiedene Versionen eines Bechers vergleichen. Deshalb wäre es sinnvoll eine Speicher-Funktion zu implementieren, die einfach und schnell den konfigurierten Becher wiederherstellen kann.

## 4.2 Nicht Funktionale Anforderungen

Mit nicht funktionalen Anforderungen sind Funktionalitäten gemeint, welche die Anwendung **im Hintergrund** leisten soll. Diese sollen die Usability und performance des Konfigurators verbessern und dem Benutzer somit ein innovatives Erlebnis bieten. In der Tabelle sehen sie eine Übersicht der nicht funktionalen Anforderungen.

**Uploadfilter** Das Design sollte realistisch gerendert werden. Dazu muss das Design wie in den Druckvorgaben hochgeladen werden. Sonst könnte es zu verzerrt aus sehen oder einfach unscharf. Nach dem Upload bekommt der User eine Rückmeldung, ob die Auflösung des Bildes in Ordnung ist. Falls nicht wird das Bild nicht gespeichert bzw. auf den Becher gerendert und der Benutzer soll das Design in der richtigen Auflösung hochladen. Ein ähnliches Vorgehen wird auch bei der Druckerei Flyeralarm bei einem Upload angewendet. Jedoch wird bei dem Konfigurator nur das Seitenverhältnis bzw. die Auflösung geprüft, nicht der Farbmodus oder andere Vorgaben zum Drucken.

**Bedienungshilfen** Ähnlich wie beim Upload des Designs soll der Kunde immer wieder Hinweise bekommen was zu tun ist. Beispielweise wie er den Becher drehen kann oder wo er die Größe des Bechers auswählen kann oder wo er die Übersicht seiner Auswahl sehen kann.

#### 4. ANFORDERUNGEN UND KONZEPTION

---

Nr.	Bezeichnung	Beschreibung
1	Uploadfilter	Der Benutzer kann zwischen verschiedenen Bechergrößen wählen.
2	Bedienungshilfen	Der Benutzer kann beispielsweise den Becher transformieren oder an den Becher heranzoomen.
3	Realistische Design-Darstellung	Der Benutzer kann nach sein Design entsprechende der Druckvorgaben hochladen
4	Responsive Webdesign	Der Benutzer kann bestimmte Zusatzoptionen auswählen.
5	Benutzerfreundlich	Der Benutzer kann sich eine Übersicht seiner Konfiguration ansehen.
6	Leistungstark	Der Benutzer kann seinen konfigurierten Becher speichern.

**Tabelle 4.2:** Übersicht der funktionalen Anforderungen

**Realistisches Design-Rendering** kubisches Problem Auch was die 3D-Darstellung angeht, sind nicht viele Vorgaben gegeben, was einen weiten Spielraum lässt. Zunächst muss der Becher in die Szene geladen werden können. Anschließend besteht die Herausforderung darin, ein vom Benutzer hochgeladenes Design realistisch auf den Becher zu projizieren. Dabei muss die kubische Form des Bechers und die Upload-Vorgaben der Druckerei berücksichtigt werden. Der Upload-Vorgang muss einfach und verständlich sein, damit der Benutzer weiß wie er sein Design richtig auf den Becher bekommt. Dabei dürfen die Ladezeiten nicht zu lange sein. Die Anwendung muss möglichst performant laufen, unabhängig von der Plattform.

**Responsive Webdesign** Es ist wichtig das der Konfigurator auf verschiedenen Displaygrößen verwendet werden kann. Bei mobilen Geräten wird es schwierig die 3D Ansicht und das gesamte Menu zusammen anzuzeigen. Damit die Inhalte groß genug und gut lesbar sein, wird das Menu auf mobilen Gräten also *Off-Canvas-Menu* implementiert. Somit wird zum einen der Becher schön groß dargestellt und zum anderen werden Menu-Items groß und gut lesbar sein. Der Benutzer eines Smartphones sollte nach Möglichkeit den gleichen Funktionsumfang genießen wie ein Benutzer auf Desktop Geräten. Dabei sollte möglichst innovativ gestalet werden.

**Benutzerfreundlichkeit** Die Webanwendung soll benutzerfreundlich sein. Dazu tragen die schon erwähnten Bedienungshilfen bei. Wichtig ist, das der Benutzer immer weiß was der

nächste Schritt ist. Um dies zu erreichen ist eine einfache und übersichtliche Gestaltung ist nötig. Sie sollte auf das wesentliche beschränkt sein und nicht überladen wirken.

**Performant** Wie die Studie im Kapitel Performance gezeigt hat, mögen Benutzer keine langen Wartezeiten. Bei dem 3D-Konfigurator soll das Laden der 3D Modelle nicht länger als 2ms dauern. Das Rendern des Designs auf den Becher ebenfalls nicht länger als 2ms. Die Länge des Uploads vom Design hängt von der Internetverbindung ab, sollte allerdings auch schnell abgeschlossen sein. Das Transformieren des Modells in der 3D Szene sollte flüssig funktionieren und keine Verzögerungen enthalten. Alles in allem sollte die gesamte Anwendung performant laufen und im Bezug auf die Leistung an eine Desktopanwendung erinnern. Besonders interessant wird dieser Aspekt auf mobilen Geräten, denn auch dort sollte die Anwendung möglichst stabil laufen.

## 4.3 Konzeption

Bevor es an die Umsetzung und Implementierung geht, beschäftigt sich dieser Abschnitt damit, **wie** der Konfigurator umgesetzt wird. Um den technischen Aufbau zu verstehen, wird ein Flussdiagramm sowie ein Komponentendiagramm verwendet und im Laufe dieses Abschnitts erläutert. Anschließend wird auch der grobe Aufbau der Benutzeroberfläche beschrieben.

### 4.3.1 Ablauf einer Konfiguration

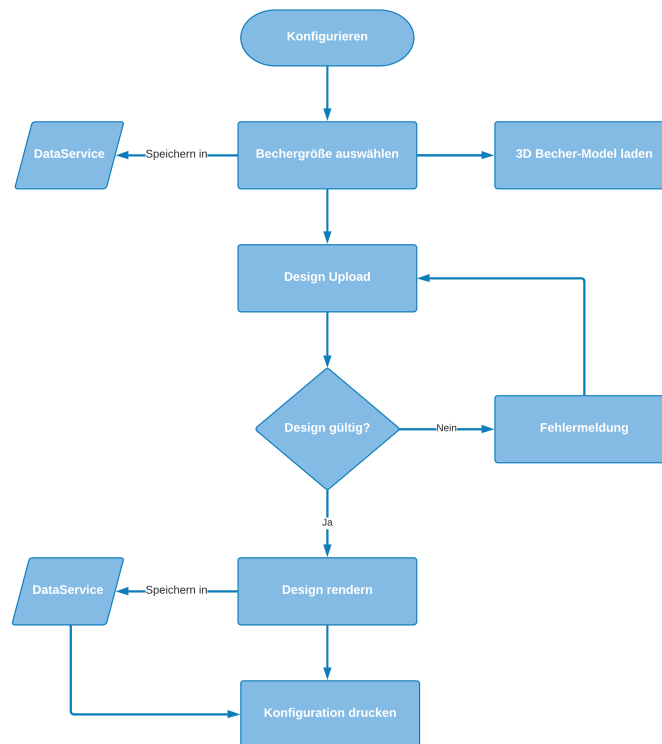
Nachdem alle Funktionen und Vorgaben bekannt sind, sollte man sich über den technischen Ablauf Gedanken machen. Das Flussdiagramm<sup>3</sup> in Abbildung 4.1 zeigt, was während dem Konfigurieren des Bechers geschieht.

Der Konfigurator für Mehrwegbecher beinhaltet vier verschiedene Größen von Bechern. Als erstes soll der Benutzer eine Bechergröße auswählen. Standardmäßig ist der Becher mit 0,2 Liter als 3D-Model geladen. Falls der Benutzer eine andere Größe wählt, aktualisiert sich das 3D-Model dementsprechend. Anschließend kann der Upload des Becher-Designs erfolgen. Hierbei muss sich der Benutzer an die Druckvorgaben halten. Wenn das Design beispielsweise in einem falschen Dateiformat ist oder eine falsche Auflösung hat, kann das Design nicht hochgeladen werden. Falls alles richtig ist, wird das Design direkt auf den Becher gerendert. Die Wahl der Größe und die hochgeladene Datei werden in dem DataService gespeichert. Am Ende der Konfiguration kann sich der User in der Übersicht alles noch einmal ansehen und bei Bedarf ausdrucken. Dabei werden die Daten aus dem DataService verwendet.

Jetzt, wo das Flussdiagramm den Ablauf einer Konfiguration beschrieben hat, sollte man sich den Komponenten der Anwendung widmen. Das Komponentendiagramm (siehe Abbildung 4.2) zeigt die Hauptkomponenten, welche später auch als Angular-Bausteine angelegt werden. Natürlich gibt es noch weitere Kindskomponenten, die in den Hauptkomponenten

---

<sup>3</sup>engl. Flowchart sollte hier als Fussnote kurz erklärt werden damit jeder weiß, was genau damit gemeint ist.

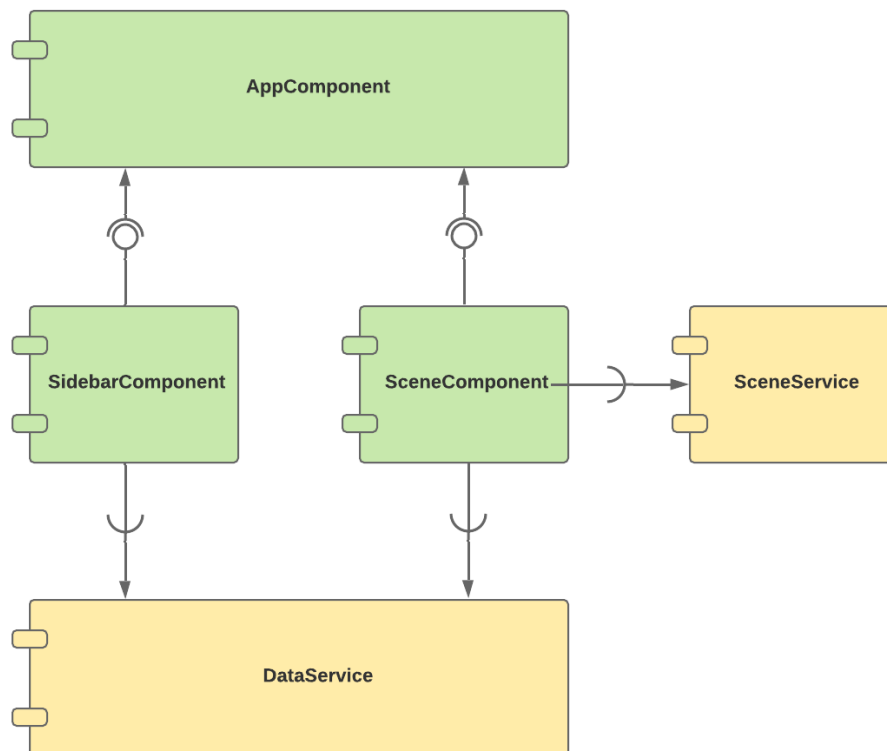


**Abbildung 4.1:** Flussdiagramm der Angular-Anwendung

verwendet werden. Darauf gehen wir aber erst in der Implementierung ein. Wie wir wissen ist die AppComponent unsere Basis-Komponente, von der alles startet. Nun hat unser Konfigurator eine SidebarComponent, welche das Konfigurationsmenü sowie die Übersicht der Konfigurationen Beeinhaltet. Das Herzstück unserer Angular-Anwendung ist die SceneComponent. Sie zeigt uns den Becher in 3D mit dem hochgeladenen Design. Um eine 3D Szene aufzubauen wird der SceneService verwendet, welcher die 3D Bibliothek Three.js nutzt. Wie schon in dem Flussdiagramm erwähnt befinden sich alle Konfigurations-Daten in dem DataService, welcher global für alle Komponenten bzw. Module zugänglich ist. Damit sollte der technische Aufbau der Anwendung klar sein.

### 4.3.2 Benutzeroberfläche

Beschäftigen wir uns nun mit der grafischen Benutzeroberfläche (UI). Am Anfang war ein 3-Schritte-Konfigurator geplant. Der Benutzer sollte also in drei Schritten seinen Becher konfigurieren. Da der Konfigurator nur wenige Einstellmöglichkeiten hat, ist es am benutzerfreundlichsten alles auf einer Seite zu konfigurieren. Somit ist die Webanwendung sehr einfach gehalten und so aufgebaut, dass eine mobile Anpassung leicht umzusetzen ist. Wie in der Abbildung zu sehen ist auf der linken Seite die 3D Szene mit WebGL aufgebaut. Hier kann der User seinen Becher betrachten oder ihn auch transformieren. Auf der rechten Seite befindet sich die Sidebar. Darin ist die komplette Menüführung mit den Konfigurationseinstellungen

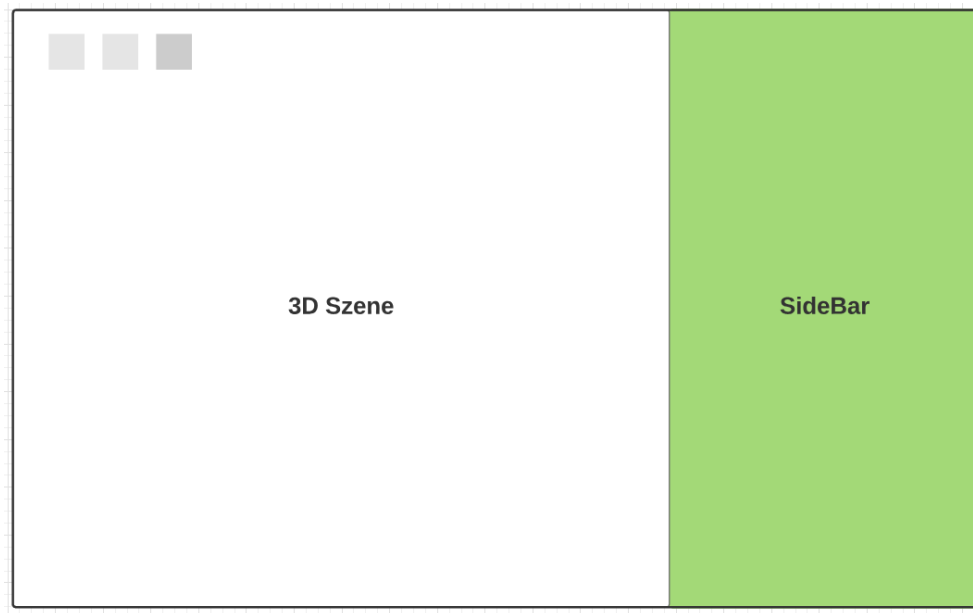


**Abbildung 4.2:** Komponentendiagramm der Angular-Anwendung

vorhanden, sowie die Übersicht aller gewählten Einstellungen. Das Menü ist hierarchisch aufgebaut, sodass der Benutzer sich von oben nach unten arbeiten kann. Zuerst wählt er die Größe des Bechers. Anschließend kann er sein Design hochladen und danach noch optionale Einstellungen wählen. Innerhalb der Sidebar befindet sich auch die Übersicht in Form einer Tabelle. Es kann über eine Tab-Navigation immer zwischen den Einstellungen und der Übersicht gewechselt werden. Die Einstellungen sollen als Accordion umgesetzt werden. Somit ist immer nur ein Menüpunkt aufgeklappt. Das macht das ganze noch etwas übersichtlicher und auch mobilfreundlicher.

Bei Tablets (Portraitansicht) befindet sich die Sidebar unterhalb der 3D Ansicht. So ist genug Platz für die Sidebar und gleichzeitig wird der Becher noch groß genug dargestellt. Auf Smartphones ist die Sidebar als Off-Canvas-Menu gelöst, das heißt, sie muss über ein Hamburger-Menü-Button geöffnet werden. Somit ist das Menü dann vollflächig (siehe Abbildung). Im Grunde verändert sich das Aussehen der Sidebar aber nicht auf der mobilen Ansicht.

**Styles** Der Schwerpunkt des Bachelorprojektes liegt bei der Entwicklung mit Angular und mit WebGL. Deshalb soll nicht viel Zeit für das Styling verwendet werden. Trotzdem soll es ein gutes und innovatives Design geben. Als Grundlage werden die Styles der Unternehmenswebseite



**Abbildung 4.3:** *Mockup der Angular-Anwendung*

sowie Bootstrap verwendet. Somit sind grundlegende Styles wie Schrift und Farben schon festgelegt und müssen nicht mehr betrachtet werden.

Am Ende diesen Kapitels sollte deutlich sein, was genau die zu lösenden Probleme sind. Es wurde auch das Konzept der Anwendung beschrieben und wie die einzelnen Teile der Anwendung umgesetzt werden. Im nächsten Kapitel folgt die Realisierung. Dort wird auf das Konzept zurückgegriffen und dieses unter Umständen noch ergänzt.

## Kapitel 5

# Umsetzung & Implementierung

Nun kommen wir an den praktischen Teil der Arbeit. In diesem Kapitel wird zunächst einiges über die Entwicklungsumgebung gesagt. Anschließend wird auf wesentliche Punkte der Umsetzung und Implementierung eingegangen. Dabei wird wieder auf die Anforderungen zurückgegriffen und erläutert wie diese umgesetzt wurden. Voraussetzung für dieses Kapitel ist das Grundlagenkapitel der Arbeit, um zusammenhänge mit den Technologien besser verstehen zu können.

### 5.1 Entwicklungsumgebung

Dieser Abschnitt befasst sich mit den verwendeten Technologien, die eine produktive Entwicklung ermöglichen. Es wird auf alle verwendeten Technologien, Bibliotheken sowie Programme eingegangen und begründet, warum eine Verwendung sinnvoll ist.

**Technologien und Bibliotheken** Wie in den vergangenen Kapiteln schon erläutert sind die Haupttechnologien das Framework *Angular* sowie die 3D Bibliothek *Three.js*, welche WebGL zum Einsatz bringt. Angular Anwendungen werden grundsätzlich mit *Node.js* installiert. Dadurch ist es uns später möglich verschiedene npm-Module wie *Three.js* oder auch *Bootstrap* zu installieren. Selbst die *Angular CLI* wird damit installiert. Zusätzlich nutzt Angular *Webpack*, um den Quellcode zu bündeln. Dabei werden die *scss*-Dateien in eine *css*-Datei komprimiert. *Sass* in Verbindung mit *Webpack* ist nicht nur in für Angular-Projekte ein beliebtes Tool zum entwickeln von Webseiten und Webanwendungen. Auch eine Versionierung des Quellcodes kommt zum Einsatz. Mit *Git* ist der Quellcode immer gesichert und man kann nach dem Testen einer bestimmten Funktion immer wieder zum letzten Push zurückkehren. Bei einem Bachelor-Projekt kann man zwar nicht den Vorteil der Teamentwicklung mit *Git* nutzen, jedoch kann man damit sehrwohl seinen Quellcode versionieren und sichern, was bei einem Bachelorprojekt durchaus sinnvoll erscheint.<sup>1</sup>

---

<sup>1</sup>Der komplette Quellcode des Projekts sowie die Dokumentation finden Sie unter <https://github.com/luk9like/gc-app>

**Entwicklungswerkzeuge** Als Text-Editor wird *PhpStorm* verwendet. Dadurch ist eine flexible Programmierung an unterschiedlichen Systemen möglich, da der Editor für Windows, MacOS als auch für Linux verfügbar ist. Er bietet eine praktische Integration von Git und kann dazu noch sehr gut mit Angular-Anwendungen arbeiten. Auch Angular CLI Befehle können teilweise direkt in dem Editor ausgeführt werden und *PhpStorm* verfügt über eine logische Code-Verfolständigkeit. Nicht zu vergessen die optimale Quellcode-Formatierung für jeden Dateityp und die Kompatibilität mit *TSlint*<sup>2</sup> *PhpStorm* verfügt zwar über ein integriertes Terminal, jedoch ist das meist sehr unpraktisch, da es nur unnötig Platz in der Anwendung wegnimmt. Deswegen werden Kommandozeilen einem zusätzlichen Terminal ausgeführt. Gerade am Anfang des Projekts wird die Angular CLI viel zum Einsatz kommen.

### 5.2 Anlegen des Angular-Projekts

Bevor die Angular-Anwendung nach und nach aufgebaut werden kann, muss Angular erst einmal installiert werden und anschließend ein Projekt erstellt werden. Mit der Angular CLI ist das sehr einfach und unproblematisch. Deshalb ist der erste Schritt diese über Node.js zu installieren. Anschließend kann es auch schon los gehen. Wie schon erwähnt wird *PhpStorm* verwendet. In diesem Editor ist es möglich ein *Angular CLI Projekt* zu erstellen. Wie in

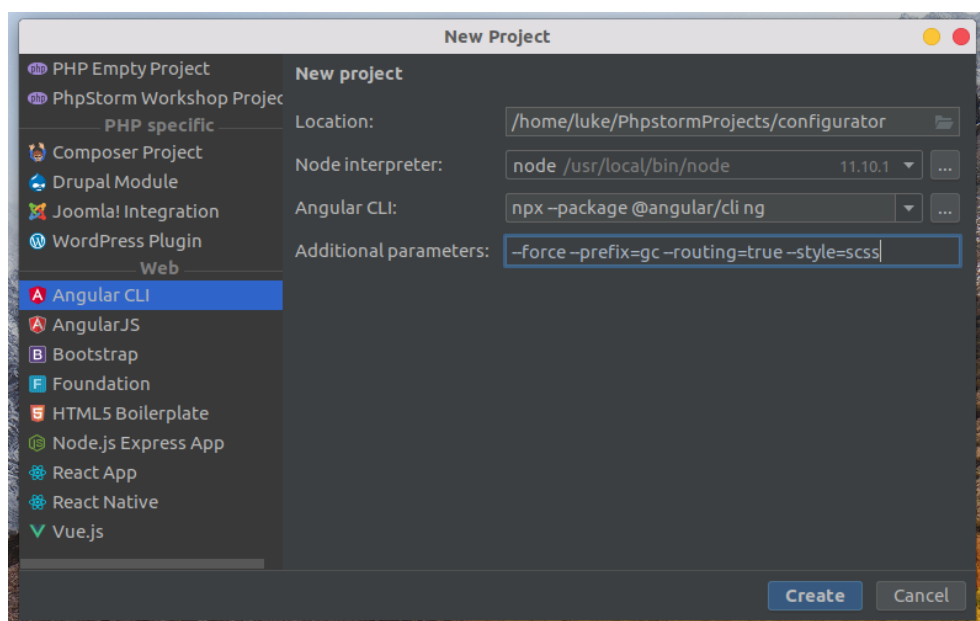


Abbildung 5.1: Anlegen eines Angular-Projekts mit PhpStorm

Abbildung 5.1 zu sehen, erkennt das Programm automatisch die *Node.js* Version sowie die

<sup>2</sup>**TSLint** ist ein erweiterbares statisches Analysewerkzeug, das TypeScript- Code auf Lesbarkeit, Wartbarkeit und Funktionsfehler überprüft. Es wird weitgehend von modernen Editoren und Build-Systemen unterstützt und kann mit Ihren eigenen Fusselregeln, Konfigurationen und Formatierungselementen angepasst werden (vergleiche <https://www.npmjs.com/package/tslint>).



Angular CLI Version, welche installiert wurden. Nun muss noch ein Pfad angegeben werden, unter dem das Projekt gespeichert wird. Es sind auch einige optionale Parameter angegeben. Mit `prefix` kann ein eigener Prefix für die Selektoren gewählt werden. Standardmäßig hat jeder Baustein in Angular vor seinem Namen ein `app` stehen. Das ist hier ersetzt worden durch `gc`. Das `RoutingModule` ist optional und kann mit dem Parameter `routing` aktiviert werden. Der letzte Parameter `style` ist eigentlich nicht unbedingt nötig. Er definiert mit welchen Styles gearbeitet werden soll (`.css`, `.scss`, `.sass`). Wenn dieser Parameter fehlt, fragt die neue Angular CLI Version automatisch beim generieren des Projekts, welcher Typ verwendet werden soll. Ist der Vorgang abgeschlossen, kann man die Anwendung schon starten und sieht den Standard Begrüßungsbildschirm. Erreichbar ist die Anwendung unter `http://localhost:4200`.

## 5.3 Module des Konfigurators

Im folgenden Abschnitt wird betrachtet aus welchen Module die Angular Anwendung aufgebaut ist. Anschließend werden die Module angelegt. Es geht hier nur um das Anlegen der Module, noch nicht um die Funktionalität der einzelnen Module. Die Abbildung zeigt eine Übersicht aller Module, welche in der Webanwendung implementiert sind. Hierbei ist zu beachten, dass die Abbildung auf wesentlichen Module beschränkt ist, da es sonst zu unübersichtlich werden würde. Eine vollständige Übersicht der Module befindet sich im Anhang. Die einzelnen

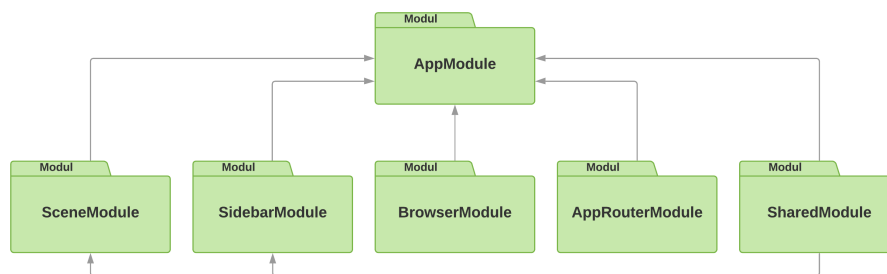


Abbildung 5.2: Module der Angular-Anwendung

Bausteine der Angular-Anwendung werden im Rahmen dieses Bachelorprojektes immer über die Angular CLI generiert, da das am einfachsten funktioniert und zeitsparend ist. Ein Modul kann mit folgendem Command Line Befehl generiert werden:

Listing 5.1: Erstellen eines Modules mittels Angular CLI

```
|| ng g m shared
```

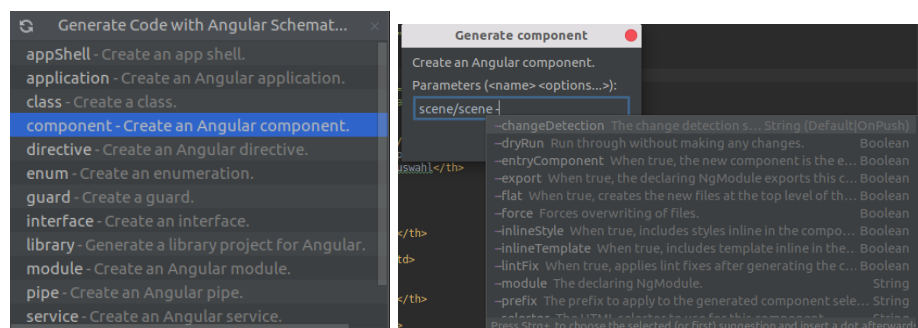
Als Beispiel wurde hier das Module `shared` angelegt. Hierin werden später alle globalen Bausteine der Angular-Anwendung erstellt. Es wurden außerdem die Appkürzungen `m` für `module` und `g` für `generate` verwendet. Ob ausgeschrieben oder abgekürzt, das hat keine Auswirkungen, sondern spart lediglich Zeit. Es gibt noch ein paar Zusatzoptionen, die

als Parameter angegeben werden können. Diese sind in der offiziellen Dokumentation von Angular aufgeführt. Diese ermöglichen verschiedene Funktionen wie `export`-Funktion oder verhindern das generieren von Testdateien. Der 3D Konfigurator hat noch zwei weitere wichtige Module. Zum einen ist das *SceneModule* für die 3D Darstellung zuständig und das Rendern des Designs. Das *SidebarModule* ist sozusagen das Konfigurationsmenu inklusive einer Übersicht in tabellarischer Form. Die hauseigenen Module von Angular wie das *BrowserModule* wurden von der Angular CLI automatisch importiert. Hier ist also kein weiterer Schritt notwendig, die Anwendung sollte lauffähig sein. Noch hat die Applikation allerdings noch keine richtige Funktionalität.

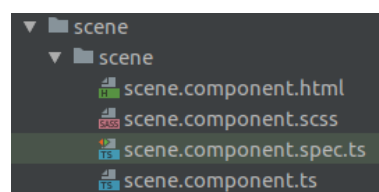
### 5.4 Komponenten des Konfigurators

Sowohl das *SidebarModule* als auch das *SceneModule* des Konfigurators haben sozusagen eine „Hauptkomponente“. Diese werden dann später in der *AppComponent* eingebunden. Die Basis-Komponente der Anwendung ist also sehr einfach und übersichtlich. Dann gibt es noch einige weitere Kindskomponenten sowie geteilte Komponenten (aus dem *sharedModule*). In diesem Kapitel werden wir uns zunächst auf die Komponenten beschränken. Alle weiteren Bausteine wie Services werden wir später noch genauer betrachten.

Die *SceneComponent* ist dafür da die komplette 3D-Szene zu erzeugen und eben das Becher-Model mit seinem Design darstellen soll. Wie alle Komponentente wurde diese mittels Angular CLI in PhpStorm angelegt, wie in Abbildung 5.3 zu sehen. In PhpStorm kann man über [File](#)



(a) Auswahl generierbarer Angular-Bausteine (b) Generieren einer Komponente mit Parametern



(c) Generierte Dateien durch die CLI

**Abbildung 5.3:** Erstellung einer Angular-Komponente mit PhpStorm

> [New](#) > [Angular Schematics](#) innerhalb des Editors die Angular CLI verwenden. Wie im ersten

Bild zu sehen, lassen sich so alle Bausteine von Angular anlegen. In diesem Fall wird eine Komponente angelegt. Nach der Auswahl wird man nun aufgefordert den Komponenten-Namen sowie optionale Parameter mit anzugeben. Das praktische ist, das PhpStorm alle verfügbaren Parameter vorschlägt und man nicht etwa in der Documentation nachschauen müsste, welchen Parameter man braucht und vermeidet Tippfehler. Nach bestätigen der Eingabe werden nun wie in Bild c) zu sehen alle erforderlichen Dateien der Komponente generiert. Zusätzlich werden im Hintergrund eventuelle Import-Anweisungen sowie Export-Anweisungen und so weiter automatisch ergänzt. In unserem Fall wurden auch die Testdatei erzeugt, welche wir uns im nächsten Kapitel genauer anschauen wollen. Die anderen drei Dateien sind wie wir wissen für Logik, Styles und Templating zuständig. Analog zu dem Erstellen der Komponente in PhpStorm funktioniert das ganze auch in einem Terminal:

Listing 5.2: Generieren einer Szene mit Angular CLI

```
|| ng g c scene/scene -m=scene
```

Sobald die Komponente angelegt ist, hat sich eigentlich keine Funktionalität. Das wird erst später mit der Logik und den Services implementiert. Für den Anfang reicht es, hier schonmal das `<canvas>`-Element anzulegen und ein paar Styles in der `scss`-Datei der Komponente anzulegen. Wer sich mit `scss` bzw. `sass` auskennt, kann hier die Schreibweise nutzen. Man kann aber auch normales `css` schreiben. Damit ist die *SceneComponent* im Grunde schon fertig.

**Kindskomponente** Genauso wie die *SceneComponent* wurde auch die *SidebarComponent* angelegt. Sie stellt die komplette Sidebar im rechten Bereich dar, also das Konfigurationsmenü und die Übersicht. Diese Komponente ist etwas komplexer und hat noch ein paar Kindskomponenten. Die Sidebar ist mit der Bootstrap-Komponente *Tabs*<sup>3</sup> umgesetzt. Wir haben also im Menu zwei Tabs, Konfigurator und Übersicht. Diese Tabs sind als eigene Komponente in *SharedModule* implementiert und können in unserer *SidebarComponent* also Kindskomponente verwendet werden. Es ist sozusagen ein eigener Typ für *Tabs* definiert. Die Umsetzung der Tab-Komponente wird nun genauer beleuchtet.

Die Tabs von Bootstrap sind in einem `<ul>`-Element implementiert, welcher sozusagen den Container darstellt. Die einzelnen Tabs sind dann `<li>`-Elemente und werden durch die Liste gebündelt. Am Ende entsteht folgendes Template für die Tabs-Komponente:

Listing 5.3: Ausschnitt der Tabs-Klasse

```
|| <ul class="nav nav-tabs nav-fill">
||   <li class="nav-item" *ngFor="let tab of tabs" (click)="
||       selectTab(tab)">
||       <a [class.active]="tab.active" class="nav-link nav-
||           title" href="#">{{tab.title}}</a>
||   </li>
|| </ul>
```

<sup>3</sup><https://getbootstrap.com/docs/4.0/components/navs/#tabs>

```
|| <ng-content></ng-content>
```

Die `<ul>`-Liste hat ein paar vordefinierte Bootstrap-Klassen, auf die hier nicht weiter eingegangen wird. Sobald ein Tab aktiv ist, soll der dazugehörige Inhalt angezeigt werden. Wie im Quellcode zu sehen gibt es nur ein `<li>`-Tag, was nicht heißt, das es nur einen Tab gibt. Die Komponente sollte eine beliebige Anzahl an Tabs haben können. Das ist in diesem Fall dynamisch realisiert. Mit der strukturellen Direktive `*ngFor` wird geprüft, wie viele Tabs im Template erstellt wurden und gibt diese dann mithilfe der `for`-Schleife aus. Damit wird allerdings noch nicht der Inhalt des aktiven Tab angezeigt. Dazu wird noch die `selectTab()`-Funktion benötigt. Bei `onClick` wird diese ausgeführt und teilt der Komponente mit, welcher Tab aktiv ist. Anschließend wird der Content durch `<ng-content>` ausgegeben. Um nun die Tabs als Komponente zu verwenden brauchen wir noch ein Tab-Komponente. Die Tabs-Komponente definiert, wie die Navigation insgesamt aufgebaut ist und die Tab-Komponente beschreibt, wie die einzelnen Tabs aufgebaut sind. Im Grunde werden hier nur die Inhalte des Tabs in einem `div` ausgegeben. Wie oben im Quellcode zu sehen greifen wir auf die Variable `tab.active` und `tab.title` zu. Diese sind in Tab definiert. In unserer Sidebar würde die Tabs folgendermaßen verwendet:

Listing 5.4: Umsetzung der Tab-Navigation

```
<gc-tabs>
  <gc-tab [tabTitle]='Konfigurator'>
    Content 1
  </gc-tab>
  <gc-tab [tabTitle]='Overview'>
    Content 2
  </gc-tab>
</gc-tabs>
```

Es könnten nun noch beliebig viele `<gc-tab>`-Elemente hinzugefügt werden und sie würden auch als Tab angezeigt werden. Nun ist bei jedem Tab ein `title` angegeben, der als Titel im Tab angezeigt werden soll. Dieser lässt sich in der Tab-Klasse über `@input` in der Variable `title` speichern und kann so an der richtigen Stelle ausgegeben werden. Damit ist Tabs-Komponente funktionsfähig. Die Tabs werden dargestellt und bei `onClick` wird der entsprechende Tab-Inhalt angezeigt. In Abbildung 5.7 ist das Endergebnis zu sehen.

Zusätzlich hat die `SidebarComponent` auch noch eigene nicht geteilte Kindskomponenten. Das Konfigurationsmenü besteht aus Größe, Design-Upload und Optionen. Diese drei sind ebenfalls als Menü umgesetzt. Es handelt sich hier wieder um eine Bootstrap Komponente, das `Accordion`, welches ein aufklappbares Menü ist (*Collapse*)<sup>4</sup>. Die einzelnen Punkte sind hier auch wieder eigenständige Komponenten die als Kindskomponenten in der `SidebarComponent` verwendet werden. Allerdings sind diese Komponenten weniger komplex. Hier sind alle Menüpunkte unterschiedlich aufgebaut, weshalb es 3 verschiedene Komponenten gibt. Das `Accordion` wird wie in der Dokumentation von Bootstrap verwendet. Wie das ganze am Ende aussieht kann man auch in Abbildung 5.7 sehen. Die Komponenten `<gc-size>`,

---

<sup>4</sup><https://getbootstrap.com/docs/4.0/components/collapse>

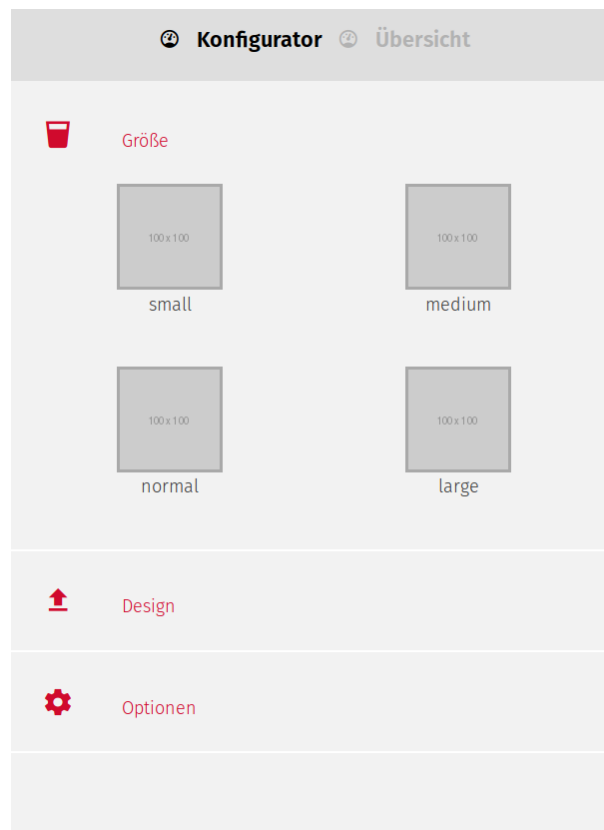


Abbildung 5.4: Sidebar der Angular-Anwendung

`<gc-upload>`, und `<gc-options>` sind dann sozusagen der Content. Die *SizeComponent* und *OptionsComponent* sind einfache `html`-Snippets. Die *UploadComponent* ist wegen dem Upload-Vorgang des Designs etwas komplexer und wird später noch genauer beschrieben.

## 5.5 3D Szene mit Model

**Szene erstellen** Die *SceneComponent* wurde schon angelegt. Nun muss noch die Funktionalität programmiert werden. In unserem Template ist alles was wir brauchen ein `<canvas>`-Element, in dem dann später die ganze 3D Szene dargestellt wird. Das erstellte `<canvas>`-Element ist ein einfaches DOM-Element im Template der *SceneComponent*. Zum Erstellen der 3D Szene und dem Laden der 3D-Models wird ein Service verwendet. Dieser Angular Baustein kann auch über die CLI angelegt werden. Der Service hat nun eine `createScene()` Funktion, die eine Scene mit Belichtung und Kamera erzeugt. Ihr wird als Parameter das `<canvas>`-Element aus der getter-Methode von *SceneComponent* übergeben.

Listing 5.5: Erstellung eines ElementRef für Canvas

```
|| (a)ViewChild('canvas')
```

```
private canvasRef: ElementRef;

private get canvas(): HTMLCanvasElement {
    return this.canvasRef.nativeElement;
}
```

Mithilfe des `@ViewChild`-Dekorators können wir eine Referenz eines DOM-Elements erzeugen. Dazu muss `ElementRef` aus dem `core`-Module importiert werden. Dadurch ist es uns nun möglich der `createScene()`-Funktion mitzuteilen, in welchem DOM-Element die Szene aufgebaut werden soll. Für den Konfigurator wurden verschiedene Belichtungen verwendet, die ein optimales Licht erzeugen. Dabei handelt es sich um ein *AmbientLight*, *KeyLight*, *FillLight* und *BackLight*. Die Kamera hat verschiedene Werte zugewiesen bekommen, damit sie in der richtigen Perspektive den Becher später darstellt.

**Model laden** Nun fehlt es noch an Leben in der Szene. Als 3D Model werden testweise Becher verwendet, die in etwa den späteren Mehrwegbechern entsprechen. Die 3D Models sind einfach `.obj`-Dateien, die aus 3D Programmen exportiert werden können. Mithilfe des *OBJ-Loaders* kann man nun die Datei in die Szene laden. Jedoch stößt man da schnell auf das Problem, das die Loader nicht standardmäßig implementiert sind. Abhilfe schafft hier das *npm*-Module *full-three*<sup>5</sup>. Es beinhaltet die komplette Three.js Bibliothek mit allen Beispielen und Loadern. Nachdem also die Szene mit Kamera und Belichtung steht, wird die `.obj`-Datei geladen.

Listing 5.6: Laden des Models mit dem OBJLoader

```
const loader = new THREE.OBJLoader();
loader.load('assets/model/test.obj', this.onModelLoadingCompleted);
```

In der Funktion `onModelLoadingCompleted` wird dann der Renderer ausgeführt, die das Model zur Szene hinzufügt und anschließend alles rendert. Jetzt ist die 3D Ansicht schon mal mit Leben gefüllt, der nackte Becher angezeigt wird (siehe Abbildung 5.5).

**Auswahl der Größe** Unter dem Menüpunkt „Größe“ soll der Benutzer eine der vier Größen wählen. Standardmäßig ist von Beginn an der 0,2 Liter Becher gewählt und ist von Anfang an sichtbar. Wie genau der `DataService` funktioniert, wird später noch einmal genauer beleuchtet. Zunächst werden in der `<gc-size>`-Komponente die verschiedenen Auswahlmöglichkeiten dargestellt. Mit der Funktion `reloadModel()` wird dem `SceneService` mitgeteilt, das der Benutzer eine Bechergröße gewählt hat. Dabei kommt wieder ein `@HostListener` zum Einsatz. In diesem Fall wird dieser mit dem `(click)`-Ereignis gebunden und führt dann die Funktion aus.

Listing 5.7: Ausschnitt der `reloadModel()`-Funktion

```
this.scene.remove(currentModel);
```

---

<sup>5</sup><https://www.npmjs.com/package/three-full>



**Abbildung 5.5:** 3D Model mit Three.js geladen

```
const loader = new THREE.OBJLoader();  
const file = 'assets/model/' + model + '.obj';  
loader.load(file, this.onModelLoadingCompleted);  
  
this.controls.update();  
this.render();
```

In der Funktion wird als erstes das aktuelle Model entfernt. Danach wird geprüft, welche Größe der User ausgewählt hat und dementsprechend wird das neue Model geladen. Anschließend muss die Szene neu gerendert werden.

**Controlling** Der Becher soll in 3D angezeigt werden und soll rotiert werden können. In Three.js wird dafür *OrbitControl* verwendet. Diese Klasse liefert verschiedene Werte mit, die verschiedene Einstellungen ermöglichen. Beispielsweise kann man die Rotationsgeschwindigkeit einstellen. Wenn man den Becher im Extremwinkel betrachtet oder zu weit hineinzoomt, kann der Becher bzw. dessen Texture unscharf aussehen oder nicht realistisch dargestellt sein. Deshalb wird über die Controls auch ein maximaler Zoom und ein maximaler Betrachtungswinkel festgesetzt.

**Responsive** Die Ansicht sollte auf allen Displaygrößen darstellbar sein. Dafür sorgt wieder ein `@HostListener`. Dieser macht nichts anderes als die Fenstergröße zu überprüfen und das `<canvas>`-Element anschließend an den Container anzupassen. Anders als die vorherigen `@HostListener` überprüft dieser das Fenster(*windows*) und nicht die Komponente. Falls sich die Fenstergröße ändert, wird der Szene über die Funktion `getAspectRatio()` die aktuelle Größe übermittelt. Dann wird die Szene aktualisiert und neu gerendert. Damit ist

verhindert, das zum Beispiel auf kleineren Geräten der Becher abgeschnitten dargestellt wird oder ähnliches.

### 5.6 Design Rendering

Jetzt geht es um das Hauptproblem der Arbeit: Auf dem Becher muss das Design des Kunden gerendert werden. Zunächst wird zum Testen ein Design erstellt, so wie es der Kunde hochladen würde. Nach verschiedenen Tests, das Design direkt auf die Außenseite des Bechers als Textur zu rendern, wurde eine besser Lösung gefunden.

**3D Zylinder** Die Bibliothek Three.js kann Geometrien erzeugen. Das heißt, man muss kein 3D Model aus einer Datei laden, sondern kann im Code eine geometrische Form erzeugen. Genau das wird für das Design des Bechers genutzt. Der Becher hat die Form eines Zylinders. Innerhalb der `onModelLoadingCompleted`-Funktion wird nun eine *CylinderBufferGeometry*<sup>6</sup> erstellt. Im Grunde umhüllt das Design den Becher wie ein Aufkleber.

Listing 5.8: Erstellung der Zylinder-Geometrie mittels Three.js

```
const geometry = new THREE.CylinderBufferGeometry(<radiusOben>, <  
    radiusUnten>, <height>, <radialSegmente>, <heightSegmente>, <  
    offenesEnde>, <lenght>);
```

Durch die Parameter ist es möglich, den Zylinder genau an die Maße des Bechers anzupassen. Er liegt ganz eng auf dem Becher, eben wie ein Aufkleber. Mit dem Parameter `<offenesEnde>` wird festgelegt, das der Cylinder hohl ist und `<lenght>` stellt sicher, das der Zylinder 360 Grad im den Becher dargestellt wird.<sup>7</sup> Die anderen Parameter beschreiben den Radius und die Höhe des Zylinders. Im *ThreeService* sind die Parameter der *CylinderBufferGeometry* in Variablen für jeden der vier Becher definiert. Bevor die Geometrie erstellt wird, muss nur das aktuelle Model geprüft werden. Dann müssen die passenden Zylinder-Parameter für die aktuelle Bechergröße geladen werden. Somit umhüllt der Zylinder immer den ganzen Becher, egal welche Bechergröße gewählt ist.

Jetzt kann das Design des Kunden als Texture auf den Zylinder gepackt werden. Das praktische ist, das sich das Design an die Maße des Zylinders anpasst. Wenn das Design in der richtigen Auflösung hochgeladen wurde, sieht es somit realistisch aus<sup>8</sup>. Aufgrund der kubischen Form des Zylinders wird das Design minimal gestaucht, genau wie beim Druck. Da der Upload des Designs noch bis hierhin noch nicht implementiert ist, wird mit einem statischen Design gearbeitet. In Kapitel 5.9 wird sich diese Arbeit mit dem Upload-Vorgang beschäftigen. Solange allerdings kein Design als Texture hochgeladen wurde, wird der erstellte Zylinder in der Scene ausgeblendet. Es würde keinen Sinn machen einen blanken Zylinder um den Becher anzuzeigen. Die 3D Darstellung des Bechers mit Design funktioniert aber grundsätzlich schon, wie man in Abbildung 5.6 sehen kann.

---

<sup>6</sup><https://threejs.org/docs/#api/en/geometries/CylinderBufferGeometry>

<sup>7</sup>Um das zu erreichen setzt man einfach den Wert auf  $2\pi$

<sup>8</sup>Im Upload-Vorgang wird der Benutzer bei falscher Auflösung des Designs darauf hingewiesen (siehe Kapitel 5.9).



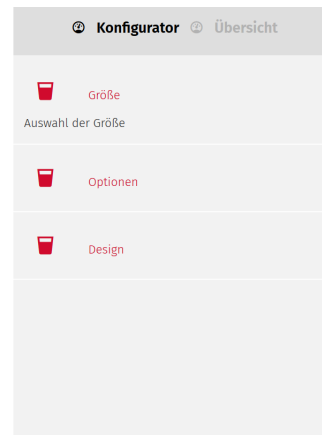


Abbildung 5.6: Darstellung des Designs auf dem Becher mit Zylinder-Geometrie

## 5.7 Mobile Umsetzung

Auf Tablets (in der Portraitansicht) ist es praktisch das Menü unterhalb der 3D Ansicht darzustellen. Das lässt sich ganz einfach mit den Column-Klassen von Bootstrap realisieren. Interessanter wird die Darstellung auf Smartphones.

Das Off-Canvas Menu ist vom Design nicht anders als die Desktop-Variante. Es wird lediglich verschoben. Das bedeutet, wenn ein Benutzer auf dem Smartphone den Konfigurator öffnet, sieht er zunächst nur den 3D Becher. Er kann dann das Off-Canvas-Menu über ein Hamburger-Menu-Button öffnen. Das ganze ist mit einer JavaScript Funktion realisiert, die einfach das Menü aus dem sichtbaren Bereich verschiebt und sie erst dann sichtbar werden lässt, wenn der Hamburger-Menu-Button gedrückt wird. Über css werden die SceneComponent sowie die SidebarComponent noch so angepasst, dass sie immer 100% des Displays einnehmen. Außerdem werden die Schriftgrößen etwas größer gemacht. Der Hamburger-Button ist ausschließlich auf Smartphones sichtbar. Wenn dieser geklickt wird, greift die Funktion `showSidebar()` bzw. `hideSidebar()`.

Listing 5.9: Ausschnitt der ShowSidebar()-Funktion

```
|| document.getElementById("mySidenav").style.width = 100%;
```

Soll die Sidebar angezeigt werden, wird die Höhe auf 100% gesetzt. Soll sich geschlossen werden, wird die Breite auf 0 gesetzt. Damit ist eine sehr simple Lösung einer mobilen Sidebar implementiert.

Die 3D Szene ist, wie schon zuvor erwähnt, auch responsive. Die OrbitControls von Three.js liefern auch eine Kompatibilität für Touch-Eingaben mit. So kann auf mobilen Geräten der Becher via Touch rotiert werden.

### 5.8 Daten-Service

Der DataService hat eine Aufgabe: Alle Daten des konfigurierten Bechers zu speichern und sie bei Anfrage einer Komponente wieder zurückzugeben. Dabei ist es wichtig das der Service nur einmal gestartet wird und die Daten global speichert. Hierbei handelt es sich also um ein Singleton. In Angular einen Service zu implementieren, ist recht unkompliziert. Analog zum *SceneService* wird dieser auch über Angular CLI anzulegen. Anders als der *SceneService*, welcher ausschließlich von der *SceneComponent* verwendet wird, soll dieser global zugänglich sein. Dabei speichert er folgende Daten:

**selectedImage** Speichert die png-Datei, welche der User hochgeladen hat

**state** Hat den Wert *true* wenn ein Bild ausgewählt wurde

**designName** Speichert den Namen der Bilddatei

**designSize** Speichert die Größe der Bilddatei

**count** Speichert die Anzahl der Becher, die der Benutzer später drucken lassen möchte

**size** Speichert die vom Benutzer gewählte Bechergröße

**stroke** Speichert, ob der Eichstrich mitgedruckt werden soll (Optionale Auswahl)

**price** Zeigt den aktuellen Staffelpreis abhängig der gewählten Anzahl.

**model** Speichert den Pfad zur .obj-Datei, welche gerendert werden soll

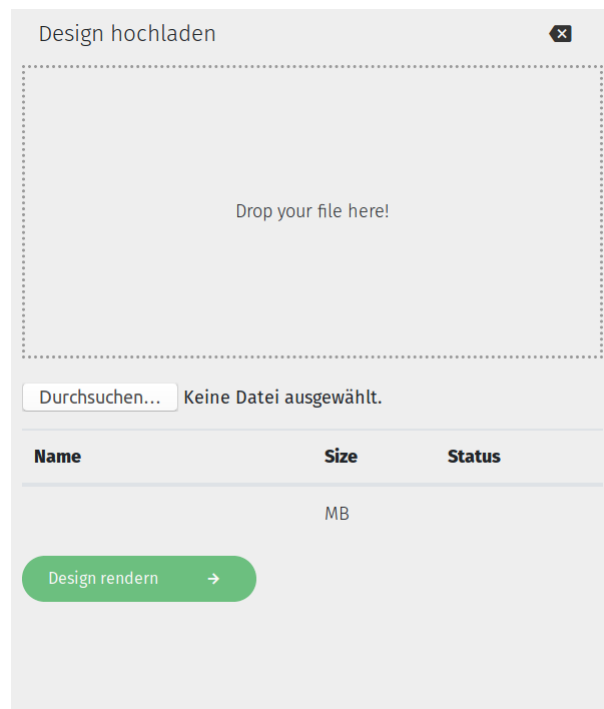
Die Eigenschaften *designName*, *count*, *size*, *stroke* und *price* werden in der Tabelle *OverviewComponent* als Bindung angezeigt. Somit ist immer die aktuelle Auswahl des Benutzers in der Tabelle. Der *DataService* ist somit ein Speicherort aller Informationen über den ausgewählten Becher und das verwendete Design. Es ist durchaus denkbar den Service später zu erweitern. Man könnte die Informationen auf einem Server sichern, sodass der User zu späteren Zeiten auf seine Konfiguration zurückgreifen kann. Angular bietet dafür eine einfache Realisierung über das *HttpClientModule*.

Auch der *SceneService* nutzt den *DataService*. Zunächst bekommt der *SceneService*, wie schon erwähnt, die Information, welche Größe gewählt wurde. Je nach dem welchen Wert *size* dann hat, ändert sich das 3D-Model. Ähnlich verhält es sich mit dem Design, was als Textur auf dem Becher angezeigt werden soll. Im nächsten Abschnitt wird genauer beschrieben, wie ein Design hochgeladen werden kann, und welche Rolle der *DataService* dort spielt.

### 5.9 Upload-Vorgang

Angenommen ein Benutzer hat sich für eine Größe entschieden und möchte nun sein Design auswählen. In der Sidebar kann über einen Button die *UploadComponent* geöffnet werden. Der User wird hier auch noch einmal darauf hingewiesen, die Druckvorgaben zu beachten.

**Route zum Upload** An dieser Stelle der Anwendung wird das *Routing* verwendet. Normalerweise befinden wir uns in dem Konfigurator unter dem root-Pfad. Beim Upload ändert sich der Pfad zu `localhost/upload`. Unter dieser Route ist definiert, dass die *UploadComponent* angezeigt werden soll und nicht die *SidebarComponent*. Hier kann der Benutzer jetzt sein Design auswählen (siehe Abbildung). Eigentlich ist der Begriff „Upload“ nicht ganz zutreffend,



**Abbildung 5.7:** Upload-Komponente der Angular-Anwendung zum auswählen des Designs

da das Design nicht wirklich auf einen Server hochgeladen wird, sondern nur in dem *DataService* gespeichert wird. Trotzdem ist klar, was die Funktion dieser Komponente ist. Der Benutzer hat zwei Möglichkeiten sein Design auszuwählen.

**Drag & Drop oder Select** Eine bekannte Methode um Dateien hochzuladen ist die Drag & Drop Methode. Man muss einfach die Datei in den dafür vorgesehenen Bereich ziehen und die Datei wird erkannt. Die zweite Möglichkeit ist es ganz klassisch über ein Input-Feld eine Datei aus dem Verzeichnis auszuwählen. Egal, für welche Methode der Benutzer sich entscheidet, passiert im Grunde das selbe. Ein Unterschied besteht darin, dass verschiedene Ereignisse gebunden werden. Bei der klassischen Input-Feld Variante wird über das Event *Change* überprüft, ob eine Datei ausgewählt wurde. Bei der Drag & Drop Variante wird das Event *drop* verwendet. Außerdem wird zusätzlich *dragover* und *dragleave* ausgeführt, wenn eine Datei in die Dropzone gezogen wird oder hinausgezogen wird. Für eine bessere Usability verändern sie die Farbe des Rahmens. Somit bekommt der User ein Feedback, wenn er seine Datei an die richtige Stelle gezogen hat.

In der *UploadComponent* sind beide Upload-Vorgänge mit der Direktiven umgesetzt. In Diesen sind Dekoratoren implementiert, welche die eben erwähnten Ereignisse über *@HostListener* binden. Für den Upload-Vorgang in diesem Konfigurator wird ein benutzerdefiniertes Ereignis benötigt. Um das zu erstellen verwendet man das *EventEmitter*-Objekt.

Listing 5.10: Umsetzung des Event-Emitter-Objekts

```
|| private filesChangeEmitter: EventEmitter<File> = new EventEmitter();
```

Zusätzlich muss der Typ der Objekte explizit festgelegt werden. In diesem Fall wird ein *File*(einfache Datei) emittiert, jedes Mal, wenn der Benutzer eine Datei ausgewählt hat. Da der User nur ein Bild hochladen darf, ist dies nicht als Array implementiert. Nun können wir den *EventEmitter* in *@HostListener* verwenden. Es muss aber noch das Ereignis mit der übergeordneten HTML-Komponente gebunden werden. Dazu benötigt man den *@Output* Dekorator für den *EventEmitter*. Auf diese Weise ist es möglich zu überprüfen, ob sich die ausgewählte Datei geändert hat.

Listing 5.11: Drop-Zone Element mit der EventEmitter-Funktion

```
|| <div gcDragDrop (filesChangeEmitter)="onFilesChange($event)">
```

Wie man im Beispiel sieht, wird jedes Mal, wenn eine Datei in die Drop-Zone geschoben wird, die Funktion *onFilesChange(\$event)* ausgeführt. Der Parameter *\$event* ist immer ausgewählte Datei. Die Methode kann jetzt mit diesem *File* arbeiten.

In der *UploadComponent* und auch an anderen Stellen der Anwendung soll der Dateiname und die Größe der Datei ausgegeben werden. Es ist nun möglich aus der übergebenen Datei beides auszulesen und in dem *DataService* zu speichern. Nun muss noch die *reloadTexture()*-Funktion ausgeführt werden und die Datei übergeben werden. In *Three.js* werden Texturen mit dem *TextureLoader* geladen bzw. gemappt. Anschließend kann man sie zur Szene hinzufügen. Allerdings kann der Loader nur URLs oder Base64-Objekte entgegen nehmen. Auf den Pfad des Bildes können wir aus Sicherheitsgründen nicht zugreifen. Deshalb muss das *File*-Objekt in ein Base64-Objekt konvertiert werden, bevor man die Textur laden kann. Das konvertierte Objekt wird dann in *selectedImage* im *DataService* gespeichert.

Listing 5.12: Base64 Methode

```
|| this.getBase64(file).then(  
||     data => this.DataServ.selectedImage = data  
|| );
```

Ist nun eine gültige *.png*-Datei ausgewählt worden, wird wie in Abbildung zu sehen Name und Dateigröße angezeigt. Jetzt ist auch der *Design rendernButton* aktiv und das Design kann mit der *reloadTexture*-Funktion gerendert werden.

**reloadTexture** Im Grunde ist diese Funktion nahezu identisch zu *reloadModel()*-Funktion. Ein Teil der Funktion ist schon bekannt. Wir haben schon gesehen wie diese Funktion den

Zylinder für das Design um den Becher erstellt. Als Erstes löscht die Funktion eine Texture, die eventuell schon zuvor gerendert wurde. Nun bekommt die Funktion das Design des Users als Parameter übergeben und kann es als Texture gemappt werden.

Listing 5.13: TextureLoader Umsetzung

```
const loader = new THREE.TextureLoader();  
const material = new THREE.MeshBasicMaterial();  
material.map = loader.load(image);
```

Um das ganze abzuschließend wird nun noch die Textur mit der Geometrie zusammengeführt und schließlich zu Szene hinzugefügt. Nachdem dann die Szene neu gerendert wurde, sollte das Design des Benutzers richtig auf dem Becher angezeigt werden.

Damit ist die Realisierung abgeschlossen. Die Funktionen wurden weitestgehend alle implementiert und müssen jetzt noch geprüft werden.



## Kapitel 6

# Testphase & Ergebnisse

Das folgende Kapitel befasst sich mit der Testumgebung von Angular. Es werden ein paar Test Unit-Tests und e2e-Tests durchgeführt. Dabei wird auch auf verschiedenen Geräteklassen getestet. Anschließend werden die Ergebnisse der Test zusammengefasst.

### 6.1 Testumgebung

In Angular gibt es grundsätzlich zwei Optionen, um die Anwendung zu testen. Als Erstes werden sogenannte *Unit-Tests* durchgeführt. Angular bringt von Haus aus eine vorkonfigurierte Testumgebung und nutzt dazu das Framework *Jasmine* und ausgeführt werden sie mit *Karma*. Theoretisch kann man diese *Unit-Tests* auch schon während der Entwicklung implementieren. Man kann damit sicherstellen, dass einzelne Bausteine der Anwendung auch wirklich das gewünschte Ergebnis erzeugen. In dieser Arbeit werden allerdings keine Tests im großen Umfang ausgeführt. Es wird sich nur auf das wesentliche konzentriert, da Jasmine sehr viele Testfunktionen mit sich bringt. Um den vollen Funktionsumfang auszuschöpfen, bedarf es etwas mehr an Zeit.

Nach den *Unit-Tests* werden die *End-to-End-Tests* durchgeführt. Diese sind noch etwas interessanter für den Konfigurator. Hiermit werden ganze User-Interaktionen ausgeführt und man kann den Konfigurator so auf Usability und Performance testen. Nebenbei erfährt man natürlich auch so, ob die Anwendung das tut, was sie soll. Diese *End-to-End-Tests* werden in Angular ebenfalls mit *Jasmine* ausgeführt.

#### 6.1.1 Unit Tests

Eine Installation oder Konfiguration ist nicht notwendig<sup>1</sup>, da Angular beim Initialisieren des Projektes bzw. beim Generieren der Angular-Bausteine schon alles bereitgestellt hat. Jeder Baustein der Anwendung hat eine `.spec.ts`-Datei, welche die Tests beinhalten. Wenn man mit `ng test` den Test startet, holt sich Angular aus der `test.ts`-Datei den *Context* der

---

<sup>1</sup>Es ist durchaus möglich mit anderen Frameworks die Anwendung zu testen. Allerdings ist es für den Konfigurator nicht notwendig und mit den Standardeinstellungen am einfachsten umzusetzen.

getestet werden soll, also alle Test-Dateien der verschiedenen Bausteine. Im Folgenden wird anhand der `SizeComponent` gezeigt, wie die Unittest aufgebaut wurden.

Listing 6.1: `size.component.spec.ts` Datei

```
import {async, ComponentFixture, TestBed} from 'angular/core/testing';

import { SizeComponent } from './size.component';

describe('SizeComponent', () => {
  let component: SizeComponent;
  let fixture: ComponentFixture<SizeComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ SizeComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(SizeComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Zunächst muss aus dem Test-Modul das `TestBed` (also die Test-Umgebung) aus dem `angular/core` importiert werden. Genauso braucht man natürlich die zu testende Komponente und muss diese auch importieren. Falls man in einer Komponente noch weitere Kindskomponenten, Direktiven oder ähnliches verwendet, müssen diese auch importiert werden. Anschließend werden mit der `configureTestingModule()`-Methode alle Bausteine deklariert und danach kompiliert. Die eigentlichen Tests befinden sich in den `it()`-Funktionen. Der erste Ausdruck ist einfach eine Beschreibung, um später die Tests in den Ergebnissen besser zuordnen zu können. Danach können Werte und Methoden überprüft werden. Der erste Test prüft lediglich mit `toBeTruthy()`, ob die Komponente erzeugt wurde. Mit der `expect()`-Funktion wird sozusagen etwas geprüft. Schlägt die Überprüfung fehl, wird in den Ergebnissen ein Error angegeben.

### 6.1.2 End-to-End Tests

Zunächst muss aus dem Test-Modul das `TestBed` (also die Test-Umgebung) aus dem `angular/core` importiert werden. Genauso braucht man natürlich die zu testende Komponente und muss diese auch importieren. Falls man in einer Komponente noch weitere Kindskomponenten, Direktiven oder ähnliches verwendet, müssen diese auch importiert werden. Anschließend werden mit der `configureTestingModule()`-Methode alle Bausteine deklariert und danach



kompiliert. Die eigentlichen Tests befinden sich in den `it()`-Funktionen. Der erste Ausdruck ist einfach eine Beschreibung, um später die Tests in den Ergebnissen besser zuordnen zu können. Danach können Werte und Methoden überprüft werden. Der erste Test prüft lediglich mit `toBeTruthy()`, ob die Komponente erzeugt wurde. Mit der `expect()`-Funktion wird sozusagen etwas geprüft. Schlägt die Überprüfung fehl, wird in den Ergebnissen ein Error angegeben.

## **6.2 Ergebnisse**

**3D Darstellung**

**Menüführung inkl. Routing**

**Upload und reload texture**

**Sonstiges**



## **Kapitel 7**

# **Zusammenfassung**

Unter funktionalen Anforderungen versteht man konkrete Funktionalitäten, welche die Anwendung leisten soll. Die Anwendung lässt sich in mehrere Module aufteilen, die dem User zu Verfügung stehen. Im Folgenden werden die funktionalen Anforderungen erläutert.



# Anhang A

## Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodo consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis.

At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, At accusam aliquyam diam diam dolore dolores duo eirmod eos erat, et nonumy sed tempor et et invidunt justo labore Stet clita ea et gubergren, kasd magna no rebum. sanctus sea sed takimata ut vero voluptua.

est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat.

Consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

# Glossar

<b>CLI</b>	Command Line Interface
<b>SEMVER</b>	Semantische Versionierung
<b>UML</b>	Unified Modeling Language





# Literaturverzeichnis

- [Bö] BÖHM, Robin: *Angular 4 im Überblick - Die wichtigsten Neuerungen - Angular.DE*. <https://angular.de/artikel/angular-4-neuerungen/>
- [Bah] BAHOR, Senad: *HTML5/WebGL vs Flash in 3D Visualisation*. [http://old.cescg.org/CESCG-2013/papers/Bahor-HTML5WebGL\\_vs\\_Flash\\_in\\_3D\\_Visualisation.pdf](http://old.cescg.org/CESCG-2013/papers/Bahor-HTML5WebGL_vs_Flash_in_3D_Visualisation.pdf)
- [DEI] DIN-EN-ISO-9241-210: *Ergonomie der Mensch-System-Interaktion - Teil 210: Prozess zur Gestaltung gebrauchstauglicher interaktiver Systeme*. <https://www.din.de/de/mitwirken/normenausschuesse/naerg/wdc-beuth:din21:135399380>
- [Dev] DEVERIA, Alexis: *Can I use... Support tables for HTML5, CSS3, etc.* <https://caniuse.com/#feat=webgl>
- [Dom] DOMIN, Andreas: *Was ist eigentlich eine Single-Page-Webanwendung?* <https://t3n.de/news/single-page-webanwendung-1023623/>
- [Fly] FLYERALARM: *Mehrwegbecher günstig und schnell bei FLYERALARM*. <https://www.flyeralarm.com/de/shop/configurator/index/id/6899/mehrwegbecher.html>
- [Gar] GARDNER, Lyza D.: *What We Mean When We Say "responsive"*. <http://alistapart.com/column/what-we-mean-when-we-say-responsive>
- [Giz] GIZYCKI, Vittoria von: *Usability — nutzerfreundliches Web-Design*. [http://dx.doi.org/10.1007/978-3-642-56377-5\\_1](http://dx.doi.org/10.1007/978-3-642-56377-5_1). In: BEIER, Markus (Hrsg.) ; GIZYCKI, Vittoria von (Hrsg.): *Usability: Nutzerfreundliches Web-Design*. Springer Berlin Heidelberg (X.media.press). – DOI 10.1007/978-3-642-56377-5\_1. – – ISBN978 – –3 – –642 – –56377 – –5,1 – –17
- [Kö] KÖGEL, Paul: *React Einsteiger Tutorial - ReactJS.DE*. <https://reactjs.de/artikel/react-tutorial-deutsch/>
- [Mar] MARCOTTE, Ethan: *Responsive Web Design*. <http://alistapart.com/article/responsive-web-design>

- [MM] MOREAU-MATHIS, Julien: *Babylon.js Essentials*. Packt Publishing Ltd. – ISBN 978-1-78588-616-4. – S. 4-6
- [Ün] ÜNLÜ, Saban: *Angular Grundkurs 1: Das Ökosystem*. Videokurs. <https://www.lynda.com/Angular-tutorials/Angular-Grundkurs-1-Okosystem/561502-2.html> Angular Grundkurs 1: Das Ökosystem
- [Pon] PONIROS, Nikolas: *Angular für Dummies*. John Wiley & Sons. – ISBN 978-3-527-81311-7. – Google-Books-ID: AUiHDwAAQBAJ
- [RF] RICHTER, Michael ; FLÜCKIGER, Markus D.: Usability und UX. [http://dx.doi.org/10.1007/978-3-662-49828-6\\_2](http://dx.doi.org/10.1007/978-3-662-49828-6_2). In: RICHTER, Michael (Hrsg.) ; FLÜCKIGER, Markus D. (Hrsg.): *Usability und UX kompakt: Produkte für Menschen*. Springer Berlin Heidelberg (IT kompakt). – DOI 10.1007/978-3-662-49828-6\_2. – – ISBN 978 – – 3 – – 662 – – 49828 – – 6, 7 – – 19
- [spr] SPREADSHIRT: *Tassen bedrucken, Kaffeebecher & Fototassen* | Spreadshirt. <https://www.spreadshirt.de/gestalten/tassen>
- [Sta] STACKOVERFLOW: *Die Stack Overflow Entwicklerumfrage 2018* | Stack Overflow. <https://www.stackoverflowbusiness.com/de/talent/ressourcen/die-stack-overflow-entwicklerumfrage-2018>
- [Ste] STEYER, Manfred: *Die Ruhe vor dem Sturm: Neuerungen in Angular 7 und Ausblick*. <https://entwickler.de/online/javascript/angular-7-ueberblick-579865876.html>
- [WMKH] WOIWODE, Gregor ; MALCHER, Ferdinand ; KOPPENHAGEN, Danny ; HOPPE, Johannes: *Angular: Grundlagen, fortgeschrittene Techniken und Best Practices mit TypeScript - ab Angular 4, inklusive NativeScript und Redux*. dpunkt.verlag. – ISBN 978-3-96088-206-0. – Kapitel 6