ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Entwicklung und Realisierung von Konzepten zur Unterstützung von Fachexperten bei der Erstellung digitaler Fragebögen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Lenard Funk
lenard.funk@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Johannes Schobel

2018

Fassung 13. Juni 2018

Satz: PDF-LaTeX 2$_\varepsilon$

Abstract

Die Durchführung von Studien macht in der Forschung einen großen Teil der Arbeit aus. Dabei steht die umfangreiche Erfassung hochdimensionaler Daten an vorderster Stelle, welche heutzutage in zunehmendem Maße auf digitaler Basis erfolgt. Die vorliegende Arbeit steht im Kontext eines Projekts, welches die Erstellung komplexer, problemspezifischer Fragebögen erlaubt, die Datenerfassung einheitlich durchführt sowie deren Auswertung vereinfacht. Der Entwurf eines Fragebogens erfolgt dabei innerhalb einer Konfigurator-Anwendung über ein graphisches Modell, wobei die einzelnen Fragen jeweils durch abstrakte Elemente repräsentiert und konfiguriert werden. Die Visualisierung dieser Fragebogen-Elemente erfolgt erst zu einem späteren Zeitpunkt in einer mobilen Endanwendung, welche zum Ausfüllen der Fragebögen dient. Durch die Abstrahierung im Erstellungsprozess erhält der Fachexperte jedoch keine direkte Vorstellung von der tatsächlichen Darstellung des Fragebogens auf mobilen Endgeräten. Für ein effizientes Arbeiten mit dem Konfigurator würden daher fundierte Praxis-Erfahrungen mit dem Gesamtsystem benötigt werden. Aus diesem Grund wird im Rahmen dieser Arbeit eine Vorschau-Funktion entwickelt, welche den Erstellungsprozess von Fragebögen durch Visualisierung der angelegten Elemente unterstützt. Die Vorschau bietet eine Auswahl von Simulationen verschiedener mobiler Geräte, in welchen der Fragebogen angezeigt wird. Daraus ergibt sich eine realitätsgetreue Voransicht des späteren Endergebnisses, was zu einer besseren und einfacheren Bedienbarkeit der Konfigurator-Anwendung beiträgt. In diesem Zusammenhang werden Konzepte vorgestellt und implementiert, welche hohe Flexibilität in der Darstellung der Vorschau sowie eine einfache Erweiterbarkeit hinsichtlich neuer Komponenten bieten.

Inhaltsverzeichnis

1

Einleitung

Diese Abschlussarbeit steht im Kontext eines Projektes zur Entwicklung eines Frameworks für die Erstellung komplexer Fragebögen. Damit soll Fachexperten die Erfassung problemspezifischer Daten erleichtert werden.

Im Zentrum des Projekts steht eine Konfigurator-Anwendung, mit deren Hilfe Fragebögen in effizienter und flexibler Weise konstruiert werden können. Dem Benutzer des Konfigurators wird mithilfe eines prozessorientierten Modells ermöglicht, komplexe Fragebogen-Strukturen zu erzeugen, welche eine effiziente und intuitive Datenerfassung erlauben. Letztere erfolgt in mobilen Anwendungen, welche die Fragebögen seitenweise darstellen. Durch die Gliederung der Fragebögen in Seiten wird ein Bezug zur noch immer etablierten papierbasierten Datenerfassung hergestellt.

Durch spezielle Elemente wird (bereits zum Zeitpunkt des Ausfüllens eines Fragebogens) die dynamische Auswertung der Antworten vorangegangener Fragen ermöglicht, wodurch die Datenerhebung für den Bearbeiter des Fragebogens individualisiert werden kann.

Die Erstellung des Fragebogen-Inhalts in der Konfigurator-Anwendung erfolgt mithilfe abstrakter Elemente, die zur Bestimmung der Abfolge in einem Flussdiagramm angeordnet werden. Den Elementen kann jeweils der textuelle oder graphische Inhalt in einem separaten Bereich zugeordnet werden.

Durch diese Abstrahierung entsteht jedoch das Problem, dass der Fachexperte beim Erstellungsprozess kein unmittelbares Feedback erhält, wie der Fragebogen später in der in der mobilen Endanwendung (d.h. beim Nutzer) aussieht.

Dies wirkt somit der angestrebten Intuitivität und Einfachheit der Fragebogen-Erstellung entgegen. Aus Sicht der Informatik zeugt diese Eigenschaft zudem von schlechter Qualität bezüglich der Benutzerfreundlichkeit, wenn ein Fragebogen gewissermaßen „blind" erstellt werden muss. Denn so muss der Benutzer den entwickelten Fragebogen zuerst veröffentlichen und auf einem mobilen Endgerät installieren, bevor er dort schließlich das Ergebnis beurteilen kann. Noch schwieriger gestaltet sich folglich die Anpassung des Fragebogens für verschiedene Geräte bzw. Bildschirmgrößen, da dieser hierfür auf allen relevanten Geräten installiert werden muss. Die Fragebogen-Erstellung setzt damit bereits Expertenwissen über das gesamte System voraus und greift zudem auf andere Teilanwendungen über, was einen stark erhöhten Aufwand impliziert.

Eine effiziente Methode das beschriebene Problem zu lösen, ist eine „Vorschau-Funktion", wie sie beispielsweise aus diversen Editoren oder Konfiguratoren (siehe Kapitel 6) bekannt ist, bereitzustellen. In einer solchen Live-Vorschau können verschiedene mobile Geräte simuliert und dem Nutzer zur Auswahl gestellt werden. Der Fragebogen wird dann innerhalb des ausgewählten Geräterahmens dargestellt. Dadurch kann der Nutzer bereits während des Erstellungsprozesses das Endergebnis betrachten und den Fragebogen-Inhalt für die verschiedenen Bildschirmgrößen in effizienter Weise anpassen.

Aus IT-Sicht ist dabei der wichtigste Aspekt ein responsives Verhalten der Vorschau in allen Bereichen. Einerseits muss sich der Vorschaubereich an die Bildschirmgröße des Nutzers anpassen, um eine optimale Darstellung zu gewährleisten. Andererseits muss sich auch der Fragebogen dynamisch an den simulierten Geräterahmen anpassen, da unterschiedlich große Geräte zur Auswahl stehen und zur Laufzeit ausgetauscht werden können.

Ein weiteres Kriterium ist eine gute Erweiterbarkeit der Vorschau-Komponente. Denn diese soll sowohl in Bezug auf die im Konfigurator verfügbaren Fragebogen-Elemente als auch hinsichtlich aktueller mobiler Geräte ohne großen Aufwand auf dem neuesten Stand gehalten werden können.

Im Rahmen dieser Abschlussarbeit wird eine Vorschau-Funktion für die existierende Konfigurator-Anwendung implementiert, die verschiedene mobile Geräte simuliert, in

welchen der vom Benutzer erstellte Inhalt dargestellt wird. Dabei wird insbesondere auf die Realitätsnähe zur tatsächlichen Darstellung in den mobilen Endanwendungen geachtet. Auch die bereits im Gesamtprojekt entwickelte Mehrsprachigkeit der Fragebögen soll in die Vorschau-Funktion integriert werden. Damit sollen Fachexperten beim Entwurf eines Fragebogens noch besser unterstützt werden, sodass eine effiziente und intuitive Verwendung des Konfigurators gewährleistet werden kann.

Der weitere Verlauf der Arbeit ist wie folgt: Kapitel 2 liefert für die aktuelle Arbeit relevante Grundlagen zum Gesamtprojekt. In Kapitel 3 wird ein Konzept für die Implementierung der Vorschau-Funktion entwickelt und Kapitel 4 stellt die wichtigsten Ergebnisse dieser Abschlussarbeit vor. Als Ergänzung werden in Kapitel 5 Möglichkeiten zur Erweiterung der Vorschau hinsichtlich neuer Fragetypen und simulierter Geräte beschrieben. In Kapitel 6 werden verwandte Arbeiten vorgestellt und Kapitel 7 präsentiert abschließend eine Schlussfolgerung sowie einen Ausblick auf mögliche zukünftige Ansätze.

2

Grundlagen

In diesem Kapitel werden Grundlagen zum Gesamtprojekt bereitgestellt, die für die Einordnung und das bessere Verständnis der zu entwickelnden Vorschau-Funktion notwendig sind.

2.1 Projekt

Das Gesamtprojekt des Fragebogen-Systems wurde 2016 ins Leben gerufen. Es enthält eine Konfigurator-Anwendung, eine Server-Anwendung und Endanwendungen für mobile Geräte, welche jeweils separat voneinander entwickelt werden. Die Architektur des Gesamtsystems ist in Abbildung 2.1 dargelegt. Die linke Spalte zeigt dabei den Konfigurator [1], mithilfe dessen Fachexperten individuelle Fragebögen erstellen und veröffentlichen können. Die rechte Spalte stellt die mobilen Endanwendungen [2] dar, die zum Ausfüllen der Fragebögen dienen. Die mittlere Spalte repräsentiert schließlich die Web-Services der Server-Komponente [3], welche als Schnittstelle zwischen den anderen Komponenten des Systems dienen.

2.1.1 Idee und Ziel

Bisherige Anwendungen, die die Erstellung von Fragebögen erleichtern sollen, sind häufig formularbasiert aufgebaut, sodass Layout, Form und Inhalt im selben Schritt festgelegt werden. Da auch heutzutage noch viele Fragebögen in papierbasierter Form auszufüllen sind (z.B. im medizinischen Bereich), muss der Fragebogen seitenweise strukturiert und die Fragen dementsprechend gegliedert werden. Weiterhin sind be-

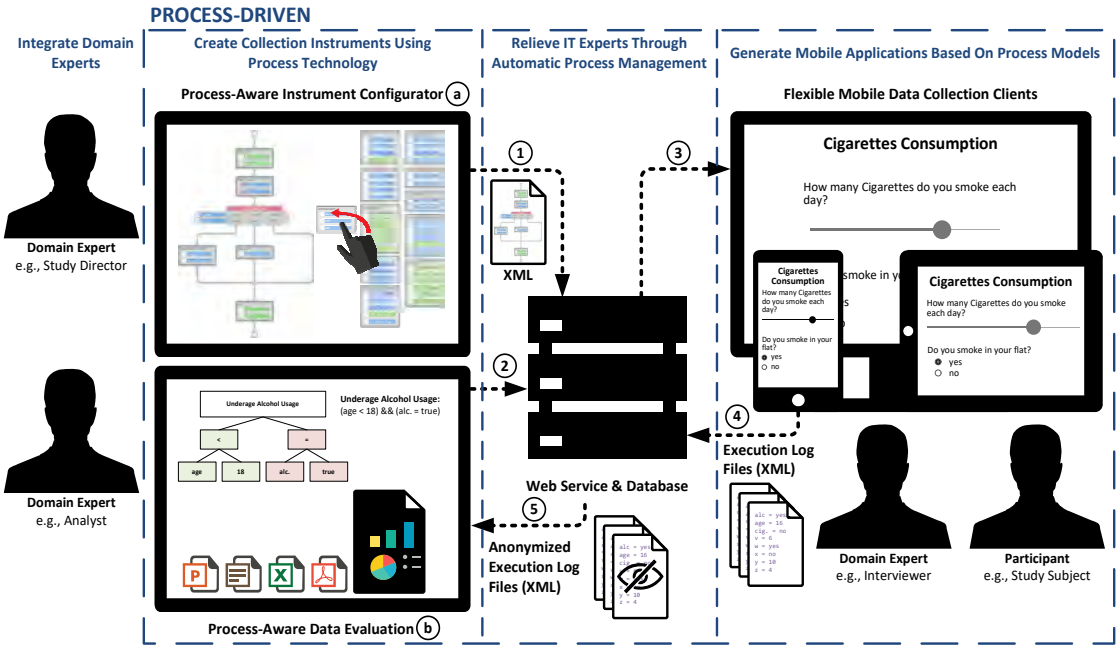


Abbildung 2.1: Gesamtarchitektur des Fragebogen-Systems [3]

stimmte Fragen nur unter bestimmten Bedingungen auszufüllen, zum Beispiel basierend auf der Antwort einer vorangehenden Frage. Dies muss somit textuell gekennzeichnet werden und der Bearbeiter muss diese Entscheidungen selbstständig nachvollziehen.

Der Fokus des Gesamtprojekts liegt insbesondere auf Fragebögen für psychologische und medizinische Studien. Das Ziel ist somit sowohl den Fragebogen-Ersteller als auch dessen Bearbeiter sich auf das für ihn Wesentliche konzentrieren lassen zu können und so ein einfacheres und effizienteres Arbeiten zu ermöglichen. Daher werden beim Design des Fragebogens abstrakte Elemente für die Fragen verwendet, denen dann der textuelle und/oder grafische Inhalt zugewiesen werden kann (siehe ⓐ in Abbildung 2.1 bzw. ① – ③ in Abbildung 2.2). Die Visualisierung der Fragen findet hierbei erst zum Zeitpunkt des Ausfüllens statt (rechte Spalte von Abbildung 2.1 bzw. ⓒ in Abbildung 2.2). Des Weiteren entstammen die Grundüberlegungen bei diesem Projekt einer prozessorientierten Sichtweise, ähnlich zur Workflow-Bildung bei ARISTAFLOW, einer Software-Lösung für Business Process Management [4]. Nach diesem Prinzip gibt es auch im Fragebogen Entscheidungselemente, welche die Antwort auf festgelegte Fragen dynamisch auswerten, wodurch dem Bearbeiter lediglich die für ihn relevanten Fragen

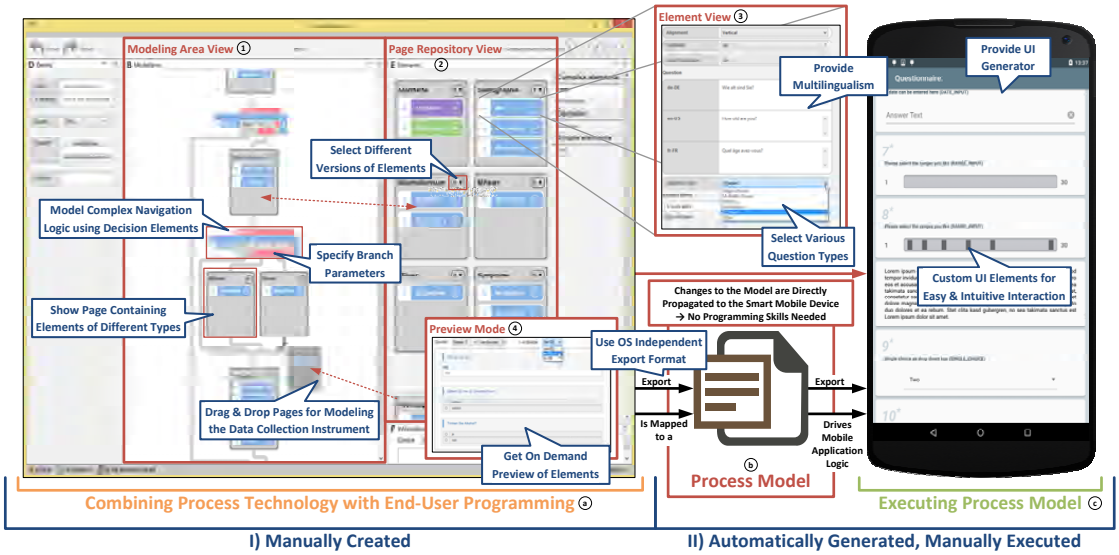


Abbildung 2.2: Ablauf der Modellierung des Fragebogens und dessen Export auf das Endgerät [3]

angezeigt und irrelevante Bereiche ausgeblendet werden können. Weitere Bestandteile des Konfigurators sowie seine Funktionsweise werden im Detail in [1] vorgestellt.

Die Abstraktion beim Design führt jedoch auch zu dem bereits in der Einleitung beschriebenen Problem, dass ein unerfahrener Nutzer des Konfigurators keine Vorstellung vom visuellen Ergebnis erhält und den Inhalt der Fragen somit nicht daran anpassen kann. Aus diesem Grund ist eine Vorschau-Funktion (④ in Abbildung 2.2) erforderlich, die im Rahmen der vorliegenden Abschlussarbeit entwickelt wird.

2.1.2 Weitere für die Vorschau relevante Features

Das Projekt bietet inzwischen zahlreiche Features und Hilfsmittel an, die im Kontext mobiler Datenerhebung relevant sind. Beispielsweise ist jeder Fragebogen mehrsprachig konfigurierbar, wobei die zu unterstützenden Sprachen beliebig gewählt werden können. Außerdem existieren zum aktuellen Zeitpunkt 19 verschiedene Fragetypen: von primitiven Standardfragen wie Wert- oder Texteingaben über interaktive Elemente, wie zum Beispiel Fragen zum Erstellen einer Rangfolge per Drag-and-Drop, bis hin zu technisch komplexen Fragetypen, die beispielsweise Freihandzeichnung auf einem

zugrundeliegenden Bild erfordern. Neben Fragen existieren noch weitere Elemente für Überschriften, Text, Abbildungen und sogar ein benutzerdefinierbares Element, welches über JSON-Objekte konfiguriert wird. Dies muss jedoch letztendlich auch von der mobilen Endanwendung unterstützt werden, was eine strikte Definition des Aufbaus dieser JSON-Objekte erfordert.

All diese genannten Bestandteile des Fragebogen-Konfigurators sind ebenso für die Vorschau-Funktion von Bedeutung. Denn analog zu den mobilen Anwendungen muss auch hier jedes einzelne Element visualisiert sowie eine Sprachauswahl implementiert werden. Im Hinblick auf eine eventuelle spätere Wiederverwendung der Fragebogen-Komponente des Vorschau-Moduls als Web-Client, sollen die Fragen nicht nur visualisiert werden, sondern insbesondere auch so verwendbar sein wie in den mobilen Endanwendungen, d.h. sie müssen bedienbar und ihre Eingabe überprüfbar sein.

2.1.3 Ausgangspunkt für die Vorschau-Funktion

Der Konfigurator besitzt mehrere Tabs für die Verwaltung aller relevanten Informationen eines Fragebogens:

- *Questionnaire*: Allgemeine Informationen zum Fragebogen, wie Titel, Beschreibung, Sprachen, etc.
- *Designer*: Erstellung des eigentlichen Fragebogens und dessen Strukturierung
- *Labels*: Anzeigetext für sonstige Elemente (z.B. „Weiter”-Button)
- *Media*: Verwaltung von Medien für den Fragebogen
- *Publish*: Veröffentlichung des fertigen Fragebogens

Der Ausgangspunkt für die Vorschau-Funktion ist der Tab *Designer*, da hier der eigentliche Inhalt des Fragebogens, welcher später auch angezeigt werden soll, modelliert und verwaltet wird. Im Designer müssen alle Elemente sich innerhalb von Page-Elementen, also von „Seiten”, befinden. In der Vorschau soll jeweils immer eine solche Seite im Ganzen dargestellt werden. Denn ein Fragebogen ist wie in der analogen, gedruckten Version auch hier seitenweise gegliedert und somit werden beim Endnutzer ebenfalls

immer alle Elemente einer Seite zusammen auf dem Bildschirm des mobilen Geräts angezeigt. Folglich wird neben einer Position für den Aufruf der Vorschau auch eine Seitenauswahl benötigt.

2.2 What You See Is What You Mean

Wie in Abschnitt 2.1 beschrieben, werden im Konfigurator abstrakte Elemente für die Erstellung und Konfiguration von Fragebögen verwendet. Dieses Prinzip folgt dem sogenannten WYSIWYM-Paradigma (What-You-See-Is-What-You-Mean). Dabei liegt der Fokus auf dem semantischen Inhalt und nicht auf dem visuellen Ergebnis, wie es beim WYSIWYG-Paradigma (What-You-See-Is-What-You-Get) der Fall ist.

Ein gängiges Beispiel für WYSIWYM ist die Dokumentensprache LaTeX [5], bei welcher Layout, Schriftgröße, etc., das heißt alle globalen Formatierungen, separat vom Inhalt definiert werden und erst nach der Kompilierung das tatsächliche Ergebnis sichtbar wird. Auch bei den weit verbreiteten Auszeichnungssprachen XML und HTML wird die Struktur nach diesem Prinzip von der Darstellung getrennt.

In ähnlicher Weise wurde auch der Fragebogen-Konfigurator entwickelt. Hierbei wird die Darstellung der einzelnen Fragebogen-Elemente von IT-Experten entwickelt und somit im System vorgegeben. Den Inhalt wiederum können Fachexperten eigenständig für ihr jeweiliges Forschungsgebiet auf einfache Weise definieren ohne sich über das Layout Gedanken machen zu müssen.

Das WYSIWYG-Paradigma, auf der anderen Seite, ist in vielen Fällen jedoch intuitiver und daher auch weiter verbreitet. Vorteile sind dabei zum Beispiel, dass keine zusätzliche Vorschau-Funktion erforderlich ist und die Arbeitsweise mehr der menschlichen Denkweise angepasst ist, da das Ergebnis der Operationen dem Nutzer sofort ersichtlich ist. Da bei der hier betrachteten Anwendung jedoch nicht die Flexibilität bezüglich visueller Darstellung, sondern vielmehr die Effizienz und Einfachheit bei der Erstellung komplexer Fragebögen im Vordergrund steht, wurde im Rahmen des Gesamtprojekts die Umsetzung des WYSIWYM-Paradigmas vorgezogen.

3

Konzept

Beim Entwurf des Konzeptes müssen verschiedene Einflüsse berücksichtigt werden. Im gesamten Projekt wird für die clientseitige Entwicklung das ANGULAR Framework verwendet, dessen modulare und komponentenorientierte Struktur sich besonders auf den Aufbau der Vorschau auswirkt. Weitere Einflüsse sind das Gesamtprojekt (z.B. durch Unterstützung von Mehrsprachigkeit in den Fragebögen) sowie eigene Überlegungen zu Benutzerfreundlichkeit und Realitätstreue.

3.1 Anforderungen

Die Anforderungen für die Vorschau-Funktion gliedern sich in funktionale und nicht-funktionale Anforderungen. Funktionale Anforderungen beschreiben Eigenschaften, die erforderlich sind, damit die entwickelte Komponente vollumfänglich als Vorschau für das spätere Formular dienen kann. Die nicht-funktionalen Anforderungen beschreiben Eigenschaften, die benötigt werden, damit die Vorschau gut bedienbar und aus IT-Sicht gut administrierbar und erweiterbar ist.

3.1.1 Funktionale Anforderungen

Gerätesimulation

Da erstellte Fragebögen auf Mobilgeräten bearbeitet werden, ist es wichtig den Konfigurator-Nutzern eine realitätsgetreue Vorstellung vom fertigen Fragebogen bereitzustellen. Aus diesem Grund, sollen mobile Gerät simuliert werden, welche den Fragebogen darstellen.

Auswahl unterschiedlicher Geräte

Insbesondere sollen Geräte verschiedener Typen (Tablet und Smartphone) und Hersteller zur Auswahl stehen, wobei vor allem deren verschiedene Bildschirmgrößen von Bedeutung sind. Denn der Inhalt (also die Fragebogen-Seite) passt sich an die Größe des umgebenden Rahmens an und kann daher in der Darstellung variieren. Mit der Simulation verschiedener Geräte soll eine noch bessere Kontrolle und leichtere Anpassbarkeit des textuellen Fragebogeninhalts ermöglicht werden.

Geräteausrichtung

Des Weiteren sollen die Geräte in ihrer Ausrichtung gedreht werden können, sodass sowohl der sogenannte *Portrait-Modus* (vertikale Ausrichtung), als auch der *Landscape-Modus* (horizontale Ausrichtung) simuliert werden können.

Responsivität

Der Fragebogen-Konfigurator ist zwar primär für größere Bildschirme ausgelegt, dennoch können diese auch bei PCs und Laptops stark variieren. Insbesondere können Browserfenster zur Laufzeit vom Benutzer vergrößert und verkleinert werden. Um eine optimale Darstellung bieten zu können, soll die Vorschau sich responsiv an die Größe des Browserfensters anpassen. Im Zuge dessen soll auch das simulierte Gerät skalieren.

Ferner soll sich auch der im Gerät dargestellte Fragebogen dynamisch an seinen Rahmen anpassen können, was eine responsive Implementierung der Fragebogen-Elemente voraussetzt.

Sprachauswahl

Bei mehrsprachigen Fragebögen kann sich der textuelle Inhalt von Sprache zu Sprache in seiner Länge und Form unterscheiden. Daher soll die Vorschau eine Sprachauswahl bieten, um dem Nutzer eine optimale Anpassung des Inhalts in jeder Sprache zu ermöglichen.

Realitätstreue

Aufgrund der verschiedenen Plattformen können die bereits existierenden Fragebogen-Elemente der mobilen Endanwendungen nicht für die zu entwickelnde Vorschau wie-

derverwendet werden. Somit wird sich die visuelle Darstellung des Fragebogens in der Vorschau zwangsläufig von der tatsächlichen auf den mobilen Geräten unterscheiden. Dennoch sollen die Elemente in der Vorschau möglichst realitätsgetreu gestaltet werden, um dem Nutzer des Konfigurators einen optimalen Eindruck vom Ergebnis bieten zu können.

3.1.2 Nicht-funktionale Anforderungen

Zugreifbarkeit

Der Zugriff auf die Vorschau-Funktion soll effizient und intuitiv gestaltet werden. Zugleich soll die Vorschau kein Störfaktor für die Nutzung der restlichen Funktionen des Konfigurators sein, z.B. durch dauerhafte Überlagerung oder Verkleinerung anderer Bereiche.

Geringe Latenzzeit

Um ein effizientes Arbeiten zu gewährleisten, sollen Verzögerungen durch das Laden der Vorschau oder ihres Inhalts minimiert werden. Folglich soll auch ein Wechsel der Sprache oder des Geräterahmens optimalerweise keinen verzögernden Einfluss auf die Darstellung des Fragebogens haben.

Erweiterbarkeit

Da das Projekt stetig weiterentwickelt wird und im Zuge dessen beispielsweise neue Fragetypen oder andere Elemente hinzugefügt werden können, soll sich eine Erweiterung der Vorschau möglichst einfach gestalten. Hierfür ist ein modularer Aufbau ebenso wichtig wie eine dynamische Verarbeitung der Fragebogen-Elemente, da diese in ihrer Anzahl, Art und Reihenfolge auf einer Fragebogen-Seite beliebig sind.

Wiederverwendbarkeit

Bislang existieren Endanwendungen für das Ausfüllen von Fragebögen lediglich für mobile Geräte. Da nun für die Vorschau-Funktion ebenfalls eine Visualisierung der Fragebögen für Webbrowser entwickelt werden muss, wird in Erwägung gezogen in Zukunft auch eine Endanwendung als Web-Client zur Verfügung zu stellen. Daher soll

die Fragebogen-Darstellung von den restlichen Funktionen des Vorschau-Moduls abgekapselt werden, d.h. die Schnittstelle zwischen Fragebogen-Seite und ihrer Umgebung (d.h. dem Geräterahmen) soll minimal sein.

3.2 Strukturierung

In diesem Abschnitt wird die grobe Struktur der Vorschau sowie des Zugriffs darauf festgelegt. Es geht dabei vorrangig um die Wahl des visuellen Aufbaus, welcher für eine gute Benutzerfreundlichkeit von hoher Bedeutung ist. Es werden daher im Folgenden denkbare Strukturierungen diskutiert und die für den gegebenen Fall optimalen Lösungen vorgestellt.

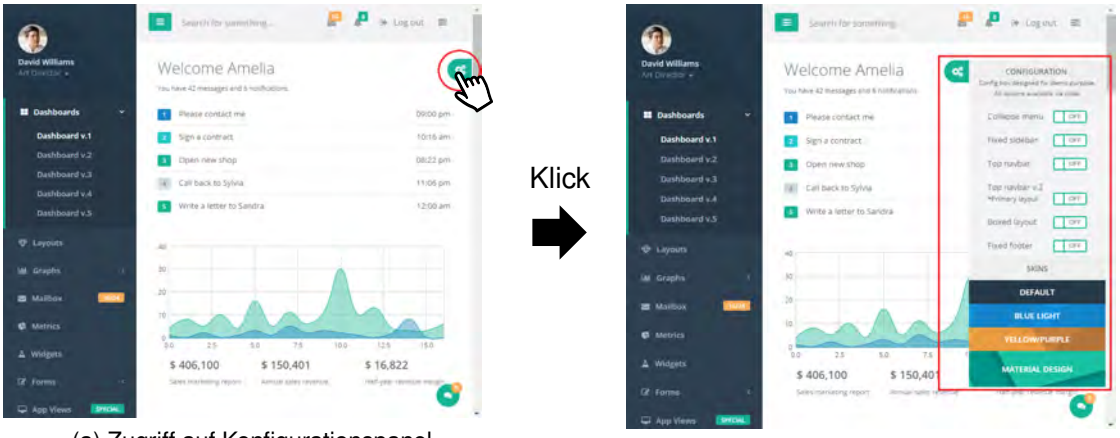
Zugriff auf die Vorschau-Funktion

Laut definierter Anforderung soll sich der Zugriff auf die Vorschau-Funktion effizient und intuitiv gestalten. Es muss folglich ein Konzept entwickelt werden, nach dem ein Nutzer

1. ohne Vorwissen von dieser Funktion Kenntnis erlangt,
2. die Vorschau mit so wenigen Aktionen wie möglich aufrufen kann und
3. die Auswahl der zu betrachtenden Fragebogen-Seite dabei intuitiv vonstatten geht.

Unter Betrachtung des ersten Kriteriums war bei der Konzeptionierung die erste Überlegung, ein zusätzliches Fenster zu verwenden, das über ein dauerhaft sichtbares Symbol am Bildschirmrand aufgerufen werden kann. Vorbild hierfür war das Konfigurationspanel (siehe Abbildung 3.1) auf der Demoseite von INSPINIA[1]. Das INSPINIA-Theme wird im gesamten Projekt als Basis-Template verwendet, weshalb es naheliegend war, eine Komponente hiervon wiederzuverwenden. Jedoch dient dieses Panel ausschließlich zur Konfiguration der Demoseite und wird nicht mit dem Theme ausgeliefert. Somit müsste diese Komponente von Grund auf neu implementiert werden. Davon abgesehen ist diese Art Zugriff auf die Vorschau-Funktion auch nicht intuitiv, da sie *außerhalb* des Designers liegt und zudem eine separate Auswahl der anzuzeigenden Fragebogen-Seite erfordert.

[1] Demoseite von INSPINIA: http://webapplayers.com/inspinia_admin-v2.7.1/ – abgerufen am 13.06.2018



(a) Zugriff auf Konfigurationspanel

(b) Geöffnetes Konfigurationspanel

Abbildung 3.1: Konfigurationspanel auf der Demoseite von INSPINIA

Um diese Auswahl intuitiver zu gestalten (siehe Kriterium 3), wäre ein Zugriff über ein Page-Element (repräsentiert eine Fragebogen-Seite) im Designer besser geeignet. Hierfür wäre beispielsweise ein Symbol direkt auf dem Element oder ein Eintrag im Kontext-Menü denkbar. Ersteres würde nur einen einzelnen Klick erfordern, um die Seite auszuwählen und die Vorschau aufzurufen. Die Lösung über das Kontext-Menü würde hingegen zwei Klicks erfordern (ein Rechtsklick auf das Page-Element und ein Klick auf den Eintrag im Kontext-Menü) und wäre somit unter Betrachtung von Kriterium 2 die schlechtere Wahl. Da jedoch das Kontext-Menü bereits implementiert ist für andere Operationen wie „Löschen” oder „Kopieren” und hierbei sogar nach Element-Typ gefiltert werden kann, sodass die Vorschau beispielsweise nur bei Page-Elementen im Kontext-Menü auftaucht, fiel die Wahl wiederum unter Betrachtung von Kriterium 1 schlussendlich auf diese Lösungsvariante (siehe Abbildung 3.2).

Vorschau-Bereich

Da das Konfigurationspanel von INSPINIA nicht zur Verfügung steht, fiel die Entscheidung auf ein eigenes Dialog-Fenster für den Vorschau-Bereich, das sich beim Zugriff öffnet. Dieses überlagert den Konfigurator vollständig und muss geschlossen werden, um mit der Bearbeitung fortsetzen zu können. Die Überlagerung widerspricht in Teilen dem Anforderungspunkt *Zugreifbarkeit*, da es den Zugriff auf andere Funktionen verhindert.

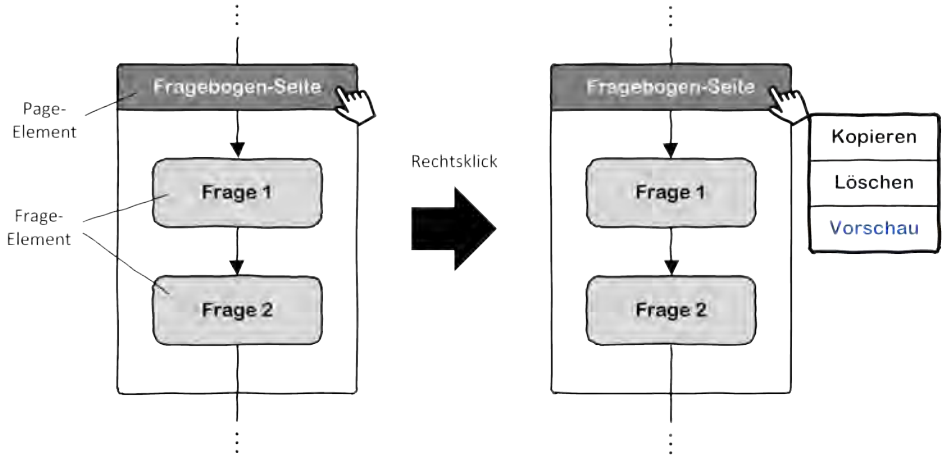


Abbildung 3.2: Zugriff auf die Vorschau-Funktion

Würde die Vorschau jedoch in die Konfigurator-Ebene integriert werden, würde sie notgedrungen andere Bereiche verkleinern und somit deren Darstellung und Zugriff insbesondere auf kleineren Bildschirmen verschlechtern. Weil die Modellierungsoberfläche des Konfigurators (im Tab *Designer*; siehe Abschnitt 2.1.3) ohnehin bereits in mehrere Spalten unterteilt ist, stellt die Platzierung der Vorschau in einer neuen Ebene hier den besten Kompromiss dar.

Die Auswertung der Anforderungen resultiert in vier Komponenten, die innerhalb der Vorschau benötigt werden:

1. Sprachauswahl
2. Auswahl für den Geräterahmen
3. Auswahl für die Orientierung des simulierten Geräts
4. Darstellung des gewählten Geräts mit eigentlicher Vorschau der Fragebogen-Seite

Drei dieser vier Komponenten stellen offensichtlich Einstellungsmöglichkeiten dar, die die vierte Komponente beeinflussen. Es bietet sich somit an die Vorschau in einen Konfigurationsbereich und einen Vorschau-Bereich zu unterteilen. Dies ist in Abbildung 3.3 schematisch dargestellt.

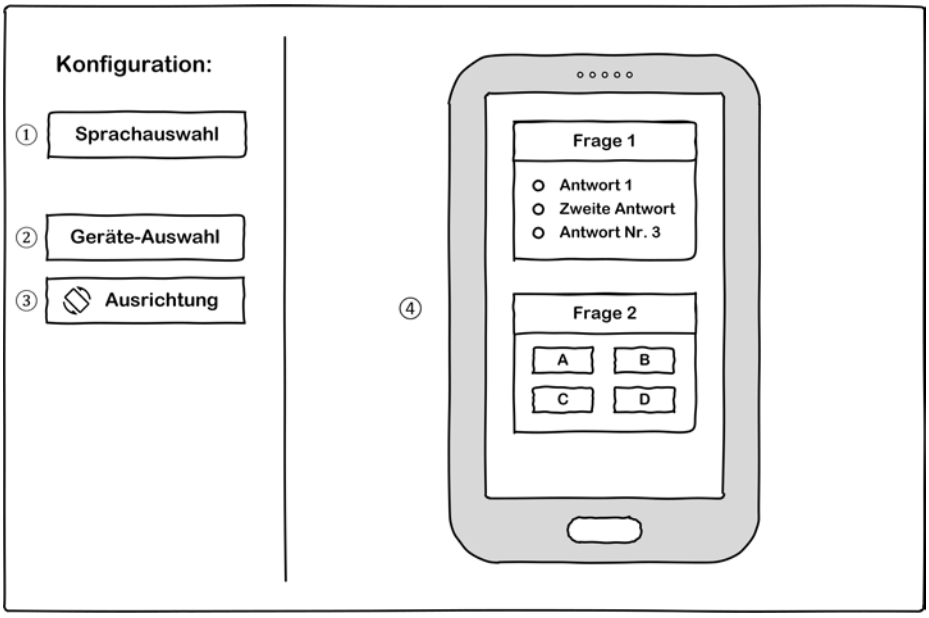


Abbildung 3.3: Struktur der Vorschau

3.3 Architektur

Die Vorschau sollte ein eigenständiges Modul sein, da sie bis auf die Abhängigkeiten bezüglich der Liste an darzustellenden Fragebogen-Elementen sowie die vom Fragebogen unterstützen Sprachen unabhängig vom Konfigurator ist. Wie im vorigen Abschnitt beschrieben, wird die Vorschau über einen Klick auf den entsprechenden Kontext-Menü-Eintrag eines Page-Elements aufgerufen (Abbildung 3.4). Dabei werden die notwendigen Daten an die Vorschau-Komponente übertragen. Ab diesem Zeitpunkt ist die Vorschau vollständig unabhängig vom Konfigurator und interagiert auch nicht mit diesem.

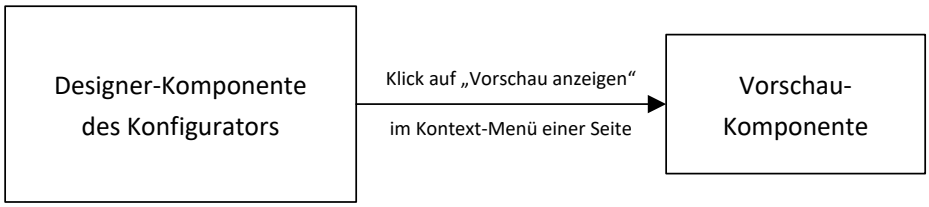


Abbildung 3.4: Architektur auf Konfigurator-Ebene

Die im vorigen Unterabschnitt *Vorschau-Bereich* aufgelisteten Komponenten sollten nach den Prinzipien der Erweiterbarkeit und Wiederverwendbarkeit ebenfalls jeweils eigene Module bilden. Konkret muss die Vorschau somit folgende Module bzw. Subkomponenten verwalten (in Anlehnung an die noch folgende Implementierung werden ab hier zunehmend englische Begriffe für Komponenten, etc. verwendet):

- *Device Manager:* Verwaltet die Geräteauswahl und -darstellung sowie die Geräteausrichtung.
- *Geräte-Bibliothek:* Verwaltet alle Geräte, wobei jedes Gerät eine Referenz zu seiner Komponente, die für die Visualisierung verantwortlich ist, sowie weitere Eigenschaften enthält, wie z.B. Hersteller und Bezeichnung, welche für die Geräteauswahl von Bedeutung sind.
- *Sprachauswahl:* Verwaltet die Auswahl aller für diesen Fragebogen verfügbaren Sprachen.
- *Dynamic Forms:* Verarbeitet die Elemente der gewählten Seite dynamisch und wählt dabei die jeweils darzustellende Komponente aus.
- *Element-Bibliothek:* Verwaltet für jedes mögliche Element eine Formular-Komponente, die für dessen Visualisierung verantwortlich ist.

Die Architektur des Vorschau-Moduls ist in Abbildung 3.5 veranschaulicht. Wie darin ersichtlich ist, interagieren die Subkomponenten der Vorschau miteinander. Die Daten fließen dabei immer über die verwaltende Vorschau-Komponente.

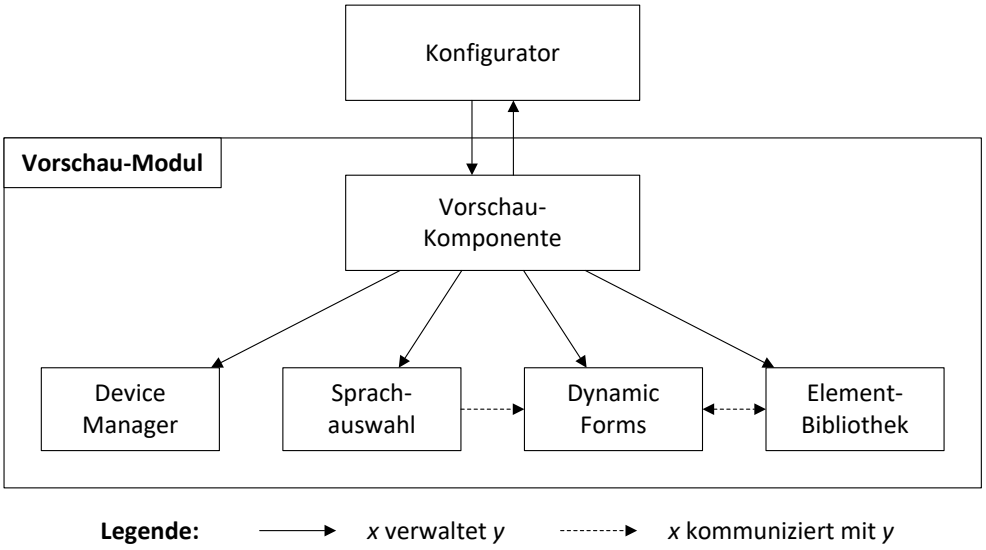


Abbildung 3.5: Architektur auf Vorschau-Ebene

3.4 Interne Kommunikation

Wie im vorigen Abschnitt beschrieben soll die Vorschau-Komponente als Verwaltungsorgan ihrer Subkomponenten fungieren. Konkret heißt das, sie ist der Initiator für alle Aktionen ihrer Subkomponenten. Sie ruft von diesen alle notwendigen Daten ab und baut daraus die Vorschau zusammen. Der Aufruf der Vorschau sollte somit eine Kommunikation initiieren, wie sie in Abbildung 3.6 dargestellt und im Folgenden beschrieben ist:

1. Der Nutzer ruft über die Benutzerschnittstelle (UI) die Vorschau-Funktion auf, woraufhin die Daten der gewählten Seite sowie die Sprachenliste vom Konfigurator an die *Vorschau*-Komponente übermittelt werden.

2. Die *Vorschau*-Komponente (VK) initialisiert die Sprachauswahl, welche eine Default-Sprache an die VK zurückgibt.

3. Die VK initialisiert nun die *Device Manager*-Komponente, diese ermittelt ein Default-Gerät, lädt die zugehörige Komponente aus der Geräte-Bibliothek und leitet sie an die VK weiter.

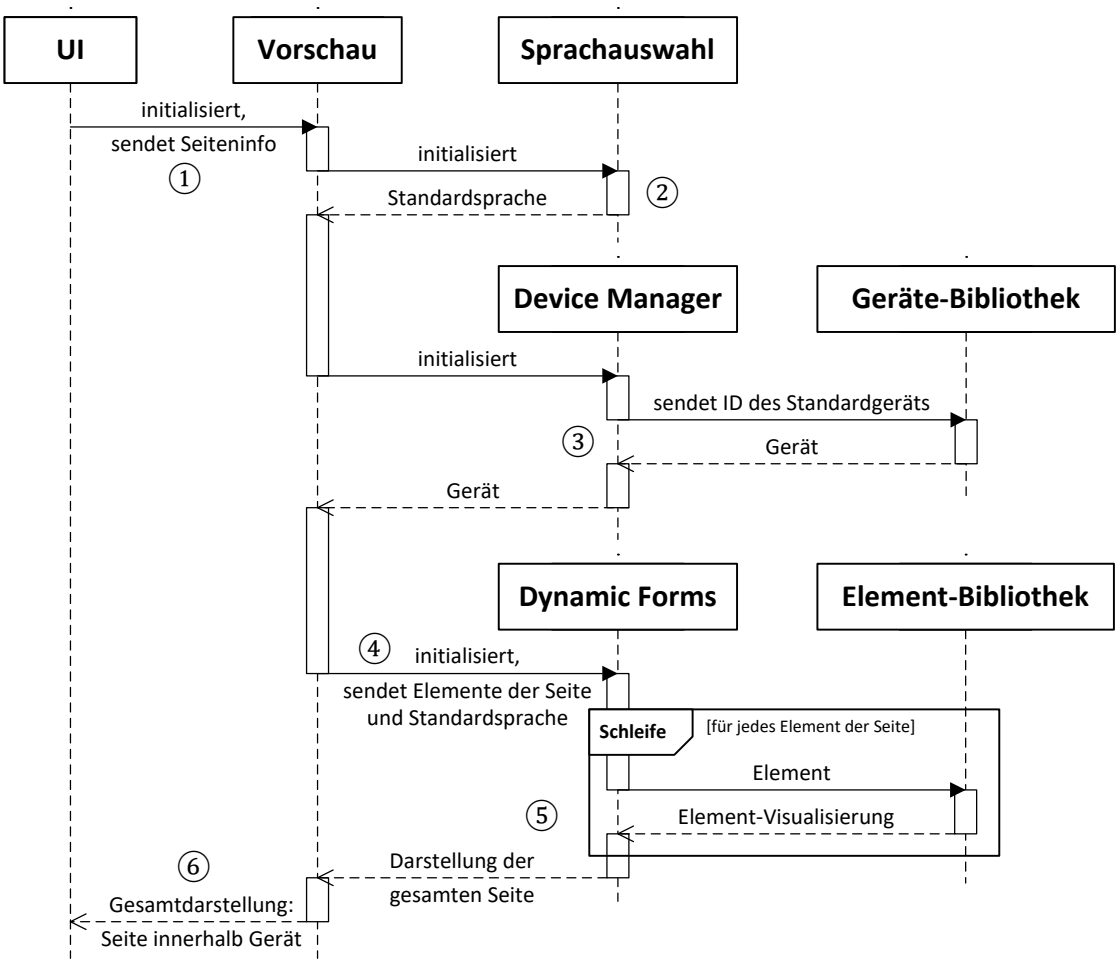


Abbildung 3.6: Interne Kommunikation beim Aufruf der Vorschau

4. Die VK initialisiert in ihrem dritten Schritt die *Dynamic Forms*-Komponente mit der Liste an Elementen der gewählten Seite sowie der Default-Sprache.
5. Die *Dynamic Forms*-Komponente lädt für jedes Element die zugehörige Komponente aus der Element-Bibliothek und baut daraus die Fragebogen-Seite in der gewählten Sprache zusammen. Anschließend gibt die *Dynamic Forms*-Komponente die Seite an die VK zurück.
6. Die VK integriert die Seite des Fragebogens in den Geräterahmen und gibt das Gesamtkonstrukt an die UI zurück, welche nun dem Nutzer die Vorschau darstellen kann.

Ändert der Nutzer im weiteren Verlauf die Sprache, soll sich dies nur auf die *Dynamic Forms*-Komponente auswirken. Dabei sollen die Elemente nicht neu zusammengebaut, sondern lediglich ihr textueller Inhalt verändert werden. Ändert der Nutzer das Gerät, soll umgekehrt auch nur der *Device Manager* reagieren und die enthaltene Fragebogen-Seite von der VK lediglich aus dem alten Geräterahmen extrahiert und in den neuen Geräterahmen integriert werden. Dies ist jedoch nicht unter jeden Umständen realisierbar, wie aus der noch folgenden Diskussion in Abschnitt 4.3 hervorgeht.

4

Implementierung

Im folgenden Kapitel werden die Vorgehensweise bei der Implementierung der Vorschau-Funktion und die dabei verwendeten Konstrukte präsentiert. Weiterhin werden wichtige Entscheidungen diskutiert, die für ein optimales Ergebnis dieser Arbeit von Bedeutung sind.

4.1 Allgemeines

In diesem ersten Abschnitt werden zunächst die verwendeten Drittanbieter-Produkte vorgestellt, die für das gesamte Vorschau-Modul benötigt werden. Insbesondere wird in die für diese Arbeit relevanten Konzepte und Strukturen des ANGULAR Frameworks eingeführt.

4.1.1 ANGULAR

ANGULAR ist ein JavaScript-Framework für clientseitige Webentwicklung, welches stark auf Modularisierung setzt. Dadurch sind ANGULAR-Projekte einfach erweiterbar. Die Benutzerschnittstelle wird hierbei in wiederverwendbare Komponenten unterteilt, wobei sowohl Layout (HTML) und Optik (CSS), als auch Verhalten (JS) innerhalb der Komponente definiert werden[1]. ANGULARS bidirektionales Data-Binding[2] ermöglicht insbesondere den einfachen Umgang mit Formularen, da hierdurch Variablen an HTML-Elemente bzw. deren Inhalt gebunden werden können. D.h. der Inhalt oder Wert eines

[1] Überblick über Architektur von ANGULAR: https://angular.io/guide/architecture – abgerufen am 13.06.2018
[2] Einführung in ANGULARS Data-Binding: https://angular.io/guide/architecture-components#data-binding – abgerufen am 13.06.2018

Formularfeldes wird bei Eingabe bzw. Auswahl automatisch in einer JavaScript-Variable gespeichert, was dessen weitere Verarbeitung erheblich vereinfacht. Außerdem bietet das Framework eine einfache benutzerdefinierbare Validierung von Formular-Eingaben[3]. Dies ist für das vorliegende Projekt von großer Bedeutung, da sehr viele verschiedene Fragetypen möglich sind und diese insbesondere meist nicht mithilfe von Standard-HTML-Formularelementen implementiert werden können.

4.1.2 Struktur von ANGULAR-Anwendungen

Durch den komponentenorientierten Ansatz von ANGULAR wird eine Struktur vorgegeben, sodass die im Konzept entwickelte Architektur nicht vollständig umgesetzt werden kann. Bei ANGULAR wird jede Komponente durch eine HTML-Direktive repräsentiert und kann dadurch an beliebiger Stelle eingebunden werden.

Eine Beispiel-Komponente `Component` sei durch die Direktive `<component>` repräsentiert. Das HTML-Template dieser Komponente sei wie folgt definiert:

```
1 <h3>Beispiel-Komponente</h3>
2 <p>Dies ist der Inhalt der Komponente.</p>
```

An beliebiger Stelle kann `Component` nun folgendermaßen eingefügt werden:

```
1 <div id="component-container">
2    <component></component>
3 </div>
```

Nach der Ausführung des ANGULAR-Compilers entsteht dadurch das HTML-Fragment:

```
1 <div id="component-container">
2    <h3>Beispiel-Komponente</h3>
3    <p>Dies ist der Inhalt der Komponente.</p>
4 </div>
```

Diese Architektur implementiert eine gute Daten-Kapselung und ermöglicht dadurch eine einfache Wiederverwendbarkeit der Komponenten. HTML ist jedoch trotz der Dynamik von ANGULAR (z.B. Data-Binding) immer noch eine statische Sprache. Somit

[3] Form-Validation von ANGULAR: https://angular.io/guide/form-validation – abgerufen am 13.06.2018

muss zum Compile-Zeitpunkt feststehen, an welcher Stelle welche Komponente angezeigt werden soll. Da der Fragebogen innerhalb des simulierten mobilen Geräts angezeigt wird und jedes Gerät eine eigene Komponente mit eigenem HTML-Template darstellt (siehe Abschnitt 4.4), muss folglich auch die für die Fragebogen-Seite zuständige `DynamicForms`-Komponente innerhalb jedes Geräts referenziert werden.

Weiterhin ist HTML wohlgeformt [6], d.h. es besitzt einen geschachtelten Aufbau. Daher werden (Kind-)GoJS-Elemente automatisch mit gelöscht, wenn das sie umgebende Eltern-Element entfernt wird. Somit wird auch die Fragebogen-Seite bei jedem Gerätewechsel neu geladen (die alte Geräte-Komponente wird gelöscht, eine neue wird eingefügt und initialisiert). Um dies zu verhindern, müssten Gerät und Fragebogen auf derselben Ebene im HTML-Code platziert werden und der Fragebogen dann mit CSS `position` über das Gerät geschoben werden. Dadurch entstehen jedoch zwei übereinanderliegende Schichten. Dieser Ansatz, mit seinen Vor- und Nachteilen, wird in Abschnitt 4.3 ausführlicher diskutiert.

4.1.3 Benutzerschnittstelle

In der Vorschau werden für die Gestaltung der Benutzerschnittstelle die CSS/JS-Bibliotheken BOOTSTRAP[4] und ANGULAR MATERIAL[5] eingesetzt. Letzteres ist ein Teilprojekt von ANGULAR, das vordefinierte UI-Komponenten liefert. Da diese bereits mit ANGULAR implementiert sind, lassen sie sich ideal integrieren. Umkehrt ist ANGULAR MATERIAL nur in Verbindung mit dem ANGULAR Framework verwendbar.

4.2 Vorschau-Komponente

Diese Komponente stellt die zentrale Komponente des Vorschau-Moduls dar. Sie initialisiert ihre Subkomponenten und koordiniert den Aufbau der Vorschau der vom Benutzer gewählten Seite des Fragebogens. Damit stellt sie die Schnittstelle zwischen der

[4] BOOTSTRAP: https://getbootstrap.com/ – abgerufen am 13.06.2018
[5] ANGULAR MATERIAL: https://material.angular.io/ – abgerufen am 13.06.2018

Fragebogen-Visualisierung (erfolgt durch ihre Subkomponenten), dem Konfigurator und der Benutzeroberfläche der Vorschau dar.

Daneben besitzt die Vorschau-Komponente auch eigenständig durchzuführende Aufgaben wie die Vorverarbeitung der Fragebogen-Daten für ihre Subkomponenten oder der initialen Auswahl einer Sprache. Diese Funktionen werden im folgenden Abschnitt beschrieben und diskutiert.

4.2.1 Initiierung

Wie bereits erwähnt, findet die einzige Kommunikation zwischen Konfigurator und Vorschau zum Zeitpunkt ihrer Initiierung statt. Dabei initialisiert die Designer-Komponente des Konfigurators die Vorschau-Komponente (erzeugt also das Dialog-Fenster) und einen Vorschau-Service. Letzterer erhält vom Designer den Teilgraphen, der die vom Benutzer gewählte Seite enthält, in Form einer Knoten-Liste und einer Kanten-Liste sowie die Liste der vom Fragebogen unterstützen Sprachen. Die Knoten besitzen dabei Referenzen auf die Fragebogen-Elemente. Die Kanten-Liste beinhaltet die Informationen, welche Knoten in welcher Reihenfolge miteinander verbunden sind. Anhand dieser Liste werden die Knoten sortiert, sodass die `DynamicForms`-Komponente später schlicht über die Knoten-Liste zu iterieren braucht und damit die Fragebogen-Elemente automatisch in der korrekten Reihenfolge darstellt.

Die Tatsache, dass die gesamten Elemente als JSON-Objekte direkt am Graph des Designers hängen, ermöglicht eine rein clientseitige Entwicklung der Vorschau, d.h. es ist keine Abfrage der Element-Informationen vom Server notwendig. Für diesen Graph wird ein Flowchart-Diagramm der JavaScript-Bibliothek GoJS[6] verwendet, welche intuitives Arbeiten mit Graphen und anderen Diagrammen im Webbrowser ermöglicht.

4.2.2 Sprachauswahl

Ein weiterer Schritt bei der Initialisierung der Vorschau-Komponente ist die Auswahl einer voreingestellten Sprache für die Anzeige des Fragebogens. Dies ist in diesem Fall

[6] GoJS-Flowchart: https://gojs.net/latest/samples/flowchart.html – abgerufen am 13.06.2018

keineswegs trivial. Denn alle vom Fragebogen zu unterstützenden Sprachen werden vom Konfigurator-Benutzer ausgewählt, weshalb es keine feste Standardsprache gibt, die von jedem Fragebogen unterstützt wird. Die einfachste und naheliegendste Lösung ist die Wahl des ersten Eintrags in der Sprachen-Liste; denn dies kann sogar statisch so programmiert werden. Jedoch zeugt dies von geringer Benutzerfreundlichkeit, wenn der Nutzer bei jedem Aufruf der Vorschau zuerst seine präferierte Sprache einstellen muss. Denn würden beispielsweise Arabisch und Englisch im Fragebogen unterstützt, würde (unter Annahme von alphabetischer Sortierung) standardmäßig Arabisch ausgewählt, was insbesondere dann ein Problem darstellt, wenn der Konfigurator-Benutzer selbst nur Englisch spricht und die Übersetzung ins Arabische von einer anderen Person gepflegt wird.

Da Englisch die Weltsprache ist und heutzutage international von einem Großteil der Menschen verstanden wird, werden mehrsprachige Fragebögen wohl in fast allen Fällen die englische Sprache unterstützen. Daher war die zweite Überlegung, standardmäßig immer die englische Sprache als Vorauswahl anzugeben. Allerdings ist es weiterhin möglich, Fragebögen ohne Englisch-Unterstützung zu erstellen, weshalb in diesem Fall immer noch auf die erste Lösungsvariante zurückgegriffen werden muss. Es wird nun also im Falle der Verfügbarkeit Englisch und ansonsten die erste Sprache in der Liste ausgewählt.

Dies ist zwar in sehr vielen Fällen eine weitaus hilfreichere Voreinstellung, jedoch noch immer nicht ideal. Wird zum Beispiel ein französischer Fragebogen erstellt, der zusätzlich auch Englisch unterstützen soll, so wird hierbei die Fremdsprache (Englisch) anstatt die Muttersprache (Französisch) standardmäßig ausgewählt. Deshalb wird nun in der finalen Lösung zuallererst überprüft, ob die Browsersprache (im Regelfall die Muttersprache des Benutzers) auch vom Fragebogen unterstützt wird. Wenn ja, wird diese ausgewählt, ansonsten wird auf die vorige Lösungsvariante zurückgegriffen. Der endgültige Entscheidungsbaum ist in Abbildung 4.1 noch einmal verdeutlicht.

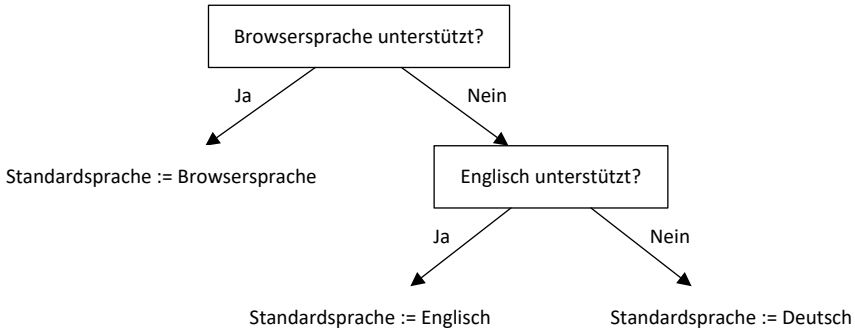


Abbildung 4.1: Auswahl der Standardsprache des Fragebogens

4.3 Geräte-Simulation

Die Simulation mobiler Geräterahmen für die Fragebogen-Vorschau stellte die größte Herausforderung dieser Abschlussarbeit dar. Insbesondere eine Auswahl mehrerer Geräte anzubieten und die damit verbundene dynamische Änderung des Containers, in dem die Fragebogen-Seite platziert wird, brachte Probleme mit sich, die in diesem und im nächsten Abschnitt diskutiert sowie mögliche Lösungen vorgestellt werden.

Das Hauptproblem ist die Darstellung unterschiedlicher Geräterahmen. Da diese austauschbar sein sollen, kann die Rahmen-Komponente nicht statisch in HTML implementiert werden. Darüber hinaus soll der Wechsel der Geräte optimalerweise die Seite des Fragebogens, also den HTML-Inhalt innerhalb des Rahmens, nicht beeinflussen. Im Rahmen dieser Arbeit konnten drei verschiedene Möglichkeiten identifiziert werden, wie ein mobiles Gerät simuliert werden kann:

- Das Gerät wird schlicht durch eine Grafik (z.B. im PNG-Format) dargestellt, die in den Hintergrund gelegt wird (siehe *Gerät als Grafik im Hintergrund)*
- Das Gerät wird rein mithilfe von HTML und CSS nachgestellt und enthält die Fragebogen-Seit als Kind-Element (siehe *Gerät als HTML/CSS-Imitation)*
- Der Geräterahmen (Grafik oder HTML/CSS-Imitation) wird in Bruchstücke aufgeteilt und um die Fragebogen-Seite herum „gebaut” (was das konkret bedeutet, wird im Weiteren noch erläutert; siehe *Segmentierung des Geräts unter Verwendung einer Tabelle)*

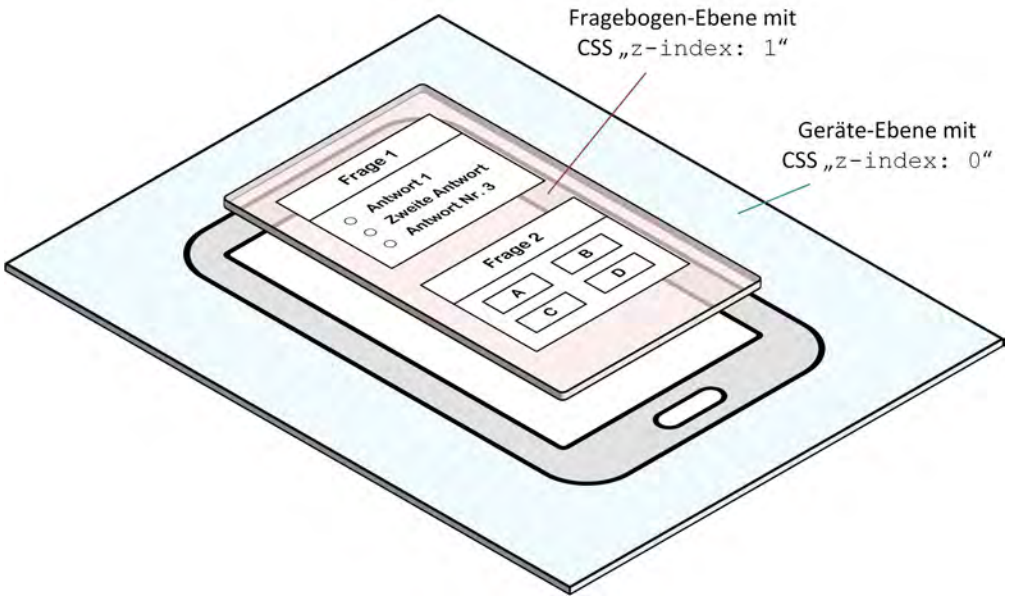


Abbildung 4.2: Geräte-Simulation mit zwei übereinanderliegenden Schichten

Gerät als Grafik im Hintergrund

Die naheliegendste Variante ist eine PNG-Grafik des Geräts zu verwenden, wobei der Bereich des Displays transparent ist bzw. herausgeschnitten wurde. Dadurch könnte man die Grafik in der Theorie sogar im Vordergrund (d.h. in der Ebene „über” dem Fragebogen) platzieren. Da jedoch der Fragebogen noch bedienbar sein soll, muss die Grafik in den Hintergrund platziert werden. In jedem Fall werden bei dieser Methode zwei Schichten benötigt (siehe Abbildung 4.2): eine für den Geräterahmen (unten) und eine für den Fragebogen (oben). Um dies zu erreichen, muss die Schicht des Fragebogens mithilfe des CSS-Attributs `z-index` in den Vordergrund gebracht werden, da HTML standardmäßig nur eine Ebene besitzt. Zusätzlich muss eine der beiden Schichten mittels CSS `position` an der anderen ausgerichtet werden.

Vorteile:

- Die Grafik kann einfach über dynamische Änderung der Source-URL ausgetauscht werden.
- Der Wechsel des Geräterahmens beeinflusst seinen Inhalt nicht, da sie sich in getrennten HTML-Elementen mit verschiedenen Eltern-Elementen befinden.

Nachteile:

- Da sich das Gerät an die Bildschirmgröße anpassen soll, müssen sowohl Grafik als auch Fragebogen-Seite skaliert werden, wobei letztere möglicherweise wieder neu ausgerichtet am Gerät ausgerichtet werden muss.
- Im Allgemeinen ist der Umgang mit CSS `position` umständlich für dynamisches Positionsverhalten und daher nicht ideal.
- Bei der Skalierung der Grafik muss auch deren Auflösung berücksichtigt werden. Ist sie zu gering, wird das Gerät unscharf dargestellt, ist sie zu hoch genug, kann es zu einem stark erhöhten Overhead und folglich zu Verzögerungen beim Laden kommen.
- Generell sind Grafiken von mobilen Geräten teilweise schwer erhältlich oder gar kostenpflichtig und müssen gegebenenfalls vor der Verwendung noch manuell bearbeitet werden.

Zusammengefasst überwiegen hierbei die Nachteile. Insbesondere ist eine Verwendung von mehreren Schichten in diesem Fall sehr ungünstig, da sich hierbei der Fragebogen nicht automatisch an den Geräterahmen anpassen kann, was von schlechter Responsivität zeugt.

Gerät als HTML/CSS-Imitation

Eine Alternative zur Grafik ist die Nachbildung der Geräterahmen mithilfe von HTML und CSS (siehe Abbildung 4.3). Dabei wird die Grundstruktur des Geräts (d.h. seine Bestandteile wie Rahmen, Knöpfte, etc.) durch HTML-Elemente festgelegt und die Optik mithilfe von CSS an die Realität angenähert.

Vorteile:

- Vermeidet die Verwendung von zwei übereinanderliegenden Schichten, da sich das HTML-Formular der Fragebogen-Seite in den HTML-Rahmen einbetten lässt.
- Simuliertes Gerät verhält sich optimal im Browser, insbesondere bei Skalierung der Vorschau.

```
<div class="mobile-device iphone8 black">
   <div class="top-bar"></div>
   <div class="sleep"></div>
   <div class="volume"></div>
   <div class="camera"></div>
   <div class="sensor"></div>
   <div class="speaker"></div>
   <div class="screen">
      <!-- Content -->
   </div>
   <div class="home"></div>
   <div class="bottom-bar"></div>
</div>
```

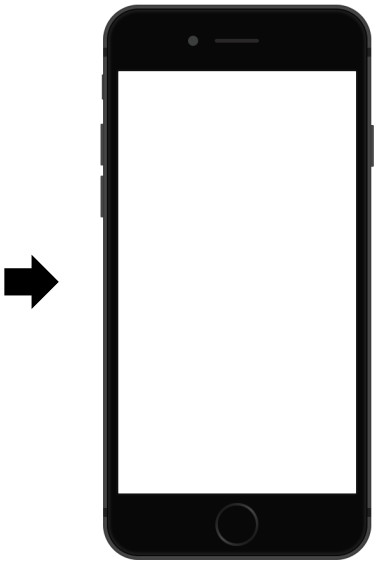


Abbildung 4.3: Geräte-Simulation rein durch HTML und CSS

- Geschachtelter Aufbau ermöglicht responsive Anpassung der Fragebogen-Seite an den Geräterahmen.
- Es existieren bereits frei zugängliche Implementierungen solcher Geräte-Imitationen, hauptsächlich zur Erstellung von Mockups mit HTML-Inhalt.

Nachteile:

- Da nun der Geräterahmen als Container für die Fragebogen-Seite dient, muss dieses jedes Mal neu erzeugt werden, wenn der Nutzer ein anderes Gerät auswählt. Dies kann insbesondere bei größerem Dateninhalt der Fragebogen-Seite Ladeverzögerungen verursachen kann.
- Auch der Gerätewechsel selbst ist aufwändiger als der schlichte Wechsel der Grafik-URL von Lösungsvariante 1, da nun beim Wechsel das neue HTML-Template des gewählten Geräts geladen werden muss.

Diese Lösung bietet also deutliche Verbesserungen von Responsivität und Skalierbarkeit und stellt außerdem auch die bessere Praktik in der HTML-Programmierung dar. Die Verzögerungen durch Neuladen der Fragebogen-Seite dürften nur selten bemerkbar

sein. Zum Beispiel dann, wenn viele Bilder im Fragebogen verwendet werden oder der Internetanschluss des Bearbeiters nur über eine geringe Bandbreite verfügt.

Segmentierung des Geräts unter Verwendung einer Tabelle

Aufgrund der noch immer möglichen Ladeverzögerungen wäre es für eine höhere Effizienz beim Gerätewechsel jedoch besser, wenn trotz guter Skalierbarkeit die strukturelle Unabhängigkeit zwischen Fragebogen und dessen umgebenden Rahmen gewährleistet werden könnte. Das heißt, der Geräterahmen sollte im HTML-DOM nicht als Container für die Fragebogen-Seite dienen, sondern (wie in Variante 1) auf derselben Ebene platziert werden können. Die einzige Möglichkeit, diesen Kompromiss zu erreichen, ist eine Segmentierung des Geräterahmens, sodass die einzelnen Segmente (sinnbildlich gesprochen) um die Fragebogen-Seite herum gelegt werden können. Für diese Konstruktion kann beispielsweise eine 3x3-HTML-Tabelle verwendet werden. Wie in Abbildung 4.4 veranschaulicht, wird dabei der Rahmen des Geräts in 8 Stücke unterteilt, welche in den Randzellen platziert werden. Dabei muss darauf geachtet werden, dass die Schnittstellen zwischen den Rahmenbruchstücken unsichtbar sind. Der im Gerät anzuzeigende Inhalt wird schließlich in die mittlere Zelle eingefügt. Dadurch ist die Fragebogen-Seite sowohl im HTML-DOM als auch in struktureller Sicht (d.h. es sind keine speziellen CSS-Attribute für die Positionierung notwendig) auf derselben Ebene mit dem Geräterahmen. Bei dieser Lösungsvariante kann der Rahmen entweder einer Grafik oder einer HTML/CSS-Nachbildung entsprechen.

Eine ähnliche Vorgehensweise wird bereits in der Android-Entwicklung verwendet, hauptsächlich um Buttons mit grafischem Hintergrund skalierbar zu machen. Mithilfe sogenannter *9-Patch-Grafiken*[7] kann dabei in der Mitte der Bereich für den Inhalt (z.B. Beschriftung) definiert werden sowie der Randbereich, dessen Hintergrund beim Hochskalieren vervielfältigt wird. Im Kontext dieser Abschlussarbeit ist eine schlichte Vervielfältigung aufgrund der komplexeren Struktur des Geräterahmens jedoch ungeeignet. Vielmehr muss jedes Bruchstück des Geräterahmens individuell skaliert werden.

[7] Erstellen von *9-Patch-Grafiken* in ANDROID STUDIO: https://developer.android.com/studio/write/draw9patch – abgerufen am 13.06.2018

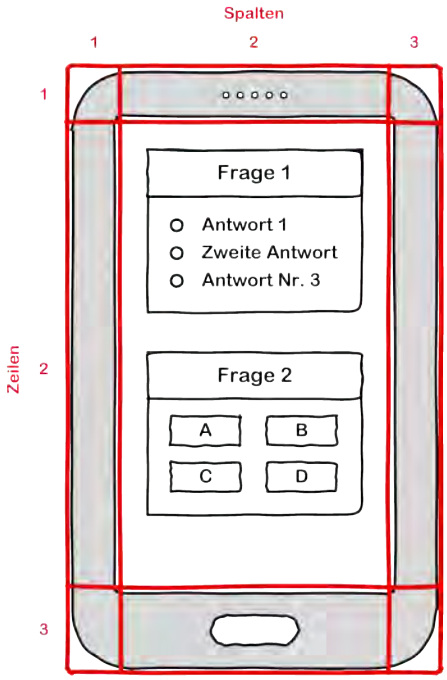


Abbildung 4.4: Geräte-Simulation mithilfe einer Tabelle

Vorteile:

- Vermeidet die Verwendung von zwei übereinanderliegenden Schichten.
- Ein Wechsel des Geräterahmens beeinflusst nicht den Inhalt, da der Aufbau nicht geschachtelt ist.
- Bei Änderung der Bildschirmgröße muss lediglich die gesamte Tabelle skaliert werden, wodurch sich die Größe sowohl des Rahmens als auch der Fragebogen-Seite entsprechend ändert.

Nachteile:

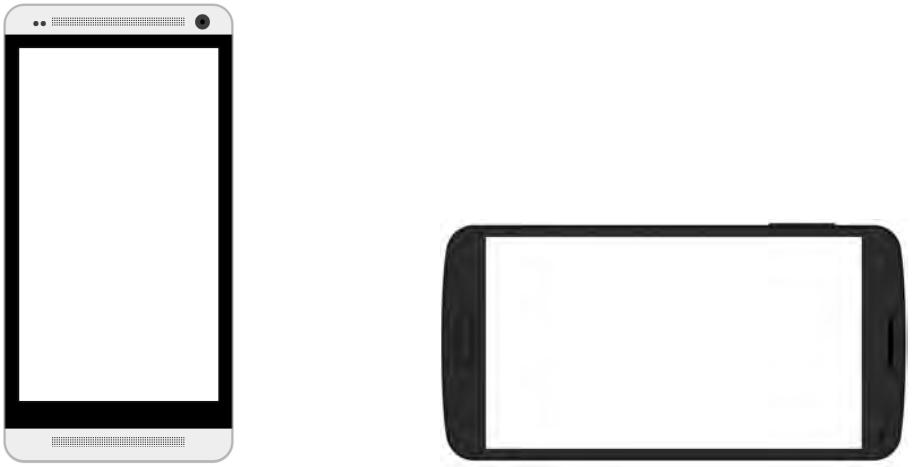
- Im Falle einer Grafik muss diese zuvor zugeschnitten und zerteilt werden. Im Falle einer HTML/CSS-Lösung müssen die zusammengehörigen Bruchstücke in der Implementierung exakt aufeinander angestimmt werden.
- In beiden Fällen kann es zu unterschiedlichem Verhalten der einzelnen Bruchstücke bei der Skalierung kommen, sodass die Schnittstellen sichtbar werden.

Abbildung 4.5: Symbolhaftes Mockup eines Smartphones

- Wird für den Rahmen eine Grafik verwendet, existieren weiterhin die Probleme der ersten Lösungsvariante *Gerät als Grafik im Hintergrund* bezüglich Auflösung und Verfügbarkeit der Bilder.

Diese Lösung stellt somit einen guten Kompromiss zwischen den Varianten 1 und 2 dar, da in der Theorie sowohl Skalierbarkeit als auch Unabhängigkeit einfach gewährleistet werden können. Jedoch ist diese Methode in der Praxis für den Einsatz in der Vorschau-Funktion ungeeignet, da ein Geräterahmen eine wesentlich komplexere Struktur darstellt als der Rahmen oder Hintergrund eines Buttons aus vorigem Beispiel zu *9-Patch-Grafiken* von Android. Darüber hinaus stellt es eine schlechte Praktik dar, Tabellen für layouttechnische Angelegenheiten ihrem eigentlichen Zweck zu entfremden. Aus diesem Grund fiel die Entscheidung für die Vorschau auf eine reine HTML/CSS-Lösung.

Da die Implementierung bereits eines einzelnen Geräts jedoch schon sehr viel Zeit beansprucht, stellt sich die Frage, ob solche Geräte-Simulationen mittels HTML und CSS bereits entwickelt wurden. Tatsächliche gibt es zahlreiche Mockups mobiler Geräte, die jedoch meist kein real existierendes Gerät wiedergeben, sondern nur symbolhaft gestaltet sind. Abbildung 4.5 ist ein Beispiel hierfür. Diese sind im Kontext dieser Arbeit jedoch ungenügend, da das Ziel verfolgt wird, dem Konfigurator-Nutzer eine möglichst realitätsnahe Vorschau zu bieten. Insbesondere sollen mehrere, real existierende Geräte dargestellt werden.



(a) HTC ONE im Portrait-Modus (b) SAMSUNG GALAXY S5 im Landscape-Modus

Abbildung 4.6: `devices.css`-Modelle zweier mobiler Geräte

Es gibt allerdings auch Projekte wie beispielsweise `devices.css` von MARVEL[8], welches nach aktuellem Stand immerhin 13 verschiedene Geräterahmen enthält sowie eine Rotationsfunktion für Portrait- und Landscape-Modus bietet (siehe Abbildung 4.6). Da das Angebot für die Vorschau passend und ausreichend ist, wird schlussendlich nun dieses Drittanbieter-Paket für die Simulation der mobilen Geräte in der Fragebogen-Vorschau verwendet.

4.4 Device Manager

Da die Templates der verschiedenen Geräte bei `devices.css` unterschiedlich in der HTML-Struktur sind, ist es unter Einhaltung der ANGULAR-Prinzipien unumgänglich für jedes Gerät eine eigene Komponente anzulegen. Für eine einfache Verwaltung und Erweiterbarkeit der in der Vorschau-Funktion unterstützten Geräte, werden diese ausschließlich über JSON-Objekte definiert. Listing 4.1 zeigt ein solches Objekt am Beispiel des APPLE IPHONE 8. Dabei enthält das JSON-Objekt eines Geräts eine Referenz zur zugehörigen Komponente (siehe Code-Zeile `4`). Dadurch kann der `DeviceManager`

[8] Demoseite des von MARVEL entwickelten `devices.css`-Projekts: https://marvelapp.github.io/devices.css/ – abgerufen am 13.06.2018

```
device = {
    "id": "apple_iphone8_1",              /* Relevant für Auswahl-Dropdown
                                             -> wird zur Laufzeit generiert */
    "component": AppleIphone8Component,   // Referenz zur Komponente
    "deviceType": DeviceType.SMARTPHONE,  // Typ des Geräts
    "manufacturer": Manufacturer.APPLE,   // Hersteller des Geräts
    "name": "iPhone 8",                   // Gerätebezeichnung
    "order": 0                            /* Möglichkeit zur Vorgabe einer
                                             Reihenfolge im Dropdown */
}
```

Listing 4.1: JSON-Objekt des Geräts APPLE IPHONE 8

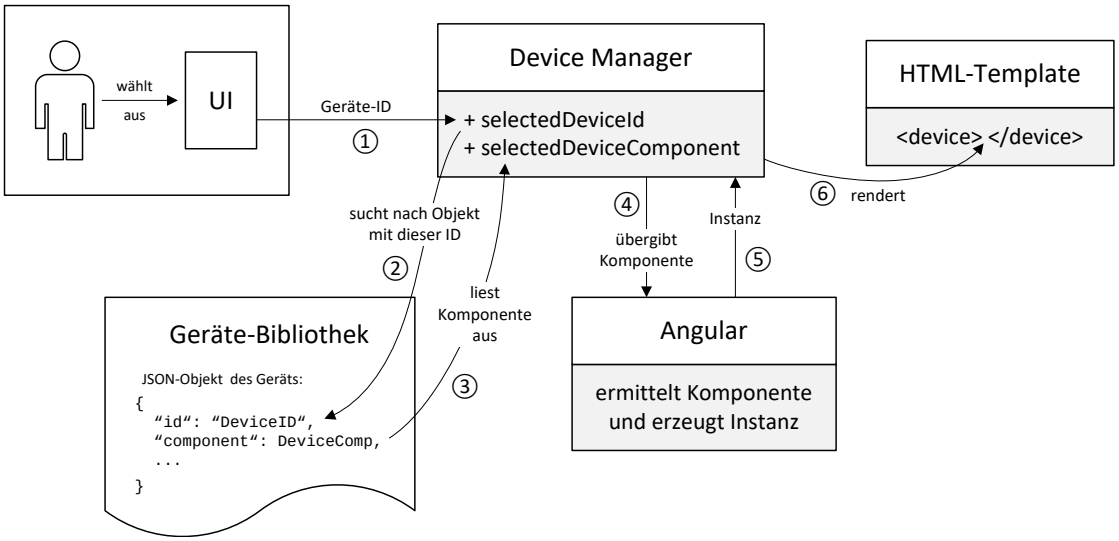


Abbildung 4.7: Interner Ablauf bei einem Gerätewechsel durch den Nutzer

mittels des von ANGULAR mitgelieferten `ComponentFactoryResolvers` dynamisch die Komponente ermitteln und an definierter Stelle einfügen und darstellen.

In Abbildung 4.7 ist der Vorgang bei einem Gerätewechsel veranschaulicht. Zuerst wird vom Benutzer im UI das neue Gerät ausgewählt ①. Über dessen ID wird in der Geräte-Bibliothek das entsprechende Objekt ermittelt ② und die zugehörige Komponente ausgelesen ③. Der Device Manager erzeugt mithilfe von ANGULAR eine Instanz dieser Komponente ④ – ⑤ und fügt sie an der definierten Stelle (hier die `device`-Direktive) im HTML-DOM ein ⑥.

```
node = {
    "category": "Question",                 // Kategorie des Elements
    "element": {                            // Element-Objekt
        "questionType": "MultipleChoice",   // Fragetyp (nur bei Frage-
        ...                                 //      elementen vorhanden)
    },
    ...
}
```

Listing 4.2: JSON-Objekt eines Knotens des Designer-Graphen

4.5 Dynamic Forms

Bislang wurde aus Gründen der besseren Verständlichkeit nur in abstrakter Form von einer `DynamicForms`-Komponente gesprochen. Tatsächlich handelt es sich dabei um ein Modul mit dem Namen `DynamicQuestionnaireForms` bestehend aus mehreren Komponenten. Der Begriff *Dynamic Forms* beschreibt eigentlich das von ANGULAR vorgestellte Prinzip[9] dynamisch Formulare aufzubauen. Die Bezeichnung `DynamicQuestionnaireForms` soll sowohl auf dieses Prinzip hindeuten, als auch darauf, dass dieses Modul explizit für das Fragebogen-System entwickelt wurde.

Die Komponenten des Moduls sind gemeinsam für die Analyse der Knoten verantwortlich, die vom Graph des Fragebogen-Designers übergeben werden, sowie für die Auswahl der entsprechenden Formular-Komponente des Fragebogen-Elements. Außerdem leiten sie die ausgewählte Sprache an die Formular-Komponenten weiter.

Die Auswertung der Knoten erfordert 3 Schritte:

1. Iteration über die Knoten-Liste,
2. Auswertung der Element-Kategorie und
3. Auswertung des Fragetyps, sofern es sich um ein Frageelement handelt.

Listing 4.2 gibt eine Übersicht über die Objekt-Struktur eines Knotens und seines zugehörigen Elements, welche für diese Auswertung erforderlich ist. Um die Aufgaben eindeutig zu trennen, werden für die drei Analyseschritte drei verschiedene Komponenten eingesetzt:

[9] ANGULARS *Dynamic Forms*: https://angular.io/guide/dynamic-form – abgerufen am 13.06.2018

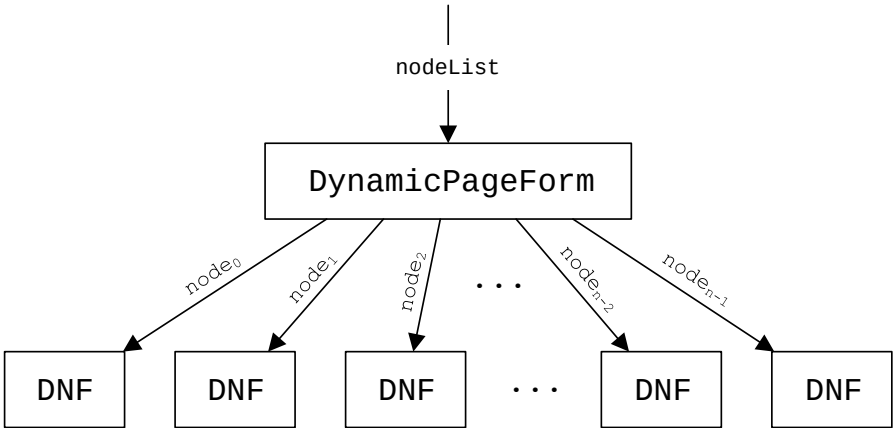


Abbildung 4.8: Iteration über Knotenliste in der `DynamicPageForm`-Komponente

- `DynamicPageForm`: Diese Komponente iteriert über die Knoten-Liste, welche die Fragebogen-Elemente der gewählten Seite enthält, und gibt die Knoten jeweils einzeln an die `DynamicNodeForm`-Komponente (DNF) weiter (siehe Abbildung 4.8).
- `DynamicNodeForm`: Hier wird nun anhand des `category`-Attributs des Knotens (`node`) die Kategorie des Elements ermittelt (siehe Abbildung 4.9). Es existieren die Kategorien *Headline, Text, Media, Question* und *Custom*. Im Falle eines Frageelements, wird das Objekt des `element`-Attributs des Knotens an die `DynamicQuestionForm`-Komponente (DQF) weitergegeben. Handelt es sich um ein Element einer anderen Kategorie, wird schon hier die entsprechende Formular-Komponente ausgewählt und das Element-Objekt daran weitergegeben.
- `DynamicQuestionForm`: Diese Komponente wertet das `questionType`-Attribut des Frageelements aus und ermittelt anhand dessen die zugehörige Formular-Komponente und gibt das Element-Objekt wiederum daran weiter (siehe Abbildung 4.10).

Die Element-Komponenten sind schließlich für die Darstellung und Eingabevalidierung zuständig. Insbesondere wählen sie anhand der übergebenen Sprache den anzuzeigenden Text aus den Übersetzungen aus, die im Element-Objekt gespeichert sind.

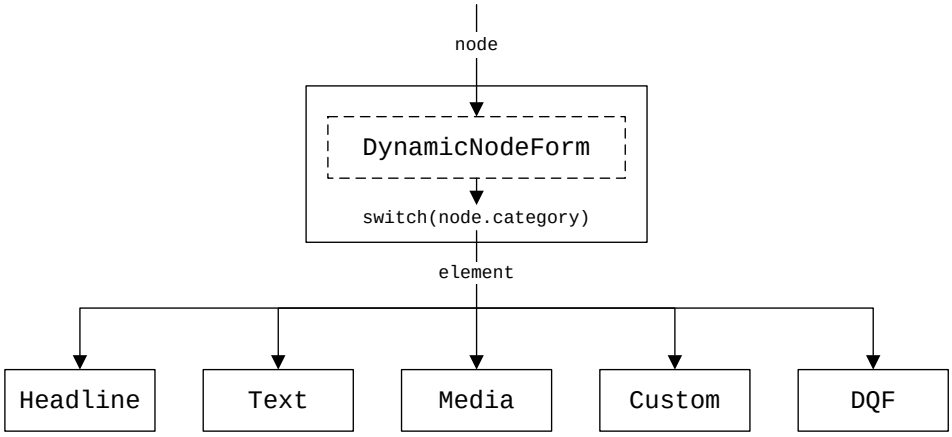


Abbildung 4.9: Auswertung der Element-Kategorie in der `DynamicNodeForm`-Komponente

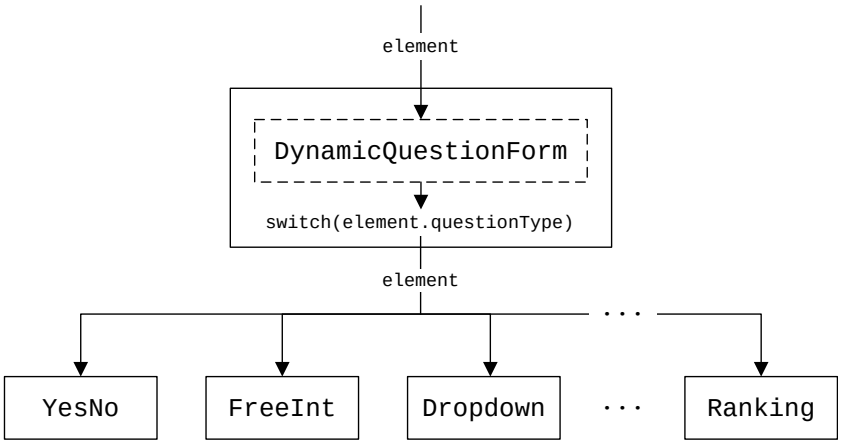


Abbildung 4.10: Auswertung des Fragetyps in der `DynamicQuestionForm`-Komponente

5

Erweiterungsmöglichkeiten

Wie bereits in Kapitel 3 beschrieben, ist die Erweiterbarkeit ein wichtiges Kriterium für das Vorschau-Modul. Denn insbesondere neue Fragebogen-Elemente werden im Zuge der Weiterentwicklung des Gesamtsystems höchstwahrscheinlich ergänzt werden, um Fachexperten das Erstellen noch besserer und präziserer Fragebögen zu ermöglichen. Weiterhin ist auch bei den simulierten Geräten die Wahrscheinlichkeit hoch, dass in Zukunft weitere hinzugefügt werden. Denn das Format und die Größe von Displays mobiler Geräte verändern sich stetig, womit auch die Unterstützung neuer Geräte erforderlich sein wird, um die Vorschau auf dem aktuellen Stand zu halten.

Durch die im Konzept entwickelte Architektur und die Realisierung mit ANGULAR bietet das Vorschau-Modul eine besonders einfache Erweiterbarkeit. Sowohl neue Fragen als auch zusätzliche Geräterahmen lassen sich in nur wenigen Schritten hinzufügen.

5.1 Fragebogen-Elemente

Im folgenden Abschnitt wird auf die Komplexität und Erweiterbarkeit der Fragebogen-Elemente in Bezug auf die Vorschau eingegangen. Dafür werden zunächst die vorhandenen Fragetypen vorgestellt und kategorisiert. Anschließend werden die Herausforderungen, die bei der Entwicklung der Visualisierung der Fragebogen-Elemente entstanden sind, am Beispiel zweier Fragetypen diskutiert. Schließlich wird die Vorgehensweise für eine effiziente Erweiterung der Vorschau um neue Fragebogen-Elemente erläutert.

5.1.1 Vorhandene Fragetypen

Die im Rahmen dieser Abschlussarbeit entwickelte Vorschau erleichtert dem Benutzer die Bedienung des Fragebogen-Konfigurators. Sie trägt durch die Visualisierung der Elemente bereits zum Zeitpunkt ihrer Erstellung dazu bei, den optimalen Fragetyp für eine gegebene Frage wählen zu können.

Aktuell existieren 19 Fragetypen im Konfigurator, die jeweils für eine bestimmte Art von Datenerfassung ausgelegt sind und sich dabei in 4 Kategorien einteilen lassen:

Auswahl

- *SingleChoice*: Auswahl genau einer aus beliebig vielen Antwortmöglichkeiten
- *YesNo*: Spezialfall von SingleChoice mit nur 2 Antwortmöglichkeiten („Ja” / „Nein”)
- *Matrix*: Tabellarischer Aufbau von SingleChoice-Fragen, wobei die Zeilen den Fragen entsprechen und die Spalten den Antwortmöglichkeiten
- *Dropdown*: Auswahl genau einer Antwort mithilfe eines Dropdown-Elements
- *MultipleChoice*: Auswahl einer oder mehrerer Antwortmöglichkeiten
- *Buttongrid*: Abwandlung von MultipleChoice, wobei die Antwortmöglichkeiten als Buttons dargestellt werden
- *SliderSingle*: Auswahl eines einzelnen Wertes innerhalb eines vorgegebenen Intervalls mithilfe eines Schiebereglers
- *SliderRange*: Auswahl eines Teilintervalls innerhalb eines vorgegebenen Intervalls mithilfe von zwei Schiebereglern
- *LikertScale*: SingleChoice in zeilenweiser Darstellung
- *LikertScaleWithImages*: Spezialfall von LikertScale mit Bildern statt Text zur Charakterisierung der Antworten

Freitext-Eingabe

- *FreeInt*: Freitext-Eingabe einer Ganzzahl

- *FreeFloat*: Freitext-Eingabe einer Dezimalzahl
- *FreeText*: Freitext-Eingabe von Stichwörtern
- *FreeTextarea*: Freitext-Eingabe eines Textes
- *FreeDate*: Freitext-Eingabe eines Datums

Bildung einer Rangfolge

- *Ranking*: Rangfolgen-Bestimmung ohne Gewichtung durch interaktive Anordnung vorgegebener Elemente
- *Distribution*: Rangfolgen-Bestimmung durch Vergabe von Gewichten

Bild-basierte Kennzeichnung

- *ImagePointAndClick*: Punktuelle Kennzeichnung auf einem vorgegebenen Bild
- *ImagePointAndDraw*: Kennzeichnung durch Freihandzeichnung auf einem vorgegebenen Bild

Einige von diesen Fragetypen stellten in der visuellen Umsetzung für die Vorschau besondere Herausforderungen dar.

5.1.2 Design-Entscheidungen

Die Herausforderungen zweier solcher Fragetypen werden hier beispielhaft diskutiert und die resultierenden Design-Entscheidungen dargelegt.

Buttongrid

Bei einer Buttongrid-Frage werden die Antwortmöglichkeiten als Buttons dargestellt und durch einen Klick darauf als ausgewählt markiert; durch einen weiteren Klick wird die Markierung wieder entfernt.

Die Länge des Antwort-Textes, der auf einem Button angezeigt wird, ist im Konfigurator nicht limitiert. Somit können sich die textuellen Inhalte der vorhandenen Buttons in ihrer Länge stark unterscheiden. Um eine optimale Darstellung zu gewährleisten, sollten die Buttons daher flexibel in ihrer Größe sein (siehe Abbildung 5.1a).

Dem gegenüber steht jedoch der psychologische Effekt, dass eine größere Fläche ein wichtigeres Element suggeriert. Laut diesem Aspekt sollten daher idealerweise alle Buttons die gleiche Größe haben. Dies würde im Umkehrschluss bedeuten, dass der Button mit dem längsten Inhalt die Größe aller Buttons vorgibt. Dies kann auf den Bildschirmen mobiler Geräte jedoch dazu führen, dass nur ein geringer Anteil aller Antwortmöglichkeiten gleichzeitig dargestellt werden kann, wodurch der Bearbeiter des Fragebogens einen schlechteren Überblick erhält. Zudem würden eventuell ungewollte Wortumbrüche stattfinden, falls ein Wort länger als die vorgegebene Breite des Buttons ist (siehe Abbildung 5.1b).

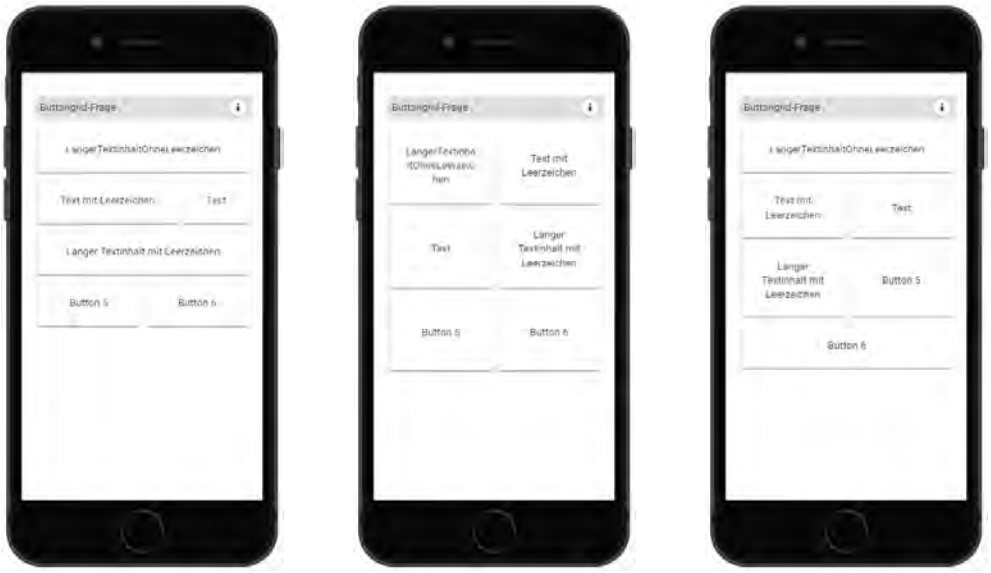
Aus Gründen der Darstellung auf mobilen Geräten und der Tatsache, dass der Button-Text vom Konfigurator-Benutzer frei definierbar ist, kann eine exakte Gitterstruktur mit gleich großen Buttons aus Usability-Gründen nicht gewährleistet werden. Der für die Vorschau gewählte Kompromiss (siehe Abbildung 5.1c) ist daher, Buttons derselben Zeile einerseits in ihrer Höhe einander anzupassen, andererseits für eine flexible, sich an den textuellen Inhalt anpassende Breite jedes Buttons zu sorgen. Diese wird wiederum minimiert, indem der Antworttext innerhalb eines Buttons umgebrochen wird (nur zwischen Wörtern). Für detaillierte layouttechnische Anpassungen ist schließlich der Konfigurator-Nutzer verantwortlich, indem er den Antworttext entsprechend variiert.

ImagePointAndDraw

Bei einer Frage vom Typ *ImagePointAndDraw* erhält der Bearbeiter des Fragebogens eine Abbildung, auf welcher er mithilfe einer Zeichenfunktion Bereiche markieren oder beliebige Objekte einzeichnen kann.

Das Problem bei der Entwicklung der Vorschau war hierbei, dass kein HTML-Element für Freihandzeichnungen existiert. Das Erstellen einer Zeichnung im Webbrowser ist mit primitiven Mitteln lediglich durch Manipulation des HTML5-Canvas-Elements via JavaScript möglich. In Kombination mit JavaScript-MouseEvents kann damit dann eine Freihandzeichnung implementiert werden.

Diese Methode ist jedoch nicht nur in ihrer Entwicklung aufwändig, sondern auch in der Wartung und Erweiterung für zusätzliche Funktionen. Da Freihandzeichnungen ein beliebtes und vielseitig verwendbares Feature in verschiedenen Domänen ist, existieren



(a) Größe aller Buttons ist vollständig flexibel und individuell

(b) Größe aller Buttons wird dem Button mit dem längste Inhalt angepasst

(c) Kompromiss: möglichst gleiche und kleine Größe; jedoch kein erzwungener Textumbruch

Abbildung 5.1: Darstellungsvarianten einer Buttongrid-Frage mit 6 Antwortmöglichkeiten

auch für Webbrowser bereits zahlreiche Implementierungen, die in den meisten Fällen ebenfalls das HTML5-Canvas-Element verwenden. Insbesondere existieren auch für ANGULAR entwickelte Varianten. Aus diesem Grund bietet es sich an, für die Vorschau ein solches Drittanbieter-Modul einzusetzen. Nach Analyse unterschiedlicher Plug-ins wird im Vorschau-Modul das Open-Source-Projekt NG2-CANVAS-WHITEBOARD[1] eingesetzt. Dieses bietet für eine bessere Bedienbarkeit der Freihandzeichnung auch die Möglichkeit einzelne Schritte rückgängig zu machen oder den gesamten Zeichenbereich zu leeren (wobei das Hintergrundbild unangetastet bleibt).

5.1.3 Abhängigkeiten

Es gibt mehrere Fragetypen, die wie *ImagePointAndDraw* Drittanbieter-Komponenten verwenden. Aktuell sind dies:

[1] Github-Repository von NG2-CANVAS-WHITEBOARD:
https://github.com/webfactorymk/ng2-canvas-whiteboard – abgerufen am 13.06.2018

- *ImagePointAndClick* / *ImagePointAndDraw*: NG2-CANVAS-WHITEBOARD (siehe voriger Unterabschnitt 5.1.2)
- *Ranking*: *bs-sortable* von NGX-BOOTSTRAP[2]
- *SliderSingle* / *SliderRange*: NG2-NOUISLIDER[3], welches wiederum von NOUISLIDER[4] abhängig ist
- Zahlreiche weitere Fragebogen-Elemente sind abhängig von Komponenten aus ANGULAR MATERIAL, welches bereits im Gesamtprojekt integriert ist (siehe Abschnitt 4.1).

Damit existieren Abhängigkeiten dieser Frage-Komponenten zu externen Modulen. Durch Einbindung dieser Module über NPM[5] stellt es jedoch keinen Mehraufwand dar, diese auf dem neusten Stand zu halten.

5.1.4 Neue Fragebogen-Elemente

Jedes Fragebogen-Element wird durch eine eigene Komponente verwaltet und dargestellt (siehe Abschnitt 3.3). Wie in Abschnitt 4.5 beschrieben, werden diese entweder durch `DynamicNodeForm` oder `DynamicQuestionForm` ausgewählt und gerendert. Aus Gründen besserer Datenkapselung und deutlicher Trennung der Aufgabenbereiche bilden die Element-Komponenten ein eigenes `ElementFormsModule` und sind nicht direkt im `DynamicQuestionnaireFormsModule` enthalten. Letzteres muss daher das `ElementFormsModule` importieren um die Element-Komponenten rendern zu können.

Soll nun beispielsweise die Vorschau für ein neues Element (einer der in Abschnitt 4.5 aufgeführten Kategorien) mit der Bezeichnung `NewElement` implementiert werden, müssen lediglich folgende 4 Schritte durchgeführt werden.

1. Eine Komponente `NewElementForm` muss dem `ElementFormsModule` hinzugefügt werden.

[2] Dokumentation von NGX-BOOTSTRAP: https://valor-software.com/ngx-bootstrap/ – abgerufen am 13.06.2018
[3] Demo-Seite von NG2-NOUISLIDER: http://tb.github.io/ng2-nouislider/ – abgerufen am 13.06.2018
[4] Demo-Seite von NOUISLIDER: https://refreshless.com/nouislider/ – abgerufen am 13.06.2018
[5] NPM: https://www.npmjs.com/ – abgerufen am 13.06.2018

2. Das `ElementFormsModule` muss die neu erstellte Komponente (aus 1.) exportieren, damit sie im `DynamicQuestionnaireFormsModule` sichtbar ist.

3. Handelt es sich bei dem Element nicht um einen neuen Fragetyp, muss die Komponente `NewElementForm` das Interface `ElementForm` implementieren. Ist das neue Element von der Kategorie *Question* muss es das Interface `QuestionForm` implementieren, welches eine Spezialisierung von `ElementForm` ist. Das jeweilige Interface gibt die benötigten Klassenvariablen vor.

4. Nun muss die neue Komponente noch einen Eintrag im HTML-Template entweder der Komponente `DynamicQuestionForm` (Element hat die Kategorie *Question*) oder der Komponente `DynamicNodeForm` (bei allen anderen Kategorien) erhalten. Man beachte dabei die zu übergebenden Parameter.

In allen Schritten kann sich leicht an den bereits existierenden Vorschau-Elementen orientiert werden. Neue Fragen oder andere Elemente lassen sich im Vorschau-Modul demnach mit sehr geringem Aufwand einpflegen. Der größte Aufwandsfaktor ist vielmehr die Implementierung der visuellen Darstellung, was wiederum vom Wesen und der Komplexität des Elements abhängt. Diese Implementierung muss über das HTML-Template und das CSS-Stylesheet der Element-Komponente erfolgen.

5.2 Geräterahmen

Im Folgenden wird eine Richtlinie vorgegeben, nach der die vorhandene Geräteauswahl einfach und effizient um neue Geräte erweitert werden kann.

Die implementierte Geräteauswahl hängt vollständig von dem Drittanbieter-Produkt `devices.css` ab. Da jedes darin verfügbare Gerät eine andere Struktur im HTML-Code aufweist, erfordert dies eine jeweils eigene Komponente pro Gerät. Wie in Abschnitt 4.4 beschrieben, können die Geräte-Komponenten dynamisch ermittelt und mithilfe von ANGULAR gerendert werden. Diese Architektur ermöglicht es, neue Geräte mit geringst möglichem Aufwand hinzuzufügen und dies insbesondere ohne nähere Kenntnisse von der Implementierung des Gerätewechsels.

Soll also beispielsweise ein neues Gerät mit der Bezeichnung `NewDevice` des Herstellers `ExampleManufacturer` zur Vorschau hinzugefügt werden, müssen hierbei die folgenden Schritte durchgeführt werden. Es wird dabei vorausgesetzt, dass die Implementierung der visuellen Darstellung bereits vom Drittanbieter-Modul `devices.css` bereitgestellt wird.

1. Zunächst muss nach festgelegter Namenskonvention die Geräte-Komponente `ExampleManufacturerNewDevice` im `DeviceFrameModule` angelegt werden.
2. Im `DeviceFrameModule` muss die Komponente als `EntryComponent` deklariert werden, damit ANGULAR diese später dynamisch erzeugen kann.
3. Im HTML-Template von `ExampleManufacturerNewDevice` muss das entsprechende HTML-Fragment eingefügt werden. Dieses kann direkt von `devices.css` bezogen werden.
4. Zuletzt muss für die neue Geräte-Komponente noch ein JSON-Objekt (siehe Listing 4.1) der Liste `DEVICES` im `DeviceModel` hinzugefügt werden, welches wichtige Konfigurations-Parameter, wie die Gerätebezeichnung oder die Komponente des Geräts, beinhaltet.

Durch die dynamische Implementierung der Geräteauswahl sind keine weiteren Schritte erforderlich. Auch hier können weitere Details zu den einzelnen Schritten den bereits angelegten Geräte entnommen werden.

Die in Kapitel 3 entwickelte Architektur und deren Umsetzung in Kapitel 4 erlauben zusammenfassend also eine einfache Erweiterbarkeit der Hauptfunktionalitäten (Gerätesimulation und Fragebogen-Visualisierung). Dies wird insbesondere durch dynamisches Laden der benötigten Inhalte sowie durch minimale Schnittstellen zwischen den Komponenten der Vorschau erreicht.

6

Verwandte Arbeiten

Wie in Kapitel 2 beschrieben verfolgt der Konfigurator mit dem WYSIWYM-Paradigma ein ähnliches Konzept wie LaTeX. Da häufig auch bei LaTeX-Dokumenten während der Bearbeitung viele Male das Zwischenergebnis betrachtet werden muss, haben zahlreiche Editoren den Kompilierungsprozess optimiert, um eine schnellere Vorschau bieten zu können. Inzwischen existieren sogar Editoren mit Live-Vorschau, wie beispielsweise OVERLEAF [7], welche dauerhaft den TeX-Code analysieren und bei fehlerfreier Syntax versuchen diesen zu kompilieren und anzuzeigen.
Einfacher gestaltet sich die Vorschau bei der Auszeichnungssprache *Markdown* [8], da hier keine Kompilierung, sondern lediglich eine Konvertierung des Textes nach HTML erforderlich ist und es dabei zu keinen Syntaxfehlern kommen kann.
Ein dem Konfigurator ähnliches Produkt ist KETTLE SPOON [9]. Dieses bietet Möglichkeiten zur Integration und Transformation komplexer Daten aus zahlreichen Quellen mithilfe von Ansätzen aus der graphischen Endbenutzer-Programmierung. Auch hier gibt es eine der Vorschau ähnlichen Funktion, welche es erlaubt nur einen Teilausschnitt der gesamten Datenoperationen eines Vorgangs auszuführen. Dies ist insbesondere deshalb sinnvoll, da in den meisten Fällen der Datenumfang viel zu hoch ist um darauf Testoperationen wiederholt durchführen zu können.

Bei Anwendungen, die mehr dem WYISWYG-Paradigma folgen, entspricht die Vorschau meist dem Editor selbst. Dabei wird auf dem sichtbaren Modell gearbeitet und der entsprechende Code im Hintergrund automatisch erzeugt. Ein Beispiel für ein solches Vorgehen ist das Grafikformat SVG [10]. Anders als bei rasterbasierten Bildformaten wie JPEG oder PNG besteht hier die Grafik im Hintergrund aus einer XML-Struktur, die jedoch normalerweise nicht betrachtet wird. Gearbeitet wird nur auf der visuellen Ebene,

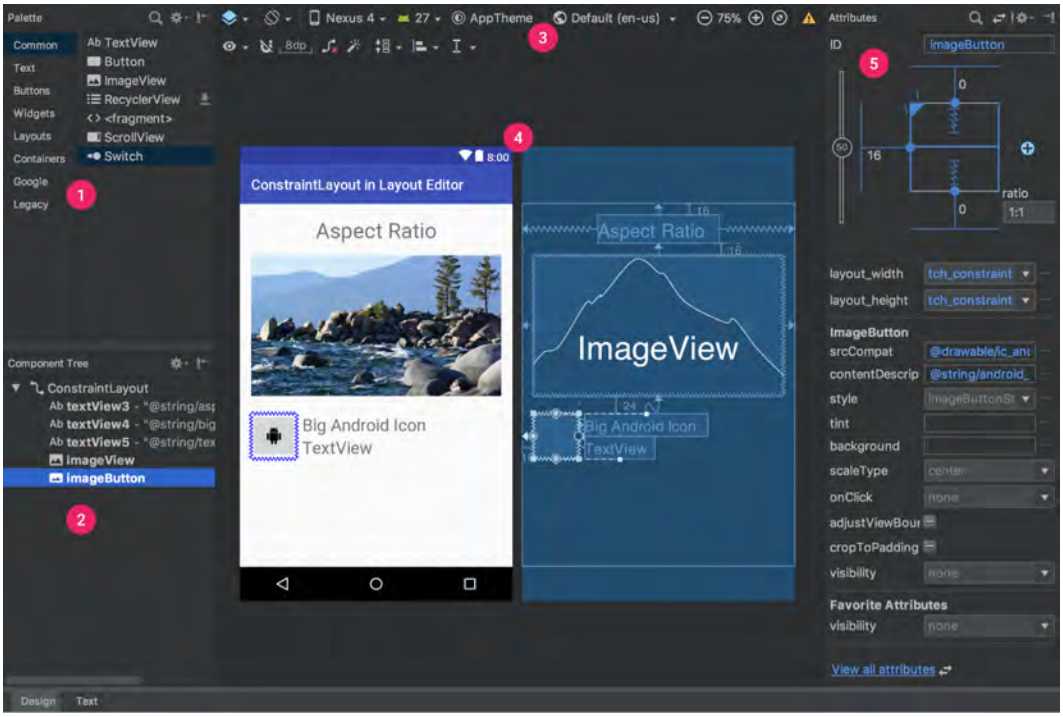


Abbildung 6.1: Layout-Editor von ANDROID STUDIO [11]

womit stetig eine Vorschau des aktuellen Zwischenergebnisses sichtbar ist.
Es existieren jedoch auch Mischvarianten der beschriebenen Konzepte wie beispielsweise der Layout-Editor von ANDROID STUDIO [11]. Hier kann sowohl auf der graphischen Ebene, als auch auf der Code-Ebene gearbeitet werden, wobei zu jedem Zeitpunkt zwischen den beiden Ebenen gewechselt werden kann. Der graphische Teil ist somit Vorschau des Code-Resultats und Editor zugleich (siehe 6.1). Insbesondere ist dieser Layout-Editor auch ein Beispiel für die Vorschau von Anwendungen in mobilen Geräten, da er zur Erstellung von Android-Anwendungen dient und für deren Darstellung das Layout von Gerätebildschirmen simuliert. Dabei sind verschiedene Gerätegrößen auswählbar, um dem Nutzer die tatsächlichen Größenverhältnisse besser erkennbar machen zu können.

Für die Entwicklung responsiver Webanwendungen bieten moderne Webbrowser wie GOOGLE CHROME [12], MOZILLA FIREFOX [13] oder MICROSOFT EDGE [14] spezielle Ansichten für mobile Geräte. Anders als die Vorschau des Fragebogen-Konfigurators simulieren sie dabei jedoch keine Geräterahmen, sondern fokussieren sich auf die Anpas-

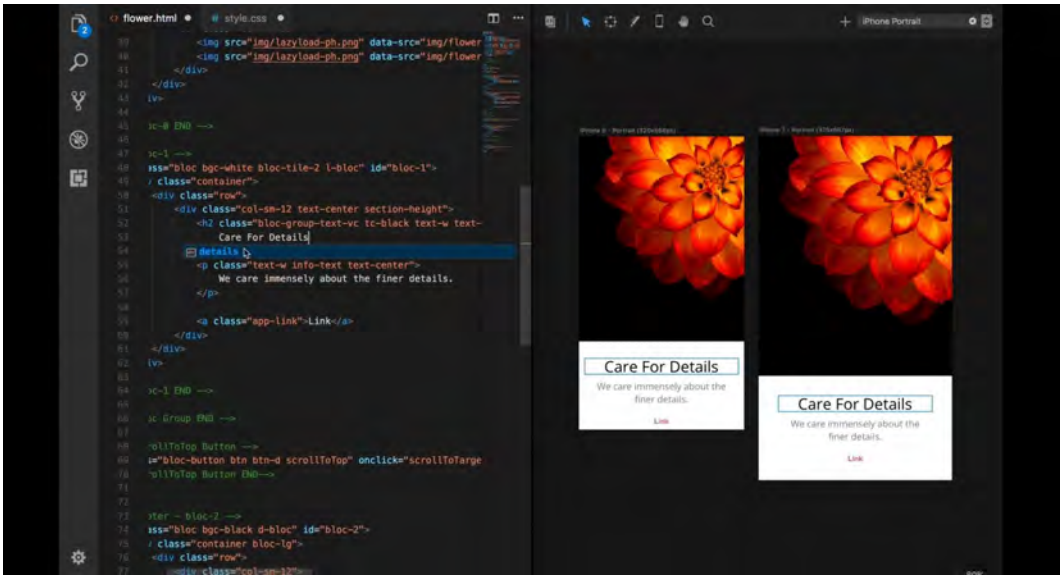


Abbildung 6.2: Live-Vorschau-Funktion von SOLIS [15]

sung der Displaygröße und die Simulation sonstiger Eigenschaften mobiler Geräte (z.B. GPS-Daten). In ähnlicher Weise bietet auch SOLIS FOR MAC [15] eine Live-Vorschau einzelner Webseiten für mobile Geräte (siehe 6.2). SOLIS ist ein eigenständiges Plug-in für verschiedene Webentwickler-Editoren, womit kein Webbrowser für die Ausführung des HTML-, CSS- oder JavaScript-Codes notwendig ist. Zusätzlich bietet es innerhalb der Vorschau direkten Zugriff auf die WebKit-Entwickler-Tools.

Simulationen realer mobiler Geräte existieren momentan hauptsächlich in den Bereichen von Mockup und Prototyping. Meist werden dabei auf statischer Ebene Screenshots in die gewünschten Geräterahmen eingefügt. Ein Beispiel hierfür ist die Webanwendung MOCKUPHONE [16]. Teilweise werden in die simulierten Geräterahmen auch Wireframes eingebettet, um ein authentischeres Modell mobiler Anwendungen zu erhalten. Die Gerätesimulationen, welche in die im Rahmen dieser Arbeit entwickelten Vorschau integriert wurden, verwendet MARVEL selbst in seinem Hauptprodukt MARVELAPP [17].

7

Fazit

Aufgrund fehlender Möglichkeiten in der Konfigurator-Anwendung den Fragebogen-Inhalt auf visueller Ebene anzupassen, wurde im Rahmen dieser Abschlussarbeit eine Vorschau-Funktion entwickelt, welche die Darstellung des Fragebogens auf unterschiedlichen mobilen Endgeräten simuliert. Durch die Auswahlmöglichkeit aktueller Geräterahmen wird eine flexible und effiziente Anpassung der Fragebogen-Inhalte gewährleistet. Außerdem wurde gezeigt, dass die Vorschau durch ihre modulare und generische Implementierung gut um neue Komponenten erweiterbar ist.

Insbesondere die Art und Weise des Aufbaus der Geräteauswahl ist von großem Vorteil in Bezug auf deren Erweiterbarkeit. Konkret ist damit die dynamische Ermittlung und Instanziierung der Geräte-Komponenten gemeint. Eine solche Struktur wäre auch bei den Fragebogen-Elementen denkbar. Die damit erreichte Aufwandseinsparung beim Hinzufügen neuer Elemente rechtfertigt zwar keine Neuimplementierung dieses Aspekts der Vorschau, jedoch ist es durchaus sinnvoll, diese generische Vorgehensweise bei zukünftigen Entwicklungen zu berücksichtigen. Insbesondere da ANGULAR seit Version 6.0 mit *Angular Elements*[1] ein neues und vereinfachtes Verfahren liefert, Komponenten zur Laufzeit in das bestehende HTML-DOM einzufügen und wieder zu entfernen.

Des Weiteren erlaubt der modulare Aufbau der Vorschau eine direkte Wiederverwendung des `DynamicQuestionnaireFormsModules`, welches für die Darstellung des Fragebogens zuständig ist. Somit könnte künftig ein Web-Client analog zu den mobilen Endanwendungen entwickelt werden, welcher dieses Modul für die Bearbeitung veröffentlichter Fragebögen wiederverwendet und damit die Datenerfassung direkt über Webbrowser ermöglicht. Innerhalb dieses Moduls muss dafür lediglich die Eingabevalidie-

[1] *Angular Elements*: https://angular.io/guide/elements – abgerufen am 13.06.2018

rung der Fragebogen-Elemente noch implementiert werden. Auf diese wurde aufgrund fehlender Anforderungen und Mittel im Rahmen der vorliegenden Arbeit verzichtet.

Die vorgestellte Implementierung der Vorschau-Funktion baut auf *Angular 5* auf, da dies zu Beginn dieser Arbeit die aktuellste Version des Frameworks darstellte. Mit *Angular 6* erschien am 03. Mai 2018 ein neuer Major-Release dieses Frameworks. Die Migration von Version 5 auf Version 6 ist empfehlenswert, da damit einige nützliche Funktionen und Komponenten, wie beispielsweise die oben erwähnten *Angular Elements*, hinzukommen. Die größte Neuerung ist die neue ANGULAR CLI-Funktion `ng update`, welche das automatische Aktualisieren externer Module und Bibliotheken, von denen das Projekt abhängig ist, stark vereinfacht.
Für die Vorschau-Funktion besteht die einzig signifikante Änderung in der Art und Weise, wie *Angular Services* und *Angular Modules* miteinander verknüpft werden. Bis *Angular 5.2* wurde ein Service lediglich als `@Injectable` annotiert und jedes Modul, welches diesen Service benötigte, musste diesen als `Provider` innerhalb der `@NgModule`-Annotation registrieren. Ab *Angular 6* hat sich die Vorgehensweise umgekehrt. Ein Service muss nun innerhalb seiner `@Injectable`-Annotation jedes Modul referenzieren, in welchem dieser verwendet wird. Ein zusätzlicher Eintrag auf Seiten der Module ist dabei nun nicht mehr notwendig. Diese Methode soll zur Reduzierung der Codemenge nach der Kompilierung beitragen, indem Services nur dort in Module eingefügt werden, wo sie auch verwendet werden. Weitere Informationen zu den Neuerungen in *Angular 6* sind in einem offiziellen Blog-Post[2] von ANGULAR zur finden.

[2] *„Version 6 of Angular Now Available"*: https://blog.angular.io/version-6-of-angular-now-available-cc56b0efa7a4 – abgerufen am 13.06.2018

Abbildungsverzeichnis

Literaturverzeichnis

[1] Schobel, J., Pryss, R., Schickler, M., Reichert, M.: A Configurator Component for End-User Defined Mobile Data Collection Processes. In: Demo Track of the 14th International Conference on Service Oriented Computing (ICSOC 2016). (2016)

[2] Schobel, J., Pryss, R., Wipp, W., Schickler, M., Reichert, M.: A Mobile Service Engine Enabling Complex Data Collection Applications. In: 14th International Conference on Service Oriented Computing (ICSOC 2016). Number 9936 in LNCS (2016) 626–633

[3] Schobel, J., Pryss, R., Schickler, M., Ruf-Leuschner, M., Elbert, T., Reichert, M.: End-User Programming of Mobile Services: Empowering Domain Experts to Implement Mobile Data Collection Applications. In: 5th IEEE International Conference on Mobile Services (MS 2016), IEEE Computer Society Press (2016) 1–8

[4] Dadam, P., Reichert, M., Rinderle-Ma, S., Lanz, A., Pryss, R., Predeschly, M., Kolb, J., Ly, L.T., Jurisch, M., Kreher, U., Göser, K.: From ADEPT to AristaFlow BPM Suite: A Research Vision Has Become Reality. In Rinderle-Ma, S., Sadiq, S., Leymann, F., eds.: Business Process Management Workshops, Berlin, Heidelberg, Springer Berlin Heidelberg (2010) 529–531

[5] Lamport, L.: LaTeX: A Document Preparation System, 2/e. Pearson Education India (1994)

[6] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (XML) 1.1. World Wide Web Consortium. Rec-xml11-20060816 edn. (2006)

[7] Overleaf: Real-time Collaborative Writing and Publishing Tools with Integrated PDF Preview. `https://www.overleaf.com/` (2018) – Abgerufen am: 13.06.2018.

[8] Leonard, S.: The text/markdown Media Type. RFC 7763, Internet Engineering Task Force (2016)

[9] Belem, D.: Data Integration - Kettle | Hitachi Vantara Community. https://community.hitachivantara.com/docs/DOC-1009855 (2018) – Abgerufen am: 13.06.2018.

[10] Andronikos, N., Atanassov, R., Bah, T., Bellamy-Royds, A., Birtles, B., Brinza, B., Concolato, C., Dahlström, E., Lilley, C., McCormack, C., Schepers, D., Schulze, D., Schwerdtfeger, R., Takagi, S., Watt, J.: Scalable Vector Graphics (SVG) 2. World Wide Web Consortium. (2016)

[11] Android Developers: Build a UI with Layout Editor | Android Developers. https://developer.android.com/studio/write/layout-editor (2018) – Abgerufen am: 13.06.2018.

[12] Bakaus, P.: Simulate Mobile Devices with Device Mode | Tools for Web Developers | Google Developers. https://developers.google.com/web/tools/chrome-devtools/device-mode/ (2018) – Abgerufen am: 13.06.2018.

[13] Mozjan: Bildschirmgrößen testen - Firefox Tools für Webentwickler | MDN. https://developer.mozilla.org/de/docs/Tools/bildschirmgroessen-testen (2018) – Abgerufen am: 13.06.2018.

[14] Doyle Navara, E.: Microsoft Edge DevTools - Emulation - Microsoft Edge Development. https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide/emulation (2018) – Abgerufen am: 13.06.2018.

[15] Solis: Solis For Mac - A Live Design Output that integrates seamlessly with your favourite code editor for true live code previewing. https://solisapp.com/ (2018) – Abgerufen am: 13.06.2018.

[16] MockUPhone: MockUPhone - Free and simple screenshots device mockups generator. https://mockuphone.com/ (2018) – Abgerufen am: 13.06.2018.

[17] MarvelApp: Marvel – Making design simple for everyone. https://marvelapp.com/ (2018) – Abgerufen am: 13.06.2018.

Name: Lenard Funk Matrikelnummer: 869299

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 14.06.2018 Lenard Funk

Lenard Funk