Responsive Web Widgets – Darstellung komplexer interaktiver Elemente von Webseiten auf unterschiedlichen Endgeräten

Marvin Schwoiger

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2013

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 30. Juni 2013

Marvin Schwoiger

Inhaltsverzeichnis

Kurzfassung

Mobile Geräte mit berührungsempfindlichen Bildschirmen, Desktop-Computer, Smart TVs und Spielekonsolen werden für den Zugriff auf das Internet verwendet. Der Responsive Web Design Ansatz versucht, Darstellungsformen von Webapplikationen zu finden, die auf all diesen Geräten anzeigbar sind. Die diversen Interaktionsmethoden zum Steuern der Geräte werden dabei außer Acht gelassen.

Das Editieren von Daten innerhalb einer Webapplikation ist für die Verwendung von Desktop-Umgebungen optimiert. Beim Ändern der Daten einer Webapplikation unter Verwendung eines Smartphones sinkt der Benutzerkomfort, da die verlangten Eingabemechanismen nicht für die vorhandenen Eingabemethoden vorgesehen sind.

In dieser Arbeit werden die Voraussetzungen zum Entwickeln eines Responsive Web Widgets betrachtet. Das Web Widget basiert auf den Methoden des Responsive Web Designs zur Anpassung auf verschiedenen Bildschirmgrößen. Außerdem werden Unterschiede der Eingabemethoden von Desktop-Computern und Geräten mit berührungsempfindlichen Bildschirmen beschrieben. Der Schwerpunkt liegt in der Bearbeitung von Tabellendaten auf kleinen Bildschirmen.

Das Ziel ist, eine Steigerung des Benutzerkomforts gegenüber dem Standardverhalten von mobilen Webbrowsern zu erreichen. Für diesen Zweck wird ein Responsive Web Widget Prototyp entwickelt, der die Daten einer HTML-Tabelle auf kleinen Bildschirmen als Diagramm darstellt. Die Darstellung der Daten als Diagramm spart Platz ein und bietet Interaktionsflächen, die mit Gesten intuitiv zu bedienen sind. Die Daten werden auf kleinen Bildschirmen zugänglich aufbereitet. Dies reduziert die Anzahl an benötigten Interaktionen bei der Erkundung und Bearbeitung der Daten durch einen Anwender.

Abstract

Mobile devices with touchscreens, desktop computers, Smart TVs and game consoles enable access to information from the Internet. The Responsive Web Design approach tries to adapt the representations of web applications on each of these devices, but their various interaction methods are not considered in the adaption of the layout.

Workflows for editing data within web applications are optimized for desktop computers. The given input mechanisms of web applications are not designed for the input methods available on mobile devices, such as smartphones. Therefore, an enhancement of usability for editing data of web applications on smartphones is required.

This thesis describes the development of a Responsive Web Widget. The Widget applies methods of the Responsive Web Design approach to adapt the layout of a website to various screen sizes. The input methods of desktop computers and touch devices are described additionally. This approach focuses on the editing of table data on small displays. The goal is to increase usability in contrast to the standard behaviour of mobile browsers.

Therefore, a Responsive Web Widget prototype was developed. The prototype displays data of an HTML table in a chart. Displaying a chart saves space on the screen and offers interaction areas for intuitive touch gestures. Hence data is presented accessibly on small screens. Consequently, this saves interactions when a user wants to explore or edit table data.

Kapitel 1

Einleitung

Die Verbreitung von mobilen, internetfähigen Geräten nimmt stetig zu. Die Zahl an verkauften Smartphones und Tablet-PCs übersteigt die Anzahl an verkauften Desktop-Computern. Viele Hersteller von Unterhaltungselektronik versuchen, den aufstrebenden Markt zu ihrem Vorteil zu nutzen. Aus diesem Grund sind verschiedene Geräte mit unterschiedlichen Betriebssystemen im Umlauf. Für die Betriebssysteme müssen Applikationen in unterschiedlichen Programmiersprachen entwickelt werden: native Applikationen für *Android* müssen in Java, für *iOS* in Objective-C und für *Windows Phone* in C# verfasst sein, um sie auf dem dafür vorgesehenen Betriebssystem installieren zu können. Damit eine Applikation für jede Plattform erhältlich ist, müssen ihre Funktionen in den diversen Programmiersprachen umgesetzt werden. Erschwerend kommt hinzu, dass verschiedene Typen von Geräten auf dem Markt sind. Einige Geräte erkennen Multi-Touch-Gesten, andere werden über Tasten bedient, und wieder andere haben kompakte Bildschirmabmessungen.

Der Ansatz der Responsive Web Widgets umgeht, unter Verwendung von Webstandards, die Limitierung einer Applikation auf ein Betriebssystem. Basierend auf dem Responsive Web Design Ansatz wird die Benutzerschnittstelle eines Responsive Web Widgets an die Bildschirmgröße des verwendeten Gerätes angepasst. Es werden Konzepte erarbeitet, die den Bedienungskomfort des Anwenders[1] durch Verwendung der Eingabemethoden des Betrachtungsgerätes erhöhen. Diese Arbeit hat ihren Schwerpunkt in der Entwicklung eines Responsive Web Widgets Prototypen. Der Prototyp gestaltet die Darstellung von HTML-Tabellen auf kleinen Bildschirmen benutzerfreundlicher. Außerdem wird ein Konzept entwickelt, das die Bearbeitung der Tabellendaten anhand der vorhandenen Eingabemethoden des Betrachtungsgerätes unterstützt.

Die Ziele sind das Standardverhalten eines mobilen Webbrowsers bei der

[1] Zugunsten der einfacheren Lesbarkeit wird sowohl für die männliche als auch die weibliche Form, die männliche Form verwendet.

Darstellung von HTML-Tabellen zu verbessern und dabei den Bedienungskomfort zu erhalten und gerätespezifisch anzupassen. Die Erarbeitung der Konzeption ist in den anschließenden Kapiteln behandelt.

Das folgende Kapitel beschreibt die Eigenschaften eines Web Widgets. Nach der Begriffserklärung werden JavaScript-Bibliotheken zur Erstellung von Web Widgets vorgestellt. Die Beschaffenheiten der Bibliotheken setzen unterschiedliche Konfigurationsmechanismen der bereitgestellten Web Widgets voraus. Die Web Widgets werden entsprechend ihrer Funktion kategorisiert. Danach werden Lösungen behandelt, die speziell für die Darstellung von Tabellen auf kleinen Bildschirmen ausgerichtet sind. Nicht jeder dieser Lösungsansätze ist mit einer Web-Technologie umgesetzt, jedoch bieten sie einen Einblick in Methoden, die für eine platzsparende Darstellung einer Tabelle hilfreich sind.

Das darauf folgende Kapitel behandelt relevante Technologien, die für die Entwicklung des Responsive Web Widgets von Bedeutung sind. Es beschreibt das SVG-Grafikformat und das Canvas-Element, die das dynamische Erstellen von Web-Grafiken ermöglichen. Anschließend werden Vergleiche zwischen dem Vektorgrafikformat und dem pixelbasierten Canvas-Element gezogen. Danach werden Media Queries vorgestellt, die zur dynamischen Anpassung von Webseiten-Layouts verwendet werden. Darauf folgt die Erklärung der Viewport-Metainformation. Die Viewport-Metainformation beeinflusst das Darstellungsverhalten von mobilen Webbrowsern.

Im Anschluss werden im nächsten Kapitel Konzepte für die Entwicklung des Responsive Web Widgets ausgearbeitet. Diese werden anhand von Anforderungen definiert. Die Unterschiede von Mouse- und Touch-Events werden aufgezeigt. Die Visualisierung von Daten setzt bestimmte Funktionen eines Web Widgets voraus, auf die im Verlauf des Kapitels eingegangen wird. Außerdem wird das Benutzerverhalten bei der Informationssuche in Tabellen beschrieben.

Aufbauend auf den Technologien, die für die Entwicklung des Responsive Web Widgets geeignet sind, wird die Implementierung erklärt. Die Umsetzung ist ohne JavaScript verwendbar, da sie das *Unobtrusive JavaScript* Prinzip realisiert. Als Architektur wird das MVVM-Paradigma benutzt. Die verwendeten Bibliotheken *Knockout*, *Raphaël* und *Hammer* sind verantwortlich für die Code-Strukturierung, das Zeichnen von SVGs und das Erkennen von Gesten. Struktur-, Präsentations- und Verhaltensschicht der Responsive Web Widgets werden separat behandelt.

Das letzte Kapitel beschreibt die Evaluierung des Responsive Web Widget Prototyps. Das Erreichen der Anforderungen der Konzeption wird begutachtet. Danach wird der Ablauf der durchgeführten Benutzerstudie erklärt. Die Ergebnisse der Studie und die daraus resultierenden Verbesserungsempfehlungen werden zusammengefasst. Zum Schluss werden Erweiterungsmöglichkeiten genannt.

Kapitel 2

Stand der Technik

Das folgende Kapitel erörtert die Definition eines Web Widgets. Es vermittelt einen Überblick von verschiedenen Web Widget Arten und behandelt deren Einsatzmöglichkeiten. Außerdem werden Bibliotheken zur Erstellung und Verwendung von Web Widgets vorgestellt. Der Schwerpunkt des Kapitels liegt auf dem Beschreiben von verschiedenen Ansätzen zur Tabellendarstellung auf kleinen Bildschirmen, da ein wesentlicher Teil der Arbeit auf diesen Erkenntnissen beruht.

2.1 Web Widgets

Web Widgets bereichern konventionelle Webseiten mit Funktionen, die dem Betrachter ein komfortables Nutzererlebnis bieten. Web Widgets sind GUI[1]-Komponenten, die in Webseiten eingebettet werden und in ihrer Komplexität und in der Art ihres Aufgabenbereiches variieren. Ein Web Widget ist auf eine bestimmte Aufgabe ausgelegt, deren Abarbeitung abhängig oder unabhängig zu dem restlichen Angebot der Webseite erfolgt. Da ein Web Widget Teil einer Webseite ist, wird es in einem Webbrowser ausgeführt und aus gängigen Web-Technologien entwickelt. Web Widgets sind vor allem durch die Verwendung in Blogs und Social-Networks bekannt. Ein W3C[2]-Entwurf zum Standardisieren von Web Widgets spezifiziert diese in [14, Abschnitt 3.1] folgendermaßen:

> *Web widgets (also known as modules or badges) are fragments of HTML, CSS, and ECMAScript (or possibly an Adobe Flash movie) that are either declaratively or dynamically included into a Web document. A common example of Web widgets is one that downloads a set of icon-sized images from a photo-sharing Web site and displays those images as a slide-show based on a*

[1] **G**raphical **U**ser **I**nterface, dt: grafische Benutzerschnittstelle
[2] World Wide Web Consortium

> *set of user preferences (eg. the images tagged 'vacation Italy'); such Web widgets are commonly seen embedded into social networking Web sites and blogs. [...] Unlike widgets, Web widgets a hosted on the server-side and are embedded into HTML documents prior to being served to the client. The creation of a Web widget usually involves having an author specify, in XML or some other format (eg. PHP), what the widget does and which APIs the Web widget depends on.*

D. h., das W3C versteht Web Widgets als Module, die auf eine API[3] zugreifen und somit auf die Response von einem Server angewiesen sind. Eine weitere Definition von Web Widgets ist in [2] beschrieben:

> *[Web Widgets are] small applications that are embedded in web sites and executed in web browsers, which makes them platform independent and usable without installation.*

Hierbei werden Web Widgets als eigenständige Applikationen angesehen, die in einem Webbrowser ausgeführt werden. Laut dieser Definition werden Web Widgets clientseitig ausgeführt. Sie sind nicht als unterstützende Programmteile der Webseite anzusehen, sondern als eigenständige Software deren Laufzeitumgebung der Webbrowser ist. Dazu steht im Widerspruch die W3C-Definition in der Web Widgets vorrangig über APIs anzusprechen sind und einen Server zur Kommunikation benötigen.

Eigendefinition: In dieser Arbeit wird unter einem Web Widget eine GUI-Komponente verstanden, die in einer Webseite eingebettet ist. Ziel dieser Komponente ist durch Erhöhung der Benutzerfreundlichkeit für bestimmte wiederkehrende Abläufe (innerhalb ihres angedachten Aufgabenbereiches), das Nutzererlebnis der gesamten Webseite zu verbessern. Zum Beispiel: die Integration einer Kalenderoberfläche zum Auswählen eines Datums. Die Ausführung des Web Widgets erfolgt zum Hauptteil clientseitig.

2.1.1 Web Widget Bibliotheken

Web Widget Bibliotheken stellen eine Ansammlung von vorgefertigten Widgets zur Verfügung. Die Web Widgets werden, unter Berücksichtigung der Anforderungen der Webentwicklung, in mehreren Webbrowsern getestet und auf ihre Stabilität geprüft. Das ist notwendig, da in den Webbrowsern verschiedene CSS- und JavaScript-Implementierungen zum Einsatz kommen. Dieselbe Programmstruktur wird von den Implementierungen unterschiedlich interpretiert. Aus diesem Grund können bei seriösen Bibliotheken Bugreports[4] verfasst werden, die das Entwicklerteam auf Mängel aufmerksam ma-

[3] **A**pplication **P**rogramming **I**nterface, Programmierschnittstelle
[4] Meldung eines Fehlverhaltens, das von Entwicklern behandelt werden soll

chen. Die Entwicklung einer Web Widget Bibliothek für verschiedene Webbrowser ist ein iterativer Prozess. Aufgrund des unterschiedlichen Verhaltens von Webbrowsern ist die Entwicklung zeitintensiv und setzt ein umfassendes Wissen voraus. Im Folgenden ist eine Auswahl an JavaScript-Bibliotheken angeführt, die das Erstellen von Web Widgets erleichtern.

jQuery UI

jQuery UI[5] ist ein Ableger der jQuery Bibliothek und auf die Verarbeitung von Benutzereingaben spezialisiert. Es stellt Erweiterungspakete für jQuery zur Verfügung, die aus vorgefertigten Web Widgets, Animationen, Effekten und Themes[6] bestehen. Es verwendet eine einfache und durchgängig dokumentierte API. Die Seite stellt einige Demos mit Beispielcode zur Verfügung und betreibt ein Forum mit einer aktiven Entwicklergemeinschaft. Um eigene Web Widget Kompositionen in den Ausführungsablauf der Bibliothek einzubetten, wird eine Widgetfactory[7] verwendet.

Yahoo User Interface Library

Die Yahoo User Interface Library[8] ist eine modular aufgebaute JavaScript-Bibliothek. Sie liegt zurzeit in der Version 3.10.0 vor. Wie bei jQuery UI ist es möglich, bei dem Download des Skripts, eine Auswahl an Zusatzmodulen zu treffen. Dies verkleinert die Größe des Skripts, damit die Ladezeiten für den Endanwender verkürzt werden. Die Einbettung der vorgefertigten Web Widgets ist ausführlich beschrieben und mit Beispielen versehen. Die Entwicklung übernimmt ein Kernteam von *Yahoo*. Der Code der Bibliothek ist als Open Source veröffentlicht.

Pergola

Pergola[9] versucht mittels SVG[10] (siehe Abschnitt 3.1.1) das Look-and-Feel einer Betriebssystemoberfläche zu simulieren. Dazu werden u. a. Fenster-Widgets bereitgestellt. Entwickelt wird Pergola von der Firma *dotuscomus* ohne Einfluss einer Benutzergemeinschaft. Die kommerzielle Nutzung der Bibliothek ist kostenpflichtig.

[5] http://jqueryui.com/
[6] Farbschemata
[7] Factory: Etabliertes Entwurfsmuster zur Erzeugung von Applikationsobjekten
[8] http://yuilibrary.com/yui/widgets/
[9] http://www.dotuscomus.com/pergola/examples.html#components
[10] **S**calabel **V**ector **G**raphics, ein Grafikformat

Mootools

Mootools[11] ist eine objektorientierte JavaScript-Bibliothek zur Erstellung von Web Widgets für desktopähnliche Benutzerabläufe. Es werden Klassen für die Entwicklung von Web Widgets bereitgestellt. Die verfügbaren Web Widgets sind zu einem großen Teil von externen Entwicklern verfasst worden und werden auf der Mootools Seite zum Download angeboten. Das bedeutet, dass etwaige Fehler von dem Autor des Web Widgets behoben werden und nicht von dem Mootools Entwicklerteam.

Gumby-Framework UI Kit

Das Gumby-Framework[12] stellt eine Sammlung verschiedener GUI-Elemente unter dem Namen *UI Kit* zur Verfügung. Die Elemente sind auf Responsive Web Design ausgelegt, und sollen damit auf Bildschirmen unterschiedlicher Größe ästhetisch ansprechbar sein und bedienbar bleiben. Es können verschiedene Designs für die Komponenten ausgewählt werden, die ebenfalls von dem Framework bereitgestellt werden.

Livepipe UI

Livepipe UI[13] ist eine Bibliothek, die auf der JavaScript-Bibliothek Prototype[14] aufbaut und diese um Web Widget Elemente erweitert. Livepipe UI zeigt die Möglichkeiten der Benutzerschnittstellen-Programmierung mit JavaScript. Die Bibliothek umfasst eine Vielzahl an Web Widgets, wie Texteditoren, Bildergalerien und Bewertungskomponenten. Die offizielle Entwicklung von Livepipe UI wurde eingestellt, da sie nur von einer Person betrieben wurde.

Sencha Ext JS

Sencha Ext JS[15] ist eine umfangreiche Web Widget Bibliothek, die das MVC[16]-Paradigma umsetzt. Es werden über 100 Web Widgets angeboten. Die Dokumentation erfolgt anhand eines Showcases. Es besteht die Möglichkeit, einige Web Widgets miteinander zu kombinieren. Das Geschäftsmodell sieht vor, dass Hilfestellungen zu Ext JS zu bezahlen sind, daher sind die Web Widgets nicht ausführlich dokumentiert. Die Dokumentation der Klassen-API ist in vollem Umfang auf dem Webauftritt von *Sencha* zu finden.

[11] http://mootools.net/forge/browse/category/widgets
[12] http://gumbyframework.com/docs/ui-kit/
[13] http://livepipe.net/
[14] http://prototypejs.org/
[15] http://www.sencha.com/products/extjs
[16] **M**odel **V**iew **C**ontroller, ein Muster zur Codestrukturierung

Google Chart Tools

Die Aufgabe von Google Chart Tools[17] besteht in der Visualisierung von Daten. Es werden verschiedene Diagrammformen und dazugehörige Interaktionsmöglichkeiten bereitgestellt. Außerdem können die Daten als Tabellen dargestellt werden und mit Filteroptionen und Sortierungsmechanismen bestückt werden. Die verwendete Darstellungstechnik kann von statischen Bildern, über SVG/VML[18], bis hin zu Adobe Flash[19] Inhalten variieren.

Vergleich der Bibliotheken

Die zuvor genannten Bibliotheken sind in Tabelle 2.1 aufgelistet. Die Tabelle zeigt Eigenschaften, die von den Bibliotheken erfüllt werden.

Die erste Kategorie *Standalone* veranschaulicht, die Unabhängigkeit einer Bibliothek. Eine Bibliothek ist unabhängig, wenn sie ohne Funktionen einer zusätzlichen externen Bibliothek betrieben werden kann. jQuery UI benötigt die jQuery Bibliothek wird aber als *Standalone* gewertet, da die Bibliotheken aus derselben Entwicklergemeinde stammen. Das Gumby-Framework benötigt ebenfalls jQuery und ist deshalb nicht eigenständig, da es von einer fremden Bibliothek abhängig ist.

Die Kategorie *Widgetfactory* zeigt die Verfügbarkeit von Klassen, die das Erstellen eigener Web Widgets ermöglichen. Eine Widgetfactory stellt Methoden zur Integration in das Framework bereit. Außerdem garantiert sie die Funktionstüchtigkeit des Web Widgets nach dem Instanziierungsprozess der Bibliothek. Das Kriterium wird nicht erfüllt, wenn bestehende Web Widgets nur anpassbar sind oder durch vorhandene Klassen benutzerdefinierte Web Widgets zu entwickeln sind.

Kategorie *Themes* stellt das Vorhandensein eines Farbschemas innerhalb der Bibliothek dar. Dies berücksichtigt die Zusammenstellung von Icon-Sammlungen, Hintergrundmustern, Schriftarten, usw. zu einem Gesamtpaket, um das Aussehen mehrerer Web Widgets zu vereinheitlichen.

2.1.2 Web Widget Einteilung

Es existiert eine Vielzahl an Web Widgets. Einige sind aus denselben Problematiken entstanden und versuchen diese Problematiken durch verschiedenartige Ansätze zu lösen. Die Bibliotheken stellen ähnliche Web Widgets zur Verfügung. Teilweise ist durch Kombination vorgefertigter Web Widgets, das Umsetzen vergleichbarer Lösungen anderer Bibliotheken möglich. Tabelle 2.2 kategorisiert die wesentlichen Web Widgets anhand ihrer Aufgabenbereiche.

[17] https://google-developers.appspot.com/chart/interactive/docs/index
[18] **V**ector **M**arkup **L**anguage ein Pendant zu SVG und wird von Microsoft entwickelt
[19] http://get.adobe.com/flashplayer/

Tabelle 2.1: Eigenschaften der Web Widget Bibliotheken.

Bibliothek	*Standalone*	*Widgetfactory*	*Themes*
jQuery UI	✓	✓	✓
Yahoo UI	✓	✓	✓
Pergola	✓	✗	✓
Mootools	✓	✓	✗
Gumby-Framework UI Kit	✗	✗	✓
Livepipe UI	✗	✗	✗
Sencha Ext JS	✓	✗	✓
Google Chart Tools	✓	✗	✗

Tabelle 2.2: Die Tabelle zeigt die Kategorisierung der Web Widgets anhand ihrer Aufgabenbereiche.

Struktur	*Formular*	*Visualisierung*	*Hinweise*	*Auslöser*
Accordion	Autocomplete	Charts	Dialog Panel	Button
Tabs	Datepicker Calendar	Rating	Tooltip Overlay	Skip links
Menü/ Navigation	Spinner	Responsive Images	Progressbar	
Datatable	Slider	Responsive Videos		
Dial	Colorpicker			
Scrollview	Selector			
Drawer	Combobox			

2.2 Allgemeine Eigenschaften von Tabellen

Tabellen werden verwendet für die Darstellung eines multidimensionalen Datenbestandes wie z. B. Spreadsheets oder Aktienkurse [10, Kap. 2]. Die Tabellen ermöglichen Relationen zwischen Daten abzubilden, und diese strukturiert aufzulisten.

In Webseiten eingebundene Tabellen sind passive Elemente. Sie ermöglichen Inhalte zu lesen und zu vergleichen, Interaktionsmethoden zum Bearbeiten der Daten sind nicht vorgesehen. Die fehlenden Interaktionsmethoden

haben Auswirkungen auf das Nutzererlebnis der gesamten Webseite.

Zusätzlich sind die Anforderungen an die Darstellungsformen der Tabellen für mobile Geräte mit kleinen Bildschirmen gestiegen. Benutzer sind darauf angewiesen, die Webseite durch Scrollen zu erkunden. Das Bestimmen von konkreten Daten aus einem großen Datenbestand macht es nötig horizontales und vertikales Scrollen zu kombinieren. Diese Kombination hat Auswirkungen auf die Übersicht der Tabelle, da es schwer möglich ist, Zusammenhänge von Zeilen und Spalten zu erkennen. Diese Problematik ist in [9] beschrieben.

Als Alternative wird von den heutigen Smartphones und Tablet-PCs die Skalierung einer Webseite auf die Bildschirmdimensionen des Betrachtungsgerätes angeboten. Bei der Skalierung ist eine Übersicht der Webseite gegeben, aber es können keine detaillierten Betrachtungen in Bezug auf den Datenbestand angestellt werden. Diese Thematik und deren Technologien werden in der heutigen Webentwicklung intensiv behandelt. In den folgenden Abschnitten dieses Kapitels werden bestehende Ansätze und Betrachtungen angeführt, die diese Problematik darlegen.

2.2.1 Aufbereitung von HTML-Tabellen

Die Definition der Tabellenstruktur des W3C ist seit der *HTML-Version 3.2*[20] standardisiert und dadurch lange in Gebrauch. Eine Tabelle kann in vielschichtigen Szenarios eingesetzt werden.

Durch die statische Struktur werden Tabellen häufig für das Gestalten von Webseiten verwendet, da ein Spalten-Layout definiert werden kann. Die Anforderungen an das Webdesign haben mit dem Aufkommen der mobilen Endgeräte eine Wandlung vollzogen. Layouts, die mit Tabellen umgesetzt sind, gelten als veraltet. Um ein Layout an verschiedene Bildschirmgrößen anzupassen, sind *Flexible-Grid* Layouts besser geeignet [36].

Tabellen zählen zu den komplexeren Elementen, die von dem HTML-Standard zur Verfügung gestellt werden, da sie vielfältig anpassbar sind. Diese Anpassungsmöglichkeiten sind notwendig um die Gegebenheiten der darzustellenden Daten umsetzen zu können.

Ein Nachteil besteht in den visuellen Eigenschaften. Ein Webdesigner hat viele Möglichkeiten eine HTML-Tabelle zu gestalten, allerdings tauchen in der Webentwicklung einige Spezialfälle auf, die von dem W3C nicht standardisiert sind. Deswegen interpretieren die Produkte der Webbrowserhersteller Markup oder CSS-Anweisungen unterschiedlich. Tabellen bestehen aus vielen Elementen, die voneinander abhängig sind und per Definition stark verschachtelt sind. Die Form einer Tabelle wird durch die beinhaltenden Daten gebildet.

[20] http://www.w3.org/TR/REC-html32

Programm 2.1: Beispiel eines Tabellen-Markups des Mozilla Developer Networks [25]; HTML-Kommentare wurden nachträglich hinzugefügt.

```
 1 <table>                          <!-- Tabelle -->
 2   <thead>                          <!-- Tabellenkopf (optional) -->
 3     <tr>
 4       <th>Header content 1</th>        <!-- Zellen des Tabellenkopfes -->
 5       <th>Header content 2</th>
 6     </tr>
 7   </thead>
 8   <tfoot>                          <!-- Tabellenfuss (optional) -->
 9     <tr>
10       <td>Footer content 1</td>       <!-- Zellen des Tabellenfusses -->
11       <td>Footer content 2</td>
12     </tr>
13   </tfoot>
14   <tbody>                          <!-- Tabellenkoerper -->
15     <tr>                             <!-- Zeilenanfang -->
16       <td>Body content 1</td>          <!-- Zellen des Tabellenkoerpers -->
17       <td>Body content 2</td>
18     </tr>                            <!-- Zeilen-Ende -->
19   </tbody>
20 </table>                        <!-- Tabellen-Ende -->
```

2.3 Ansätze zur visuellen und interaktiven Anpassung von Tabellen auf kleinen Bildschirmen

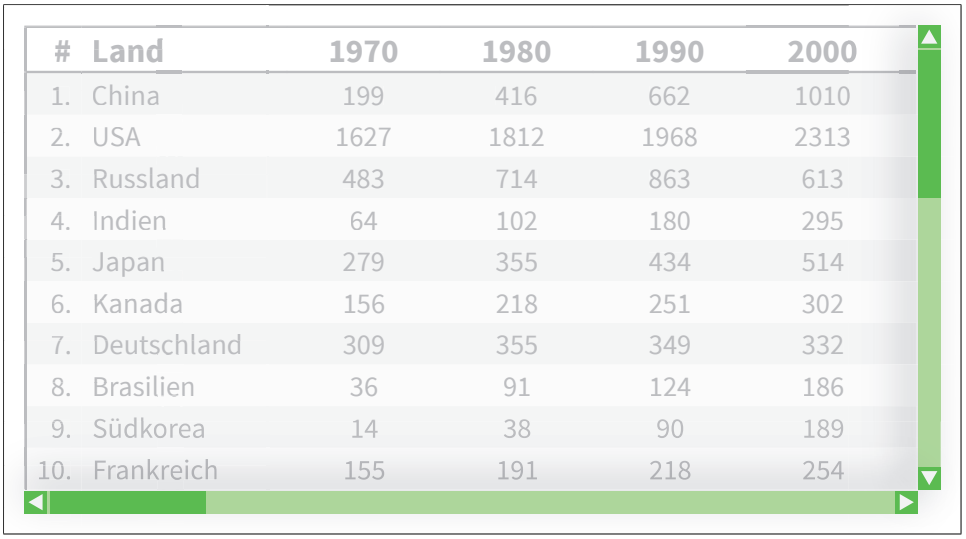
In diesem Abschnitt werden Darstellungsformen beschrieben, die Platzeinsparungen auf kleinen Bildschirmen erreichen. Die vorgestellten Ansätze behandeln Methoden, die Tabellendaten filtern oder, durch Arrangieren der Daten, alternative Präsentationsmöglichkeiten bieten.

2.3.1 Gewöhnliche HTML-Tabellen

Eine gewöhnliche HTML-Tabelle ist plattformübergreifend darstellbar, da sie ohne zusätzliche technische Hilfsmittel mit validem HTML-Markup von jedem Webbrowser, der die W3C-Recommendation implementiert, angezeigt werden kann. Alle marktrelevanten Webbrowser sind in der Lage, Tabellen-Markup zu parsen und zu interpretieren. In Programm 2.1 ist ein typischer Aufbau einer HTML-Tabelle zu sehen.

Überblicks-Tabelle

Die HTML-Tabelle wird in einem mobilen Webbrowser skaliert dargestellt. Die horizontale Dimension der Tabelle wird an den vorhandenen Betrachtungsraum angepasst, wenn die ursprüngliche Tabellenbreite jene des An-



#	Land	1970	1980	1990	2000
1.	China	199	416	662	1010
2.	USA	1627	1812	1968	2313
3.	Russland	483	714	863	613
4.	Indien	64	102	180	295
5.	Japan	279	355	434	514
6.	Kanada	156	218	251	302
7.	Deutschland	309	355	349	332
8.	Brasilien	36	91	124	186
9.	Südkorea	14	38	90	189
10.	Frankreich	155	191	218	254

Abbildung 2.1: Grafische Darstellung einer scrollbaren HTML-Tabelle. Der Ausschnitt ist im Viewport verschiebbar.

zeigemediums übertrifft. Bei der Skalierung der Tabellenbreite werden die Spaltenbreiten verkleinert. Der Inhalt der Zellen passt sich durch Textumbruch oder dem Ausblenden des Zelleninhaltes an die Gegebenheiten der Spaltenbreite an, siehe [7].

Scrollbarer Betrachtungsraum

Übertrifft die Tabellengröße, Breite und Höhe des Bildschirmes, werden von dem Webbrowser Scrollbalken hinzugefügt. Durch die hinzugefügten Scrollbalken werden die Abmessungen des zu kleinen Anzeigebereichs zusätzlich verringert. Das Scrollen verschiebt den Ausschnitt der Tabelle im Viewport. Dadurch können alle Zellen angesteuert und betrachtet werden. Abbildung 2.1 zeigt eine grafische Darstellung dieses Ansatzes.

Eine Abwandlung dieses Ansatzes ist die Entwicklung von Bushell[21]. Die erste Spalte ist von der Scrollfunktion exkludiert, da sie als Bezeichner für die Inhalte der restlichen Spalten fungiert.

2.3.2 Linearisierte Tabellen

Der multidimensionale Charakter der Datendarstellung wird bei linearisierten Tabellen aufgegeben. Die Daten werden in einer Liste, bestehend aus weiteren Unterlisten, wiedergegeben. Eine Unterliste besteht aus den Zellen einer Zeile. Der Zusammenhang des Spaltenkopfes und den Daten der Spalte

[21] http://dbushell.com/demos/tables/rt_05-01-12.html

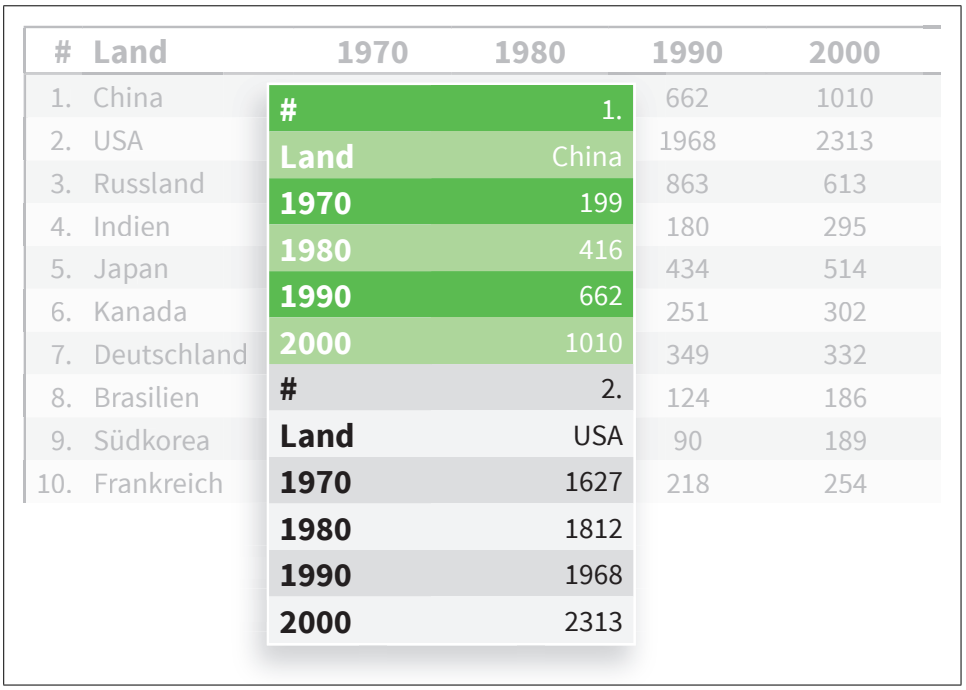


Abbildung 2.2: Darstellung einer linearisierten Tabelle. Die Tabellenköpfe werden in vertikaler Ausrichtung mehrmals eingefügt.

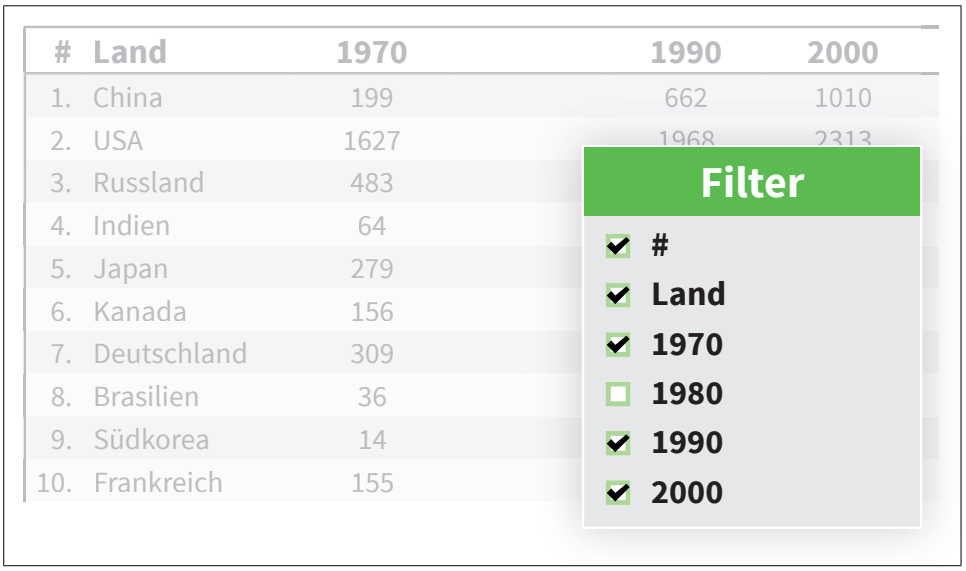
ist durch die Zeilenumbrüche der Listendarstellung nicht mehr gegeben. Um den Kontext wieder herzustellen, muss die Bezeichnung des Spaltenkopfes zu den Werten jeder Unterliste hinzugefügt werden. Dies ist in [7] beschrieben.

Bei der Umsetzung einer linearisierten Tabelle von Coyier[22] wird die Tabellenstruktur auf kleinen Bildschirmen von einem horizontalen Erscheinungsbild auf ein vertikales Erscheinungsbild umgewandelt, siehe Abb. 2.2.

Die Namen der Tabellenspalten müssen für diesen Ansatz in jedem Datensatz wiedergegeben werden, da sonst der Bezug der Daten in den Zellen zu ihren Spalten verloren geht. Vergleiche zwischen den Datenhaltungen fallen schwer, da durch die vertikale Ausrichtung die Höhe des Bildschirmes überschritten wird und zu dem nächsten Datensatz gescrollt werden muss.

Dieser Ansatz ist für das Darstellen von Personendaten geeignet, da das Vergleichen von z. B. E-Mail-Adressen für den Benutzer nicht relevant ist. Die wiederkehrende Spaltenbezeichnung kann als Visitenkarten-Ansicht genutzt werden. Die Struktur der Tabelle wird nicht mehr durch die Auszeichnungssprache bestimmt, sondern durch Präsentations-Paramter.

[22] http://css-tricks.com/examples/ResponsiveTables/responsive.php



#	Land	1970	1990	2000
1.	China	199	662	1010
2.	USA	1627	1968	2313
3.	Russland	483		
4.	Indien	64		
5.	Japan	279		
6.	Kanada	156		
7.	Deutschland	309		
8.	Brasilien	36		
9.	Südkorea	14		
10.	Frankreich	155		

Abbildung 2.3: Abbildung einer Tabelle mit Filteroption. Die Spalten werden durch Aktivieren der Checkboxen eingeschaltet.

2.3.3 Tabellen mit Filteroption

Bei dem Ansatz der *Filament group*[23] werden Spalten ausgeblendet, die keinen Platz auf dem Bildschirm haben. Eine bereitgestellte Filteroption erlaubt Spalten der Tabelle ein- oder auszuschalten wie in Abb. 2.3 dargestellt. Die Tabelle bleibt in ihrer ursprünglichen Darstellungsform erhalten und wird durch Benutzerinteraktion verschmälert oder verbreitert. Mit dieser Methode können Eigenschaften eines Datenbestandes ein- und ausgeblendet werden.

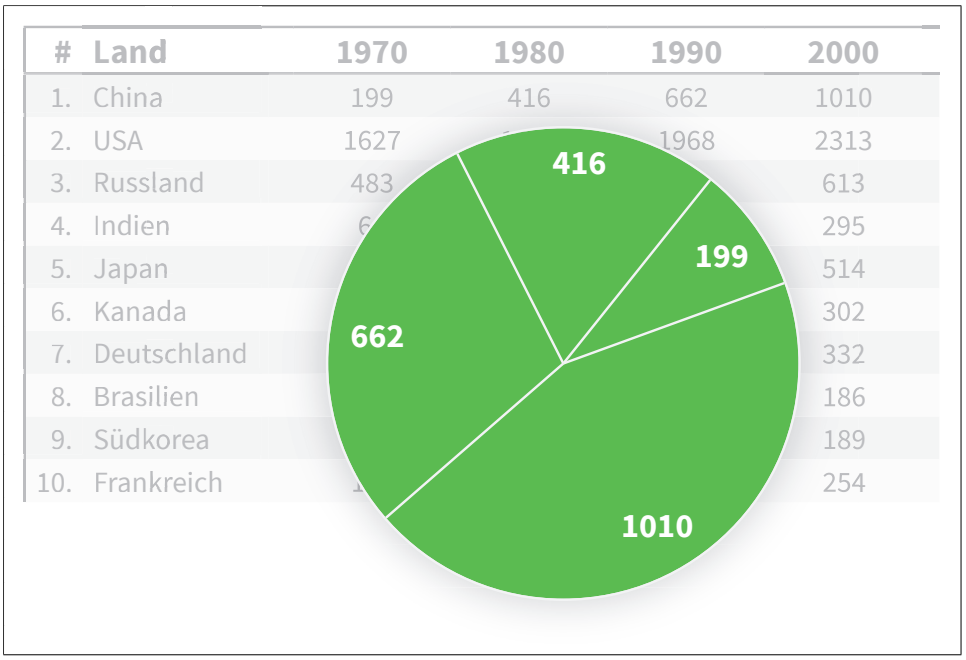
Ein Nachteil dieses Ansatzes ist, dass dem Benutzer bei zu kleinen Bildschirmen diktiert wird, welche Spalten ausgeblendet werden. Sollte der Benutzer sämtliche Spalten wieder einschalten, verhält sich die Tabelle wie eine gewöhnliche Tabelle, siehe Abschnitt 2.3.1. D. h., Webseiten können bei Anwendungsfällen, die nicht vom Webentwickler beachtet werden, Layoutprobleme bekommen.

2.3.4 Tabellen als Diagramme

Die Daten werden in einem Diagramm visuell aufbereitet[24]. Hierbei müssen die Daten in einem verarbeitbaren Format vorliegen.

Bei einer simplen Filterung durch *reguläre Ausdrücke* werden die numerischen Werte erkannt, allerdings werden die Größenverhältnisse von an-

[23] http://filamentgroup.com/examples/rwd-table-patterns/
[24] http://jsbin.com/emexa4



#	Land	1970	1980	1990	2000
1.	China	199	416	662	1010
2.	USA	1627		1968	2313
3.	Russland	483			613
4.	Indien	6			295
5.	Japan				514
6.	Kanada				302
7.	Deutschland				332
8.	Brasilien				186
9.	Südkorea				189
10.	Frankreich	1			254

Abbildung 2.4: Grafische Darstellung einer Tabelle als Diagramm.

gegebenen Einheiten verworfen und eine inkorrekte Darstellung ist die Folge.

Zum Beispiel kann *Masse* in Kilogramm oder Gramm angegeben werden. Um dies zu verdeutlichen, werden *kg* oder *g* am Ende des Wertes vermerkt. Bei der Extrahierung der nummerischen Werte werden die Einheiten ignoriert und die extrahierten Daten stimmen nicht mit den Ursprungswerten überein. Aus diesem Grund ist die Datenaufbereitung sorgfältig zu prüfen. Eine generelle Diagrammdarstellung ist nicht möglich, da nicht jede Art von Datenaufkommen in einem universalen Diagramm sinnvoll dargestellt werden kann.

Der Vorteil einer visuellen Abstraktion von Tabellen liegt in der Platzersparnis am Bildschirm, ohne Informationen zu verlieren. Abbildung 2.4 zeigt diesen Lösungsansatz.

2.3.5 Ausgeblendete Tabellen

Eine Tabelle, die die Bildschirmgröße überragt, wird nicht dargestellt und durch einen Hyperlink mit Hinweis über die Ausblendung ersetzt[25]. Der Benutzer folgt dem Hyperlink und kann die Tabelle in voller Größe ansehen.

Der Vorteil dieser Methode liegt darin, dass der Textfluss der umgebenden Elemente nicht von der Tabelle gestört wird. Außerdem wird dem Benutzer die Möglichkeit gegeben, die Tabelle gesondert anzusehen.

[25] http://jsbin.com/apane6/14

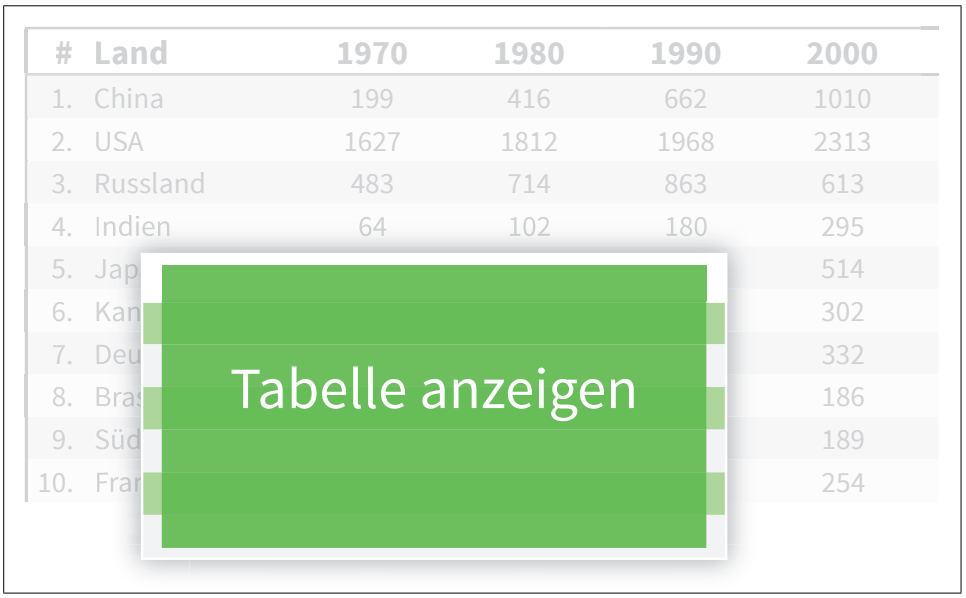


Abbildung 2.5: Beispiel einer versteckten Tabelle.

Die Größe der Tabelle kann zu denselben Hindernissen führen wie bei der gewöhnlichen Tabelle, siehe Abschnitt 2.3.1. Ein Beispiel dieser Umsetzung ist in Abb. 2.5 zu sehen.

2.3.6 Darstellungen basierend auf Tabellendaten

In [6] werden die Daten einer Tabelle auf zwei Arten interpretiert. Zum einen werden sie als mehrstufige Relationen aufgefasst und zum anderen als Matrix. Aus den zwei Dateninterpretationen werden die drei Abbildungsmöglichkeiten Normal-, Datensatz- und Zellendarstellung für Tabellen abgeleitet.

Normaldarstellung

Bei der Normaldarstellung wird die Tabelle in ihrer Ursprungsform angezeigt. Zusätzlich werden Funktionen zur Verfügung gestellt, die das Einblenden der Tabellenköpfe sowie das Verstecken von nicht gebrauchten Zeilen und Spalten ermöglichen. Das Klicken auf die Tabellenköpfe der Spalten oder die Bezeichner in den Zeilen klappt die Spalte oder die Zeile ein, um Platz zu schaffen. Als Bezeichner werden die Inhalte in der ersten Zelle einer Zeile benannt, die namensgebend für die Gesamtheit der restlichen Zellen in derselben Zeile sind.

Eine aktive Zelle zeigt den Bezeichner und den Tabellenkopf an. Dies verringert die Anzahl an Interaktionen, da die Informationen über der Zelle

angezeigt werden. Das Verharren des Mauszeigers über einer eingeklappten Zelle blendet den Inhalt der Zelle ein. Ein Mausklick auf eine eingeklappte Zelle klappt die Zelle aus und damit auch den Tabellenkopf und den Bezeichner. Außerdem kann zu der Datensatzdarstellung und der Zellendarstellung gewechselt werden.

Datensatzdarstellung

Diese Darstellung konzentriert sich auf das Anzeigen einer Zeile oder einer Spalte. Die Datenrelationen werden als Entitäten interpretiert, um Zusammenhänge erfassen zu können. Die Darstellung erfolgt in Form einer linearisierten Tabelle, siehe Abschnitt 2.3.2. Durch Doppelklick auf den Bezeichner oder einen Tabellenkopf werden alle Daten, die nicht in der korrespondierenden Zeile oder Spalte vorkommen, ausgeblendet. Die linearisierte Form stellt Typen und Werte der Zellen gegenüber.

Zellendarstellung

Die Zellendarstellung basiert auf der Matrix-Interpretation der Daten. Der Zelleninhalt wird durch den Tabellenkopf und den Bezeichner einer Zeile definiert. Ist dieser Modus aktiv, wird ein Tabellenkopf im oberen Bereich des Bildschirmes fixiert. Die aktive Spalte wird mit ihren Inhalten unter dem Tabellenkopf angezeigt, die restlichen Spalten werden ausgeblendet. Zusätzlich zu den Inhalten der Spalte werden die Bezeichner der Zeilen eingeblendet.

2.3.7 Webbrowser Plug-in

Eine Plug-in Methode für den *Microsoft Internet Explorer* zur Tabellendarstellung auf Handheld Devices wird in [9] beschrieben. Dazu wird die HTML-Tabelle als semantisch korrekt ausgezeichnete Tabelle oder als Layout-Tabelle identifiziert. Wenn die Tabelle eine Datentabelle ist, wird das Verhältnis der Daten zueinander anhand des `<th>`-Elementes kategorisiert.

Die Anordnung der Tabelle kann in die drei Kategorien Spalten-, Zeilen- oder Spalten-Zeilen-Ausrichtung eingeteilt werden. Ist kein Tabellenkopf ausgezeichnet, wird angenommen, dass die Daten spaltenweise angeordnet sind, wobei die erste Zeile der Tabelle als Tabellenkopf fungiert. Danach wird der Inhalt der Zellen nach Datentyp – Bilder, Formularelemente, Ziffern, etc. – kategorisiert. Die Typen werden in einer Matrix festgehalten, welche die identen Dimensionen hat wie die Ausgangstabelle. Das Arbeiten mit der Matrix erzeugt weniger Aufwand als vergleichbare DOM[26]-Zugriffe.

Ist die Erkennung der Datentabelle abgeschlossen, wird über das Plug-in eine Werkzeugleiste eingeblendet, die sechs Befehle bereitstellt. Die Befehle führen folgende Funktionen zum Editieren der Tabelle aus:

[26] **D**ocument **O**bject **M**odel ermöglicht programmatischen Zugriff auf HTML-Elemente

Tabelle 2.3: Vergleich der Darstellungsansätze von Tabellen

Ansatz	*Strukturveränderung*	*Interaktiv*	*Installation*
Gewöhnlich	✗	✗	✗
Linearisiert	✓	✗	✗
Filter	✗	✓	✗
Diagramm	✓	✗	✗
Ausblenden	✓	✓	✗
Tabellendaten	✓	✓	✓
Plug-in	✗	✓	✓

1. aufsteigend sortieren,
2. absteigend sortieren,
3. Ausgangsordnung der Daten wiederherstellen,
4. aktuelle Spalte ausblenden,
5. Spalte neben der aktuellen Spalte einblenden und
6. Zurücksetzen der Änderungen.

Das Plug-in ist für eine Internet Explorer Version eines Desktop-Computer entwickelt und wurde auf mobilen Geräten nicht getestet. Der Test auf dem Desktop-Computer wurde bei einer angegebenen Fensterbreite von 320 Pixel durchgeführt.

2.4 Vergleich der Ansätze

Ziel der jeweiligen Lösungsvorschläge ist, die Darstellung einer Tabelle auf kleinen Bildschirmen benutzerfreundlich zu gestalten. Trotz dieser Gemeinsamkeit sind die Ansätze der Darstellungsformen in ihren Ausführungen verschieden. In Tabelle 2.3 werden Eigenschaften der Darstellungsformen gegenübergestellt.

Die Spalte *Strukturveränderung* bezeichnet das Verhalten einer Tabelle auf kleinen Bildschirmen. Wird die Zeilen- und Spaltenanordnung der Tabelle aufgegeben, wird dies als Strukturveränderung bewertet.

Kategorie *Interaktiv* zeigt die Ansätze, die eine Benutzerinteraktion zulassen wie das Aktivieren von Filtern, das Sortieren oder das Einschalten von Zusatzfunktionen.

Die letzte Spalte *Installation* veranschaulicht die Notwendigkeit von Zusatzsoftware. Der Darstellungsansatz verwendet Zusatzsoftware, wenn die Standardfunktionen des Webbrowsers zur Darstellung des Ansatzes nicht ausreichen.

Kapitel 3

Technologische Grundlagen

Dieses Kapitel erläutert die zugrunde liegenden Techniken für das Entwickeln von Responsive Web Widgets. Es werden die Möglichkeiten der dynamischen Erzeugung von grafischen Inhalten, Bezug nehmend auf HTML5 konforme Praktiken, diskutiert. Des Weiteren werden Media Queries, die für die Entwicklung von mobilen Webseiten essenziell sind, betrachtet. Zum Steuern des Webbrowserverhaltens auf mobilen Geräten wurden über Meta-Elemente weitere Konfigurationsmöglichkeiten eingeführt, welche am Schluss dieses Kapitels behandelt werden.

3.1 SVG und Canvas

Scalable Vector Graphics und das Canvas-Element werden verwendet, um in einem Webbrowser Bilddaten anzuzeigen. Die Besonderheit an diesen Bilddaten ist, dass sie dynamisch erzeugt werden können.

Die Erzeugung von dynamischen Bildinhalten war früher auf Adobe Flash beschränkt. Mit der Einführung des Canvas-Elementes und des SVG-Standards ist es möglich, ohne zusätzliche Plug-ins Bilddaten im Webbrowser, zu verändern.

3.1.1 Scalable Vector Graphics

SVG ist ein Format, das auf dem XML-Standard[1] basiert. Zur Erstellung eines SVG-Bildes ist ein simpler Texteditor geeignet. Im SVG-Standard [33] werden einfache geometrische Objekte wie Rechtecke, Kreise und Pfade bereitgestellt. SVGs sind für die Darstellung von Illustrationen, Logos und Charts geeignet. Des Weiteren ist das Abspielen von Animationen, das Erstellen von Verläufen, die Verwendung von mathematisch beschriebenen Kurven, das Setzen von Texten, das Transformieren von Objekten und das Einbetten anderer Grafikformate möglich. Zur Laufzeit besteht die Mög-

[1] **Ex**tensible **M**arkup **L**anguage http://www.w3.org/TR/xml/

lichkeit, auf all diese Elemente des SVGs zuzugreifen. Dies erleichtert das dynamische Erzeugen von ansprechend gestalteten, interaktiven Applikationen.

Einbinden von SVGs in eine Webseite

Ein SVG kann über mehrere Wege in eine Webseite eingebunden werden. Ist eine SVG-Datei vorhanden, kann diese, wie auf Webseiten üblich, referenziert werden. Dazu ist zum einen das Einbinden der Datei mit einem `<img>`-Element oder zum anderen mit einem `<object>`-Element, durchführbar. Die Funktion der dynamischen Bearbeitung des SVGs unterscheidet die beiden Varianten.

Durch die Einbettung mit dem `<img>`-Element hat der Webentwickler keinen Zugriff auf die SVG-Elemente, der Zustand des Bildes kann nicht verändert werden. Ist das SVG über das `<object>`-Element referenziert, erhält der Webentwickler Zugriff auf den DOM und kann somit SVG-Elemente hinzufügen, löschen oder verändern. Programm 3.1 zeigt den Zugriff auf SVG-Elemente.

Zusätzlich kann ein SVG kann über die *Inline*-Methode in den HTML-Code eingebunden werden. Dabei wird der SVG-Code direkt in den HTML-Code geschrieben. Hierbei ist darauf zu achten, dass diese Möglichkeit nur mit HTML5- oder XHTML[2]-Dokumenten funktioniert. Außerdem kann es zu Problemen in der Darstellung kommen, wenn der Webbrowser aufgrund der verschiedenen Dokumenttypdefinitionen den Inhalt falsch interpretiert. Der Webbrowser zeigt dann die XML-Struktur an und stellt nicht die Webseite dar. Dieses Problem ist serverseitig mit MIME-Types[3] zu lösen. Ausführliche Beschreibungen der Techniken sind in [1, Kap. 16] zu finden.

3.1.2 Das Canvas-Element

Das Canvas-Element ist eine auflösungsabhängige Zeichenfläche, die in eine Webseite eingebettet wird, und ist Teil der HTML5-Draft Spezifikation [20]. Da der Standardisierungsprozess des W3C sehr lange dauert, haben Webbrowserhersteller zu dem WHATWG[4]-Gremium gegründet. In diesem Gremium wird unter dem Namen *HTML Living Standard* eine inoffizielle Spezifikation entwickelt, die eine gemeinsame Richtlinie für neue Technologien unter den Webbrowserherstellern darstellt.

Das Canvas-Element stellt ein Kontext-Objekt zur Verfügung, das über JavaScript ansprechbar ist. Das Kontext-Objekt ermöglicht den Zugriff auf eine API, die das Verändern des Canvas-Elementes zulässt. Das Kontext-

[2] **Ex**tensible **H**ypertext **M**arkup **L**anguage: eine Untermenge von XML zum Erstellen von Webseiten

[3] **M**ultipurpose **I**nternet **M**ail **E**xtension: Medientyp für Online-Inhalte

[4] http://wiki.whatwg.org/wiki/FAQ#What_is_the_WHATWG.3F

Programm 3.1: Verändertes Beispiel eines SVGs aus der W3C Recommendation [33]. Es wurden `id`-Attribute vergeben, um den Zugriff auf das Rechteck-Element mit JavaScript zu veranschaulichen.

```
1 ...
2 <svg id="vectorGraphic"  xmlns="http://www.w3.org/2000/svg" ...>
3   <rect id="rectangle" .../>
4 </svg>
5 ...
6 <script>
7   // Zugriff auf die Zeichenflaeche
8   var graphic = document.getElementById('vectorGraphic');
9
10  // Zugriff auf ein Element der Zeichenflaeche
11  var bar = document.getElementById('rectangle');
12 </script>
13 ...
```

Objekt kann, dem *HTML Living Standard* [35] entsprechend, entweder als `canvas.getContext('2d')` oder als `canvas.getContext('webgl')` initialisiert werden.

Der `webgl`-Kontext erlaubt das Zeichnen von dreidimensionalen Modellen, während der `2d`-Kontext für zweidimensionale Abbildungen verwendet wird. Mit dem Zugriff auf das Kontext-Objekt sind Pixel-Manipulationen auf dem Canvas-Element durchführbar.

Ähnlich wie bei SVG können Grundobjekte gezeichnet werden, Effekte erstellt werden, Bilder geladen werden und Transparenzen gesetzt werden. Außerdem sind Transformationen und das Schreiben von Texten in der API vorhanden. Allerdings werden die Buchstaben nach dem Zeichnen nicht als Text behandelt, sondern sind gerasterte Bilddaten. In Programm 3.2 wird ein exemplarischer Aufbau zur Verwendung eines Canvas-Elementes gezeigt.

Die Reihenfolge der Darstellungen der gezeichneten Objekte hängt von dem Verlauf der Erstellung ab. Das neueste Objekt wird über allen anderen gezeichnet. Da das Canvas-Element eine gerasterte Zeichenfläche ist, kann nach dem Zeichnen nicht mehr auf die gezeichneten Objekte als Einheit zugegriffen werden. Manipulationen erfolgen über die Farbwerte der Pixel.

Um eine Animation zu realisieren, müssen die Bereiche, die sich ändern, neu gezeichnet werden. Der Webentwickler ist für den Ablauf der Animation verantwortlich. Da Pixel schneller gezeichnet werden können, als Objekte im DOM erstellt werden können, eignet sich das Canvas-Element für die Umsetzung von Spielen mit JavaScript [17].

Programm 3.2: Zeichnen eines Rechtecks mit dem Canvas-Element.

```
1 ...
2 <canvas id="artboard" width="500" height="500"></canvas>
3 ...
4 <script>
5   // Zugriff auf die Zeichenflaeche
6   var context = document.getElementById('artboard').getContext('2d');
7
8   // Zeichnen eines Rechtecks auf der Zeichenflaeche
9   context.fillRect(0,0,50,50);
10 </script>
11 ...
```

3.1.3 Vergleiche zwischen dem Canvas-Element und SVG

Die Stärke von SVG liegt in der Auflösungsunabhängigkeit, das Canvas-Element kann höhere Detailtreue abbilden. Die Dateigröße eines SVGs ist in den meisten Fällen sehr klein, da es eine reine Textdatei ist, jedoch steigt die Größe mit der Komplexität der Abbildung. Die Datei kann komprimiert werden und dadurch Speicherbedarf einsparen. Auf Objekte, die in SVG gezeichnet wurden, kann später zugegriffen werden. Die Manipulationen sind DOM-Operationen. Das Canvas-Element ist Teil der HTML5-Draft Spezifikation, SVG basiert auf XML. SVG kann mit Informationen für eine barrierefreie Aufbereitung der Grafik bereichert werden, das Canvas-Element ist für komplexere Abbildungen wie Spiele geeignet. Der Inhalt des Canvas-Elementes kann über JavaScript als Bild abgespeichert werden. In [17] und [1, Kap. 16] sind weitere Aspekte angeführt.

Webbrowsersupport

In Tabelle 3.1 sind die ersten Webbrowserversionen für Desktop-Computer und mobile Geräte aufgelistet, die die Canvas-Eigenschaften [16] oder den SVG [15] Standard zu über 80% unterstützen.

3.2 Media Queries

Die Verwendung von Media Queries ermöglicht, Webseiten mit CSS-Regeln an verschiedene Displaygrößen anzupassen. Anhand von Media Queries kann unterschieden werden, welche Größen auf dem Anzeigegerät vorhanden sind. Zusätzlich kann sowohl zwischen Querformat und Hochformat differenziert werden, als auch Abfragen über die Displaygrößen oder die Ausmaße des Webbrowserfensters getroffen werden [4, S. 80].

Eine Media Query besteht aus einem *Media Type* und der eigentlichen Query. Als Media Types werden Angaben bezeichnet, die ein Stylesheet für

Tabelle 3.1: Übersicht der Webbrowserkompatibilität für das Canvas-Element [16] und SVG [15] (eingetragen sind die Versionsnummern der Webbrowser, die erstmalig über 80% der Eigenschaften implementiert haben).

Webbrowser	SVG	Canvas
Internet Explorer	9.0	9.0
Mozilla Firefox	3.0	2.0
Google Chrome	4.0	4.0
Apple Safari	3.2	3.1
Opera	9.0	9.0
iOS Safari	3.2	3.2
Android Browser	3.0	2.1
Blackberry Browser	7.0	7.0
Opera Mobile	10.0	10.0
Chrome für Android	25.0	25.0
Firefox für Android	19.0	19.0

Programm 3.3: Definition einer Media Query; die Schriftgröße des Dokumentes wird auf 100% gesetzt, wenn die Mindestbreite des Bildschirmes 1024 Pixel beträgt und die Bildschirmbreite größer ist als die Bildschirmhöhe.

```
1 @media screen and (min-width: 1024px) and (orientation: landscape) {
2   body { font-size: 100%; }
3 }
```

ein zutreffendes Zielmedium bereitstellen. Zielmedien können seit CSS2 [32] etwa Monitore, TV, Drucker, Handhelds usw. sein.

Da seit Einführung der Smartphones mehrere Variationen von Bildschirmgrößen vorhanden sind, müssen genauere Abfragen über das Zielmedium möglich sein. Dies wird seit CSS3 mit einer Media Query ermöglicht. Eine Media Query kann nicht nur den Typ des Zielmediums abfragen, sondern auch dessen Dimensionen [4, Kap. 4].

Die Media Query selbst kann aus mehreren Queries bestehen, die mit logischen Operatoren miteinander verknüpft sind. Mögliche Operatoren sind *and*, *not*, *only* und *or*, das mit einem **,** (Komma) gekennzeichnet wird. Detaillierte Beschreibungen zu den logischen Operatoren sind in [24] zu finden. Programm 3.3 zeigt eine Media Query unter der Verwendung zweier Queries, die mit dem logischen Operator *and* verknüpft sind.

3.3 Metainformationen für mobile Geräte

Als Metainformationen werden Angaben verstanden, die im `<head>`-Element eines HTML-Dokumentes definiert werden und Angaben über das HTML-Dokument enthalten, z. B. den Autor, das Erstellungsdatum, oder den Titel der Webseite. Für diesen Zweck werden die folgenden Elemente vom HTML-Standard [30] angeboten:

`<base>` gibt die Basis-URL eines Dokumentes an.

`<title>` beinhaltet den Titel des HTML-Dokumentes.

`<link>` wird verwendet, um Beziehungen zwischen Dokumenten abzubilden.

`<script>` ist für die Einbindung von Skripten in HTML-Dokumente verantwortlich.

`<style>` enthält CSS-Regeln für das Design einer Webseite.

`<meta>` Elemente werden dazu verwendet, Informationen in einem HTML-Dokument abzubilden, für die sonst kein entsprechendes Element aus dieser Liste vorgesehen ist.

Mit dem ansteigenden Verkauf von Smartphones ist der Bedarf an Webseiten für den mobilen Gebrauch gestiegen. Webentwickler sind mit dem Umstand konfrontiert, dass Webseiten auf unterschiedlichen Displaygrößen betrachtet werden, und müssen Optimierungen für kleine und große Bildschirme durchführen. Dies hat zur Folge, dass die Firma *Apple*[5] eigene Metainformationstypen für den Webbrowser *iOS Safari* eingeführt hat. Diese ermöglichen, das Verhalten des mobilen Webbrowsers bei der Anzeige von Webseiten zu ändern. Mit `name="apple-mobile-web-app-capable"` und `name="apple-mobile-web-app-status-bar-style"` sind Informationen eingeführt worden, um den Vollbildmodus des Gerätes zu aktivieren und das Verhalten der Statusanzeige zu konfigurieren. Diese beiden Informationen haben im Gegensatz zu dem `name="viewport"`-Metaelement nur auf den Betriebssystemen von Apple Bedeutung[21, 27].

3.3.1 Viewport Metatag

Für die Darstellung von Webseiten auf mobilen Geräten besteht die Möglichkeit, eine `name="viewport"`-Metainformation festzulegen, die das Darstellungsverhalten des mobilen Webbrowsers beeinflusst [27]. Der Inhalt dieser `viewport`-Metainformation wird im `content`-Attribut gesetzt, das verschiedene Angaben im Bezug auf die Darstellung der Seite beinhalten kann. Diese Angaben sind miteinander kombinierbar. Die `viewport`-Metainformation ist ebenfalls eine Entwicklung von Apple und in keinem HTML-Standard enthalten. Trotzdem haben viele Webbrowserhersteller die `viewport`-Metainformation übernommen [18, 21, 27, 29].

[5] http://www.apple.com/

Programm 3.4: Definition eines Viewports für mobile Geräte. Die gesamte Breite des Bildschirmes wird genutzt und die Webseite wird in Originalgröße angezeigt. Der Benutzer hat die Möglichkeit, durch Zoomen die Webseite vergrößert oder verkleinert darzustellen; Beispiel aus [21].

```
1 ...
2 <meta name="viewport" content="width=device-width, initial-scale=1.0,
      user-scalable=yes">
3 ...
```

Parameter des `content`-Attributes der `viewport`-Metainformation

Mithilfe der `viewport`-Metainformation können mehrere Parameter gesetzt werden. Dafür werden dem `content`-Attribut die Parameter als Eigenschaft-Wert-Paare übergeben. Die folgende Liste an Eigenschaften ist aus [29] zusammengefasst.

`width` ist der Wert des Viewports, der die Breite angibt. Der Wert kann in Pixel angegeben werden oder mit `device-width` als Konstante gesetzt werden. `device-width` gibt die Bildschirmgröße eines Gerätes an.

`height` wird standardmäßig aus der `width` Angabe und dem Seitenverhältnis des Bildschirmes berechnet, kann aber ebenfalls als Pixelwert oder mit `device-height` als Konstante gesetzt werden.

`minimum-scale` ist der kleinste Zoomfaktor. Ein nummerischer Wert bestimmt die kleinstmögliche Anzeige der Webseite.

`maximum-scale` ist das Gegenstück zu `minimum-scale`. Die Originalabmessungen der Elemente werden um diesen Skalierungsfaktor größer dargestellt.

`initial-scale` gibt einen Skalierungsfaktor an. Ist der Skalierungsfaktor auf `1.0` gesetzt, werden die Elemente in Originalgröße dargestellt. `initial-scale` ist ein nummerischer Wert und bezieht sich auf die Skalierung der Webseite, nachdem diese vollständig geladen wurde. Danach kann der Benutzer im Bereich zwischen `maximum-scale` und `minimum-scale` die Webseite vergrößern oder verkleinern.

`user-scalable` kann dazu verwendet werden, dem Benutzer das Zoomen auf der Webseite zu erlauben oder zu verbieten. Die Eigenschaft kann mit `yes` oder `no` gesetzt werden.

Programm 3.4 veranschaulicht das Benutzen einer `viewport`-Metainformation unter Verwendung der eben genannten Eigenschaften.

Kapitel 4

Entwurf

Der Entwurf beschreibt Gedanken zur Entwicklung von Responsive Web Widgets. Anforderungen an Responsive Web Widgets werden vorgestellt. Differenzen zwischen Touch-Gesten und Mauseingaben werden aufgezeigt und ihre entsprechenden Anwendungen beschrieben, um einen reibungslosen Ablauf auf verschiedenen Geräten gewährleisten zu können. Der Schwerpunkt liegt bei der Konzeption von Responsive Web Widgets, die auf die Darstellung und Manipulation von tabellarisch aufbereiteten Daten spezialisiert sind.

4.1 Allgemeine Voraussetzungen

In den letzten Jahren erfolgten bedeutende Änderungen in den Betrachtungsgewohnheiten von Webseiten. Neue Geräte unterschiedlichen Typs sind mit neuer Webbrowser-Software auf den Markt gerückt. Um diese Geräte mit Webinhalten bedienen zu können, müssen Webseiten auf die unterschiedlichen Hardware- und Softwaregegebenheiten mit diversen Anpassungen reagieren.

Der Ausdruck Responsive Web ist der Inbegriff für dieses Unterfangen. Da nicht abgeschätzt werden kann, mit welchen Geräten eine Webseite betrachtet wird, muss eine Webseite möglichst flexibel aufgebaut sein. Zurzeit sind Smartphones und Tablet-PCs beliebte Zugangsgeräte zum Betrachten von Online-Inhalten, dies wird durch eine aktuelle Statistik – siehe Abbildung 4.1 – bestätigt. Responsive Web Design bedeutet, die Gegebenheiten des Betrachtungsgeräts, so gut als möglich auszunutzen.

Zum Beispiel können Webseiten mit Screenreadern betrachtet werden, diese setzen voraus, dass eine Webseite semantisch korrekt aufbereitet ist. Damit ist gegeben, dass der Screenreader Zusammenhänge von Textpassagen erkennt und Reihenfolgen beachten kann.

Es ist nicht möglich innerhalb eines Webprojektes für jede Bildschirmauflösung und jedes Zielgerät eigens angelegte Webseiten anzubieten, da

Entwicklungs- und Wartungsaufwand zu hoch sind und die Wiederverwendbarkeit darunter leidet. Native Applikationen müssen für jedes, der im Umlauf befindlichen Betriebssysteme, gesondert entwickelt werden.Jedes Betriebssystem setzt eine eigene Programmiersprache voraus: *iOS* verwendet Objective-C, *Android*[1] benutzt Java und *Windows Phone*[2] benötigt C#. Native Applikationen bieten einen besseren Bedienungsablauf als Web Widgets, da sie direkten Zugriff auf die Hardware des Gerätes haben. Außerdem können sie auf Ressourcen wie Kontakte oder SMS auf dem Gerät zugreifen und diese für die Anwendung nutzen.

Webapplikationen können aus Sicherheitsgründen keine Systemdaten erreichen, dienen aber als kleinster gemeinsamer Nenner, der alle Technologien geräteübergreifend miteinander verbindet. Der Responsive Web Design Ansatz versucht unter Verwendung von bestehenden Webstandards und Methoden wie dem flexiblen Layout, siehe Abschnitt 4.2.4, die Spanne an Geräten abzudecken.

4.2 Benutzeroberfläche

Bei der Gestaltung der Benutzeroberfläche für Smartphones oder Tablet-PCs ist darauf zu achten, dass die Größen der Interaktionsflächen ausreichend bemessen sind. Touch-Gesten sind im Gegensatz zu den pixelexakten Interaktionen eines Mauszeigers sehr ungenau. Eine Touch-Geste ist nicht auf einen Punkt oder Pixel beschränkt, sondern deckt eine Fläche ab. Komponenten der Benutzeroberfläche müssen einer Mindestgröße entsprechen, damit die Benutzerfreundlichkeit einer Webseite auf mobilen Geräten nicht eingeschränkt ist.

Hersteller von Betriebssystemen für mobile Geräte veröffentlichen Richtlinien als Hilfestellung für Entwickler, damit das Nutzererlebnis auf verschiedenen Geräten konsistent bleibt. Tabelle 4.1 zeigt die kleinsten angedachten Abmessungen eines Buttons auf einem Touchscreen, sodass die Benutzerfreundlichkeit erhalten bleibt.

Microsoft gibt für *Windows Phone 8* die Abmessungen in Millimeter für 262 *dpi*[3] gefertigte Displays an. Bei Googles *Android* werden die Dimensionen in *dp*[4] angeführt. Diese relative Maßeinheit soll das Arbeiten mit verschiedenen Displayfertigungen und Auflösungsunterschieden erleichtern. Je nach dpi des Bildschirmes können dabei die Abmessungen eines Buttons variieren, wenn absolute Maßeinheiten verwendet werden.

Die absoluten Maßeinheiten in CSS wie `mm` oder `in` (Zoll) werden z. B. im Webbrowser Firefox über die Pixelabmessungen berechnet [23]. Die Angabe

[1] http://www.android.com/

[2] http://www.windowsphone.com/de-at

[3] **d**ots **p**er **i**nch: Auflösungmaßeinheit bezogen auf einen Zoll

[4] **d**ensity-independent **p**ixels

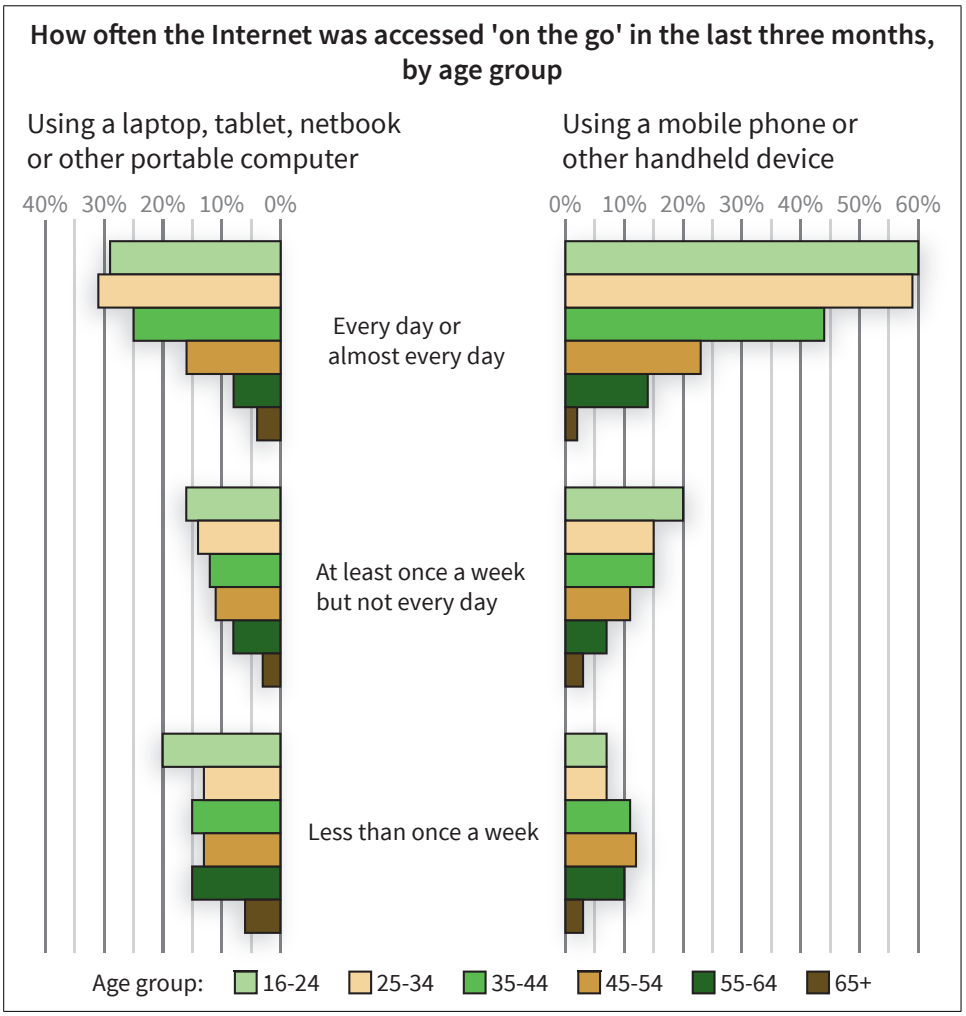


Abbildung 4.1: Die Häufigkeit der mobilen Internetnutzung nach Altersgruppen sortiert. Zu erkennen ist, dass mehr als die Hälfte der Bevölkerungsgruppen unter 35 Jahren, jeden Tag ein Smartphone oder etwas Ähnliches für den Internetzugriff nutzt. Die Datenerhebung stammt aus dem Vereinigten Königreich vom *Office for National Statistics* [28].

`1in` hat die Länge von `96px`. Da die Pixelgröße von der Displayfertigung abhängig ist, variiert auch die Abmessung von `in`, da Firefox einen Zoll aus der Definition von 96 Pixeln ableitet. Aus diesem Grund sollten im Webbereich, für ein durchgängiges Benutzerschnittstellen-Design, relative Maßeinheiten wie `em`[5] verwendet werden.

[5]Schriftgrößenabhängige Maßeinheit

Tabelle 4.1: Minimale Dimensionen eines Buttons für Touch-Interaktion aus den jeweiligen Design Richtlinien von Android [12], iOS [40] und Windows [38].

Betriebssystem	*Auflösung*	*Minimale Buttonabmessungen*
Android	—	48 dp × 48 dp ≈ 7–10 mm × 7–10 mm
iOS iPhone	low < 160 dpi high > 160 dpi	43 px × 43 px 86 px ×86 px
iOS iPad	low < 160 dpi high > 160 dpi	55 px × 55 px 110 px × 110 px
Windows Phone 8	262 dpi	90 px × 90 px ≈ 7–9 mm × 7–9 mm

4.2.1 Touch-Gesten

Geräte, die über einen Touchscreen verfügen, werden mittels Berührungen und Bewegungen direkt auf dem Bildschirm von einem oder mehreren Fingern gesteuert. Diese Bewegungen müssen einen bestimmten Ablauf erfüllen, damit sie vom Betriebssystem als Aufforderung zum Ausführen von Aktionen, interpretiert werden. Bezeichnet werden diese Bewegungen als Gesten. Dabei sind zwischen Single-Touch-Gesten und Multi-Touch-Gesten zu unterscheiden. Bei Multi-Touch-Gesten sind mehrere Finger in den Bewegungsablauf involviert.

Das zeitgleiche Erkennen von mehreren Berührungen setzt einen Touchscreen voraus, der diese Technik unterstützt. Da einige berührungssensitive Geräte im Umlauf sind, die keine Multi-Touch-Gesten unterstützen, sind Single-Touch-Gesten, für die Entwicklung von geräteübergreifenden Applikationen, Multi-Touch-Gesten vorzuziehen. Etablierte Gesten sind in Abbildung 4.2 schematisch dargestellt. Die in [8, Kap. 5] und [11] gesammelten Gesten werden im Folgenden beschrieben:

Tap/Touch ist eine kurze Berührung des Bildschirmes mit einem Finger; die häufigste Geste bei der Verwendung von Touchscreens.

Double touch wird eine Bewegung genannt, bei der zwei schnell aufeinanderfolgende Tap Gesten ausgeführt werden.

Long press ist ähnlich der Tap Geste. Die Dauer der Bildschirmberührung ist länger. Der Finger soll in dieser Zeit nicht beweget werden.

Swipe/Flick erfolgt durch Drücken auf den Bildschirm und einer schnellen Bewegung des Fingers. Der Kontakt darf während der Bewegungsphase nicht unterbrochen werden.

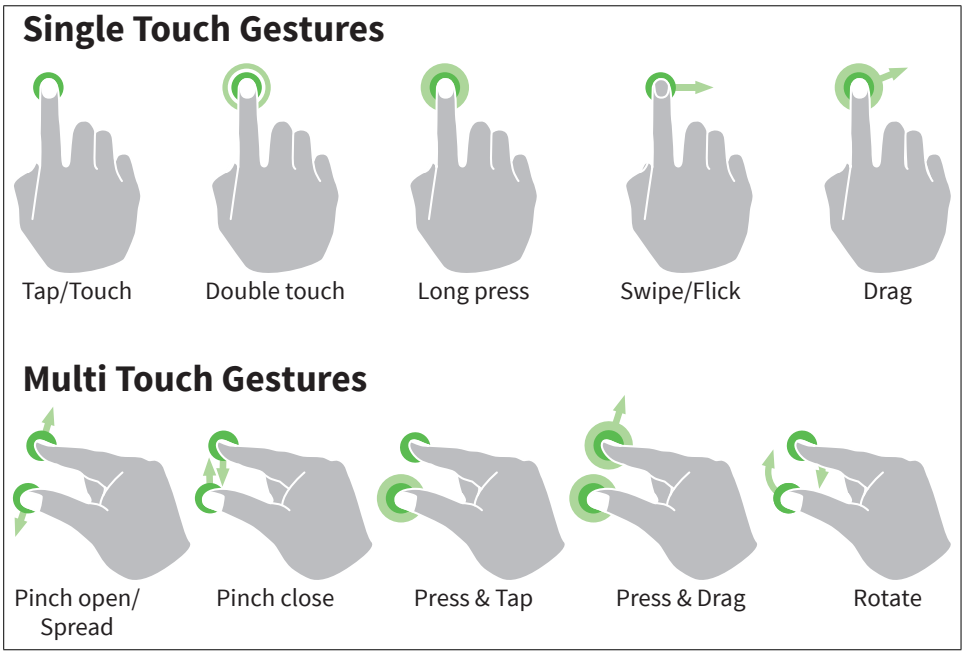


Abbildung 4.2: Etablierte Gesten für verschiedene Geräte mit Touchscreen. Die obere Reihe zeigt Gesten, die mit einem Finger durchgeführt werden, die untere Reihe bildet Multi-Touch-Gesten ab.

Drag wird durch eine Long press Geste begonnen. Anschließend wird der Finger über den Bildschirm geführt. Zwischen der Long press Geste und der Bewegung wird der Finger nicht abgesetzt.

Pinch open/Spread ist eine Multi-Touch-Geste. Zwei Finger berühren zeitgleich den Bildschirm. Das Gerät erkennt zwei Berührungsflächen. Die Fingerspitzen werden, ohne den Bildschirm zu verlassen, auseinandergezogen.

Pinch close ist die entgegengesetzte Geste zu Pinch open. Die Fingerspitzen der beiden Finger werden zueinander hinbewegt.

Press and Tap wird die Kombination aus einer Long press Geste und einer Tap Geste genannt. Sie ist eine Multi-Touch-Geste. Als Erstes wird eine Long press Geste, wie oben beschrieben, ausgeführt. Anschließend wird mit einem zweiten Finger eine Tap Geste getätigt, während der erste Finger noch auf dem Bildschirm verweilt.

Press and Drag ist die Kombination der zwei Gesten Long press und Drag und ebenfalls eine Multi-Touch-Geste. Der Finger, der die Long press Geste ausführt, bleibt in Kontakt mit dem Bildschirm. In weiterer Folge berührt ein zweiter Finger den Bildschirm und wird im ständigen Kontakt über diesen bewegt. Der erste Finger verweilt währenddessen in Ruheposition auf dem Bildschirm.

Rotate zwei Finger berühren zeitgleich den Bildschirm. Beide Finger werden durch eine Drehbewegung der Hand entlang des Bildschirmes rotiert. Während der Drehbewegung müssen beide Finger den Kontakt zum Bildschirm aufrechterhalten. Die Rotate Geste zählt zu den Multi-Touch-Gesten.

4.2.2 Mouse-Events und Touch-Events

Eine Herausforderung für das Entwickeln von geräteübergreifenden Applikationen ist das Abbilden der spezifischen Interaktionsmethoden auf die Funktionsbefehle der Applikation.

Smartphones und Tablet-PCs haben als Eingabemöglichkeiten Touch-Gesten. Die Geräte werden anhand der Unterstützung von Multi-Touch-Gesten oder Single-Touch-Gesten unterschieden. In seltenen Fällen haben Desktop-Computer und Laptops Monitore, die Touch-Gesten verarbeiten können.

Die vorherrschende Eingabemöglichkeit auf diesen Geräten ist eine Kombination aus Tastatur und Zeigergerät. Als Zeigergeräte werden Mäuse, Trackballs, Touchpads und Grafiktabletts verstanden.

Applikationen müssen so aufgebaut sein, dass die Möglichkeit besteht, gerätespezifische Interaktionsmechanismen zu nutzen. Ausgehend von zwei simplen Eingabeaktionen, dem Mausklick auf einem Desktop-Computer und einer Tap Geste auf einem Touchscreen, werden die unterschiedlichen Phasen beim Ausführen einer Aktion beschrieben.

Mobile Versionen von Webbrowsern interpretieren einen Mausklick als Tap Geste. Dies ist eine etablierte und gute Lösung um Befehle geräteübergreifend auszulösen. Doch bei genauer Betrachtung sind hier erhebliche Unterschiede erkennbar. Als einfaches Szenario soll ein Button auf einer Webseite gedrückt werden. Der Ablauf mit Verwendung einer Maus ist folgender:

- Bewegung der Maus in Richtung des Buttons.
- Der Mauszeiger ist über dem Button. Der Button kann dies durch ein grafisches Feedback kundtun.
- Drücken der linken Maustaste startet den Mausklick. Der Button registriert, dass er gedrückt wird.
- Solange der Mauszeiger nicht bewegt wird, ist die Dauer des gedrückten Zustandes der Taste unerheblich. Der Button bleibt gedrückt.
- Loslassen der Maustaste beendet den Mausklick. Der Button bemerkt, dass er nicht mehr aktiv ist.
- Der Mauszeiger wird an einen anderen Ort bewegt. Der Button registriert, dass der Mauszeiger nicht mehr über ihm ist.

Der Ablauf mit Tap Geste:

- Der Finger wird in Richtung des Buttons bewegt.

Tabelle 4.2: Gegenüberstellung von Mouse-Events und deren Touch-Event-Äquivalenten. Ein neuer W3C-Standard wird Pointer-Events einführen [31].

Mouse-Event	*Touch-Event*	*Pointer-Event*
mousedown	touchstart	pointerdown
mouseup	touchend	pointerup
mouseenter	touchenter	pointerenter
mouseleave	touchleave	pointerleave
mousemove	touchmove	pointermove
click	**Nicht vorhanden**	**Nicht kompatibel**
mouseover	**Nicht vorhanden**	pointerover
mouseout	**Nicht vorhanden**	pointerout

- Der Finger ist knapp über dem Button. Die Distanz zwischen Finger und Bildschirm kann nicht berücksichtigt werden. Der Button kann kein Feedback anzeigen, solange der Finger nicht den Bildschirm berührt.
- Mit dem Berühren des Buttons auf dem Bildschirm startet die Tap Geste. Der Button wird gedrückt.
- Bleibt der Finger längere Zeit auf dem Bildschirm, ohne zwischenzeitliche Bewegung, wird ein Long press ausgeführt und nicht die Aktion, die die Tap Geste auslösen sollte.
- Der Kontakt zwischen Finger und Bildschirm wird unterbrochen. Die Tap Geste wird beendet. Der Button registriert, dass er losgelassen wird.

Der Mausklick ist einer der meistverwendeten Eingabemechanismen, kann aber, wie in Tabelle 4.2 ersichtlich, keinem Touch-Event gegenübergestellt werden. Ein Mausklick kann aus den zwei aufeinanderfolgenden Mouse-Events mousedown und mouseup abgebildet werden. Touch-Events können nicht standardkonform in Webbrowsern behandelt werden, darum haben Webbrowserhersteller Touch-Events auf mobilen Geräten als Mouse-Events zurückgegeben [39].

Eine Tap Geste wird von den mobilen Versionen der Webbrowser als *mousedown* und *mouseup* interpretiert, wenn diese in einem kurzen Zeitintervall innerhalb eines kleinen Bewegungsradius vorkommen. Das mobile Internet wird immer mehr genutzt und deswegen wurden zusätzlich zu den Mouse-Events Touch-Events eingeführt, die jedoch in keinem Standard vorgesehen sind. Zum jetzigen Zeitpunkt werden sie nur von Google Chrome und Mozilla Firefox unterstützt, siehe [26, 37]. Außerdem werden bei diesem

Ansatz auf mobilen Geräten nicht nur die Touch-Events ausgelöst, sondern – aus Kompatibilitätsgründen – zusätzlich die Mouse-Events [22].

Das Auslösen von den verschiedenen Eventtypen kann zu Problemen führen, wenn eine Webseite nur für den Desktop-Computer entwickelt wurde. Aktuell ist vom W3C eine Candidate Recommendation für Pointer-Events in Arbeit [31]. Mouse- und Touch-Events werden darin gemeinsam mit den Events anderer Eingabemethoden als Pointer-Events plattformunabhängig ansprechbar sein. Das Entwickeln von Webapplikationen unabhängig von deren Eingabemethoden soll damit erleichtert werden, allerdings ist auch hier der Mausklick von der Kompatibilität ausgenommen, da er z. B. auch durch das Drücken der Eingabetaste ausgelöst werden kann.

Nominell sind *mousemove*-Events und *touchmove*-Events gleichbedeutend, in technischer Hinsicht verhalten sie sich jedoch verschiedenartig. Während der Ausführung eines *touchmove*-Events wird immer das Element zurückgegeben, in dem das *touchmove*-Event begonnen hat. Bei einem *mousemove*-Event wird jenes Element zurück gegeben auf das der Mauszeiger gerade zeigt. Dieser Umstand muss beachtet werden, wenn auf Desktop-Computern und Geräten mit Touchscreens dieselben Aktionen auf einem Element ausgeführt werden sollen [22].

4.2.3 Aktionen

Nach Durchführung der jeweiligen Touch-Gesten erwarten Benutzer das Ausführen von etablierten Aktionen. Ein Element soll selektiert werden, wenn eine Tap Geste darauf angewendet wird. Auf eine Long press Geste erfolgt üblicherweise ein Wechsel des Applikationsmodus, z. B. von einem Betrachtungsmodus zu einem Editiermodus.

Swipe und Drag können zum Scrollen verwendet werden, wobei die Swipe Geste schnelleres Scrollen auslöst als eine Drag Geste. Ein Element, das selektiert wird und durch eine Drag Geste aus dem Bildschirm oder auf eine dafür vorgesehene Fläche gezogen wird, kann als gelöscht angesehen werden. Diese und weitere Methoden sind von Wroblewski in [8, Kap. 5] beschrieben.

4.2.4 Flexibles Layout

Die Benutzerschnittstellen-Programmierung für verschiedene Geräte muss unterschiedliche Bildschirmgrößen beachten. Aus diesem Grund wird eine geräteübergreifende Applikation mit einer flexiblen Layoutstruktur ausgestattet [8, Kap. 7].

Dabei werden, in Anwendung des Responsive Web Designs, die Dimensionen von HTML-Elementen ohne absolute Längenangaben ausgezeichnet. Somit ist gegeben, dass durch die Verwendung von relativen Einheiten wie Prozent- oder *em*-Angaben, HTML-Elemente ihre Breite dynamisch an die Platzverhältnisse des Bildschirmes anpassen.

Die Beeinflussung des Textflusses durch die dynamischen Größenverhältnisse der Elemente ist ebenfalls zu beachten. Texte, die durch das Responsive Web Widget aufbereitet werden, weisen unterschiedliche Längen und Buchstabenanzahlen auf. Je nach Einsatzgebiet muss das Responsive Web Widget die Texte lesbar und übersichtlich darstellen.

Durch CSS-Regeln, die die *float*-Eigenschaft betreffen, können HTML-Elemente so gesteuert werden, dass Inhalte – je nach Bildschirm – an verschiedenen Positionen platzierbar sind. Des Weiteren werden mit dem Einsatz von scrollbaren Bereichen überlange Inhalte zugänglich.

4.2.5 Datenvisualisierung

Daten können textuell als Tabellen oder grafisch als Diagramme dargestellt werden. Das Verwenden von Diagrammen macht das schnelle Ausmachen von Datenzusammenhängen möglich. Bei aufwendigen Tabellen kann dies mehr Zeit in Anspruch nehmen. Nicht jedes Diagramm ist die passende Lösung für einen Datenbestand, da Fehlinterpretationen möglich sind, wenn eine ungünstige Diagrammart gewählt wird [19].

Diagramme sind als Alternative für die Datenaufbereitung geeignet, da ihre Verwendung Platzersparnis bewirken kann. Grafische Elemente sind auf kleinen Bildschirmen einfacher zu verkleinern als Texte in Tabellenzellen. Signifikante Wertunterschiede können mithilfe von logarithmischen oder quadratischen Auszeichnungen der Skalen visuell abgeglichen werden. Das Arrangement der Daten variiert je nach Diagramm, gleichzeitig können die grafischen Repräsentationen der Daten als Interaktionsflächen dienen.

Heer beschreibt in [3], dass bei der Untersuchung oder Manipulation von Daten der Benutzer die Möglichkeit haben muss, Elemente zu selektieren. Selektionen können einen Betrachtungsraum einschränken oder die Vorstufe für das Filtern oder das Bearbeiten von Elementen sein. Außerdem soll als Erstes ein Überblick über die Datenlage gegeben werden. Anschließend schreibt Heer, dass mit Zoom- und Filterfunktionen die Möglichkeit bereitgestellt werden soll, ein Fokus auf bestimmte Interessensgebiete zu setzen.

Des Weiteren sollen die Interessensgebiete auf Wunsch des Benutzers Detailinformationen einblenden. Dabei soll auf den Kontext geachtet werden, indem die Daten dargestellt werden. Heer warnt vor der Verwendung von *mouseover*-Events für das Einblenden von Details, da diese nicht für Touchscreens geeignet sind.

4.2.6 Benutzerverhalten bei der Betrachtung von Tabellen

Auf dem Monitor eines Desktop-Computers ist genügend Raum für die Darstellung von umfangreichen Webseiten vorhanden. Ist der Bildschirm zu klein für die umfassende Darstellung, können durch Maus und Tastatur schnelle Interaktionen zum Betrachten der ganzen Webseite getätigt wer-

den.

Mobile Geräte werden immer häufiger für das Surfen im Internet verwendet, haben aber eingeschränkte Steuerungsmöglichkeiten. Es ist ein Repertoire an Fingergesten-Erkennung vorhanden, doch können die meisten Aktionen nur mit einem Finger ausgelöst werden. Beim Arbeiten mit Tastatur und Maus können am Desktop-Computer beide Hände gleichzeitig Aktionen ausführen. Eine Hand kann durch eine Applikation navigieren, während die andere Hand Befehle der Applikation auslöst.

Bei dem Betrachten einer Tabelle wird in [9] von Xu folgendes Benutzerverhalten angeführt:

1. Betrachtung der ganzen Tabelle,
2. Betrachtung von bestimmten Zeilen oder Spalten,
3. Herausfinden wie Zeilen und Spalten zusammenhängen und
4. Vergleiche – basierend auf den Zelleninhalten – ziehen.

Daraus resultiert, dass auf kleinen Bildschirmen viele Interaktionen nötig sind, um einen detaillierten Blick auf den Datenbestand zu erlangen. Mit jedem Vergleich von zwei weit auseinanderliegenden Werten kommt es zu einer Wiederholung der Interaktionen.

Tajima beobachtet in [6, S. 260] das Benutzerverhalten etwas anders:

1. Betrachtung einer Zelle als Schnittpunkt von Zeilen und Spalten,
2. Betrachtung von Zellen innerhalb einer Zeile oder einer Spalte,
3. die Tabelle zeilenweise oder spaltenweise lesen,
4. Vergleiche zwischen den Datensätzen von bestimmten Zeilen oder Spalten ziehen und
5. Suchen eines bestimmten Wertes in der Tabelle.

Tajima kritisiert, dass auf kleinen Bildschirmen das Navigieren durch große Tabellen einen Nachteil für den Benutzer bringt. Die Kopfzeile der Tabelle kann durch die eingeschränkte Bildschirmgröße aus dem Sichtbereich verschwinden. Dadurch ist der Kontext der betrachteten Zelle nicht mehr ersichtlich. Bei Tabellen, die nur nummerische Werte beinhalten, kann dies zur Folge haben, dass oft vor und zurück navigiert werden muss.

In den Arbeiten von Xu und Tajima kommen beide zu dem Schluss, dass bei der Betrachtung von Tabellen auf kleinen Bildschirmen der übermäßige Navigationsaufwand das Hauptproblem ist.

4.3 Anforderungen an Responsive Web Widgets

Darstellung einer Tabelle auf einem Smartphone. Die linke Seite zeigt die Tabelle in Originalgröße, deswegen ist nur ein Ausschnitt sichtbar. Die zweite Version zeigt die Tabelle in skalierter Form, die Lesbarkeit nimmt ab.

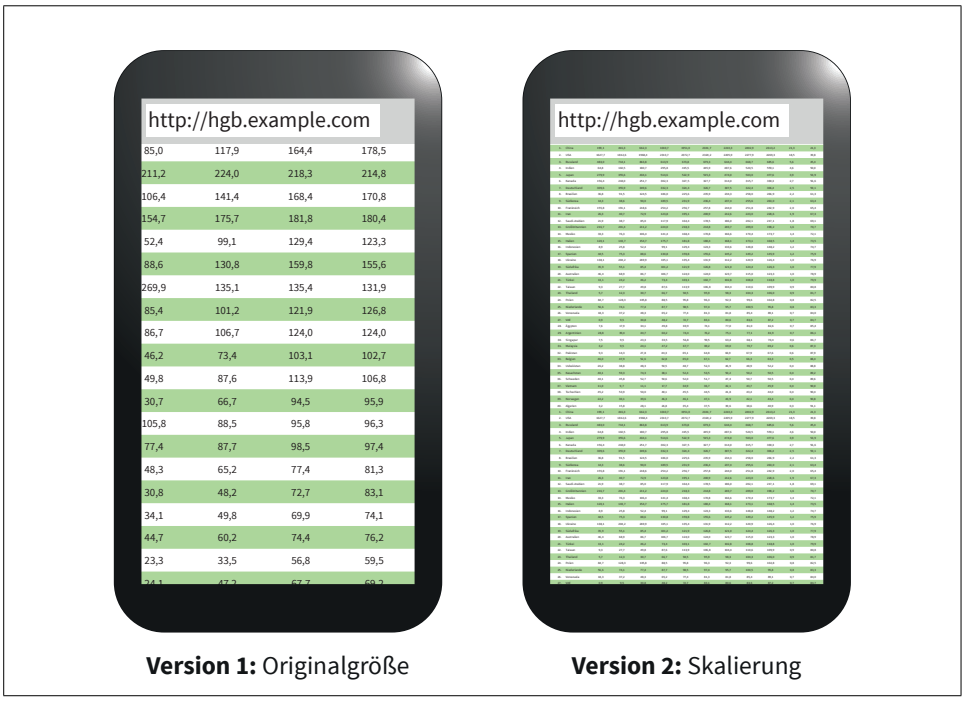


Abbildung 4.3: Darstellung einer Tabelle auf einem Smartphone. Die linke Seite zeigt die Tabelle in Originalgröße, deswegen ist nur ein Ausschnitt sichtbar. Die zweite Version zeigt die Tabelle in skalierter Form, die Lesbarkeit nimmt ab.

Der Darstellung von HTML-Tabellen auf kleinen Bildschirmen wird wenig Aufmerksamkeit geschenkt. In Abbildung 4.3 ist zu erkennen, dass das Standardverhalten der Tabellenanspassung auf einem Smartphone mit Android-Betriebssystem unzureichend ist.

In beiden Versionen ist auszumachen, dass die Zusammenhänge der tabellarisch aufbereiteten Daten schwer zu erschließen sind. Auf der linken Seite der Abbildung wird die Tabelle in Originalgröße angezeigt. Es wird aufgrund des Platzmangels nur ein Ausschnitt des Datenaufkommens dargestellt. In diesem Fall muss horizontal und vertikal gescrollt werden, um eine Übersicht zu erhalten.

Da Benutzer bereits gewohnt sind bei der Betrachtung von Webseiten, die für mobile Geräte optimiert sind, vertikal zu scrollen, sollte sich das Scrollen auf diese eine Richtung beschränken. In der zweiten Version werden die Dimensionen der Tabelle auf die Bildschirmgröße angepasst. Das Lesen der Zelleninhalte ist durch die geringe Größe der Schrift nicht möglich. Der Benutzer kann auf diese Weise die Anzahl der Spalten erahnen, jedoch können keine Schlüsse gezogen werden, wie die dargestellten Daten zueinander in Verbindung stehen. Mit der Anzahl an Spalten nimmt die Übersicht der

skalierten HTML-Tabelle ab, da weniger Platz für jede Spalte zur Verfügung steht.

Passives Verhalten: HTML-Tabellen werden vorwiegend zum Betrachten von Daten eingesetzt. Sie sind somit Elemente in einer Webseite, die keine Interaktionen zulassen.

Sind Änderungen der Daten erforderlich, muss durch eine serverseitige Lösung der Datenbestand aktualisiert werden. Dies geschieht anhand von CMS[6]-Applikationen, Datenbankeingriffen oder dem Hochladen aktuellerer HTML-Dateien. Diese Zwischenschritte können eingespart werden, wenn das direkte Ändern der Daten in einer HTML-Tabelle mit anschließend automatisierten Aktualisierungen an den Server ermöglicht wird.

4.4 Konzeption des Responsive Web Widgets

Da, wie in Abschnitt 4.2.5 erwähnt, nicht jedes Diagramm für jede Datenlage geeignet ist, werden in dieser Arbeit zwei Diagramme mit unterschiedlichen Eigenschaften ausgewählt. Zum einen wird das Kreisdiagramm vorgestellt, das einen guten Überblick für Verteilungen und Anteile bietet und zum anderen das Balkendiagramm, das Gruppierungen und Direktvergleiche erlaubt.

4.4.1 Kreisdiagramm Widget

Das Responsive Web Widget verwendet als Darstellung der Tabellendaten ein Kreisdiagramm, wenn die Bildschirmgröße nicht für die Anzeige der Tabelle ausreicht.

Dieses Kreisdiagramm stellt die Eigenschaften einer Entität dar. Als Entität wird die Zeile einer Tabelle gedeutet.

Die erste Spalte beinhaltet den Namen der Entität. Mit Ausnahme der ersten Spalte der Tabelle werden die restlichen Spalten als Eigenschaften einer Entität gewertet. Der Tabellenkopf bezeichnet in seiner ersten Zelle den Typ der Entität. Alle weiteren Kopfzellen sind als Bezeichner namensgebend für die Eigenschaften der Entität. Abbildung 4.4 veranschaulicht die genannte Definition. Das Diagramm bildet anfänglich die erste gelistete Eigenschaft der Entitäten ab; die erste Spalte in der Tabelle mit nummerischen Werten.

Das Responsive Web Widget wird in zweit Teile gegliedert. Ein Bereich beinhaltet das Kreisdiagramm, der andere Bereich dient als Informationsbereich. In vertikaler Orientierung wird das Kreisdiagramm im oberen Bereich des Bildschirmes angezeigt.

Durch die Swipe Geste auf die linke und rechte Seite kann zwischen den dargestellten Eigenschaften der Entität gewechselt werden. Das Vergrößern

[6] **C**ontent **M**anagement **S**ystem: dt. Verwaltungssystem für Onlineinhalte.

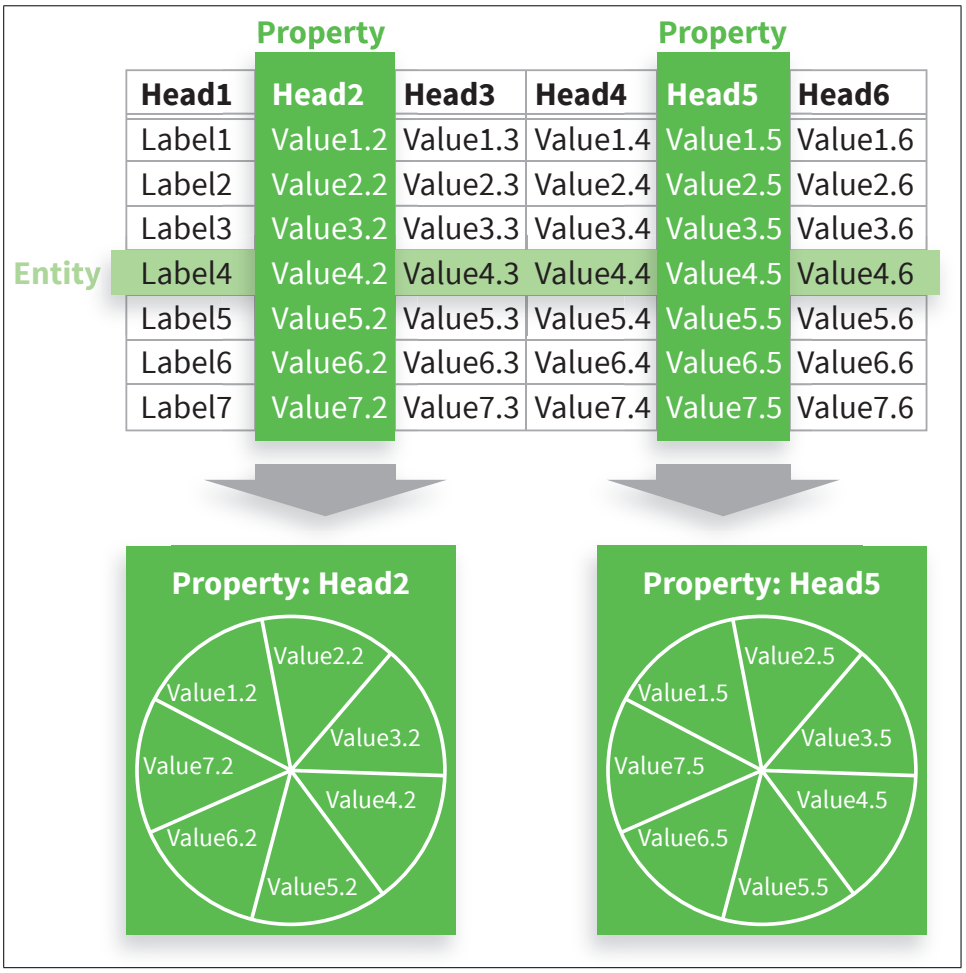


Abbildung 4.4: Die Abbildung zeigt die Verwendung der Daten im Kreisdiagramm. Die Spalten der Tabelle werden als Eigenschaften interpretiert. Für jede Eigenschaft wird ein eigenes Kreisdiagramm erstellt.

und Verkleinern des Diagrammes ist im oberen Bereich der Pinch open Geste bzw. mit der Pinch close Geste zu bewerkstelligen. Dies soll das Interagieren mit dem Diagramm – durch die größeren Flächen im eingezoomten Zustand – erleichtern.

Eine Long press Geste auf das Diagramm löst einen Wechsel in den Editiermodus aus. In diesem Modus reagiert das Diagramm auf weitere Touch-Gesten und wird um Interaktionsflächen - sogenannte Angreifer - erweitert. Die Interaktionsflächen liegen am Rand des Kreises an den Grenzen eines Sektors. Die Sektoren des Kreises können mit den Angreifern geändert werden. Die Grenzen des Sektors werden verschoben und der repräsentierte Wert wird geändert.

Sollen angrenzende Sektoren nicht unmittelbar von den Änderungen betroffen sein, können diese Sektoren gesperrt werden. Ein Sektor wird mit einer Double touch Geste gesperrt, damit kann das beschränkte Interagieren mit einem Sektor und das Gruppieren von Sektoren erreicht werden.

Die Gruppierung erfolgt automatisch mit dem Sperren eines Sektors. Da der gesperrte Sektor seine Fläche nicht verändert, wird die Wertänderung bei einer Drag Geste an den nächsten freien Sektor weitergegeben. Ein Verschieben der Sektoren entlang des Kreises ist somit möglich, allerdings kann die Reihenfolge der Sektoren nicht verändert werden. Das Entsperren eines Sektors funktioniert mit einer weiteren Double touch Geste.

Werte können nicht nur mit dem Angreifer geändert werden, sondern auch direkt über Eingabefelder. Eine Long press Geste auf die Anzeige, die die Summe der Eigenschaftswerte abbildet, schaltet die Editierfunktion für die Gesamtzahl entsperrt. Wird die Summe geändert, während kein Sektor selektiert ist, wird die Differenz von Ausgangswert und neuem Gesamtwert auf alle entsperrten Sektoren gleichmäßig aufgeteilt.

Sind nicht mindestens zwei Sektoren entsperrt, kann die Änderung der Gesamtzahl nicht durchgeführt werden. Ist während der Änderung der Gesamtzahl ein Sektor selektiert, wird dieser um die Differenz verändert. Dieses Vorgehen erlaubt eine Änderung im relativen oder absoluten Verhältnis von Gesamtzahl zu Sektoren.

Im unteren Bereich des Responsive Web Widgets sind die Informationen der Entitäten, wie Name und Wert, mit zugehöriger Farbcodierung zu sehen. Jedem Namen ist der absolute Wert der aktuellen Eigenschaft zugeordnet. Durch linke und rechte Swipe Bewegungen in diesem Bereich können alle Eigenschaftswerte der angezeigten Entitäten nachgeschlagen werden. Eine Tap Geste auf den Sektor einer Entität rückt die Wertinformation automatisch in den sichtbaren Bereich.

Über den Informationsbereich ist ebenfalls eine Änderung des Eigenschaftswertes bzw. der Sektorgröße möglich. Mit einer Long press Geste auf die Eigenschaft kann der Wert über ein Eingabefeld eingestellt werden. Die Änderung der Eigenschaft hat keine Auswirkungen auf die Gesamtzahl. Wird die Änderung übernommen, werden jene Entitäten beeinflusst, die nicht gesperrt sind.

Der Anteil der anderen Entitäten wird proportional zu ihren Werten verändert. Diese Funktion kann nur genutzt werden, wenn mindestens zwei Entitäten nicht gesperrt sind, da sonst die Änderungen nicht umgesetzt werden können. Nach dem Editieren müssen die Änderungen bestätigt oder verworfen werden. Eine Long press Geste auf das Diagramm bestätigt die Änderungen.

Das Speichern wird über eine *Callback-Funktion* in das Responsive Web Widget eingebunden, damit der Webentwickler entscheiden kann, wie der geänderte Datenbestand weiter verarbeitet werden soll. Sollen die Änderungen nicht gespeichert werden, wird über eine Schaltfläche das Kommando

zum Zurücksetzen des Datenbestandes aufgerufen.

Die Betrachtung des Responsive Web Widgets in horizontaler Ausrichtung führt zu Veränderungen der Platzverhältnisse. Das Responsive Web Widget soll auf diese Veränderungen reagieren. Mit einem flexiblen Layout wird die Größe des Diagramms an die Abmessungen der Höhe des Gerätes angepasst. Das Kreisdiagramm nimmt den linken Bereich des Gerätes in Anspruch. Auf der rechten Seite wird nun der Informationsbereich angezeigt. Die Interaktionsmöglichkeiten bleiben, bis auf zwei Ausnahmen, gleich.

Der Wechsel der Diagramme durch die Swipe Gesten erfolgt nicht mehr in der horizontalen Richtung, sondern in der vertikalen Richtung. Dies ist intuitiver, da das Diagramm aus dem Bildschirm gezogen wird und nicht in den Informationsbereich bewegt werden kann.

4.4.2 Balkendiagramm Widget

In Abschnitt 4.4.1 ist bereits erklärt, dass Reihen in Tabellen als Entitäten interpretiert werden. Im Gegensatz zu dem Kreisdiagramm Widget, werden im Balkendiagramm Widget nicht nur die Eigenschaften einer Entität dargestellt, sondern die Entität als Ganzes, siehe Abbildung 4.5. Dazu wird aus der gegebenen HTML-Tabelle das Datenmodell erstellt. Die Abbildung der Daten für Smartphones erfolgt im Hochformat. Die Entität wird mit ihrem Namen und ihren Eigenschaften dargestellt.

Jede Eigenschaft wird in Form eines Balkens abgebildet und erhält eine eindeutige Farbe. Damit wird gewährleistet, dass Balken mit der gleichen Farbe in unterschiedlichen Entitätsrepräsentationen dieselbe Eigenschaft abbilden. Die Länge des Balkens hängt von dem Wert der Eigenschaft ab. Die Höhe eines Balkens soll an die Gegebenheiten eines Touchscreens angepasst sein, damit mit einem Finger ein Balken selektiert werden kann.

Zu Beginn werden die Balken der Eigenschaften überlappend angezeigt, um Platz zu sparen. Die Entitäten werden untereinander gereiht, damit horizontales Scrollen vermieden wird. Die Reihung ist abhängig von der Reihenfolge in der HTML-Tabelle. Die Werte der Eigenschaften werden nach dem Initialisieren nach Größe sortiert, der größte Balken ist an der obersten Stelle, damit werden schnell Minimum und Maximum der Eigenschaften erkannt.

Durch die Double touch Geste auf eine Entität oder eine Tap Geste auf deren Namen, können die Balken auseinandergeklappt werden. Die nachfolgenden Entitäten werden nach unten bewegt. Eine weitere Double touch Geste auf die Entität oder eine Tap Geste auf den Namen stellt die überlappende Darstellung der Eigenschaftsrepräsentationen wieder her. Die danach abgebildeten Entitäten werden wieder an ihren Ursprungsplatz gesetzt. Auf diese Weise kann jede Entität in einen zusammengefalteten oder einen ausgeklappten Zustand versetzt werden.

Außerdem steht eine Filteroption zur Verfügung, die eine Eigenschaft

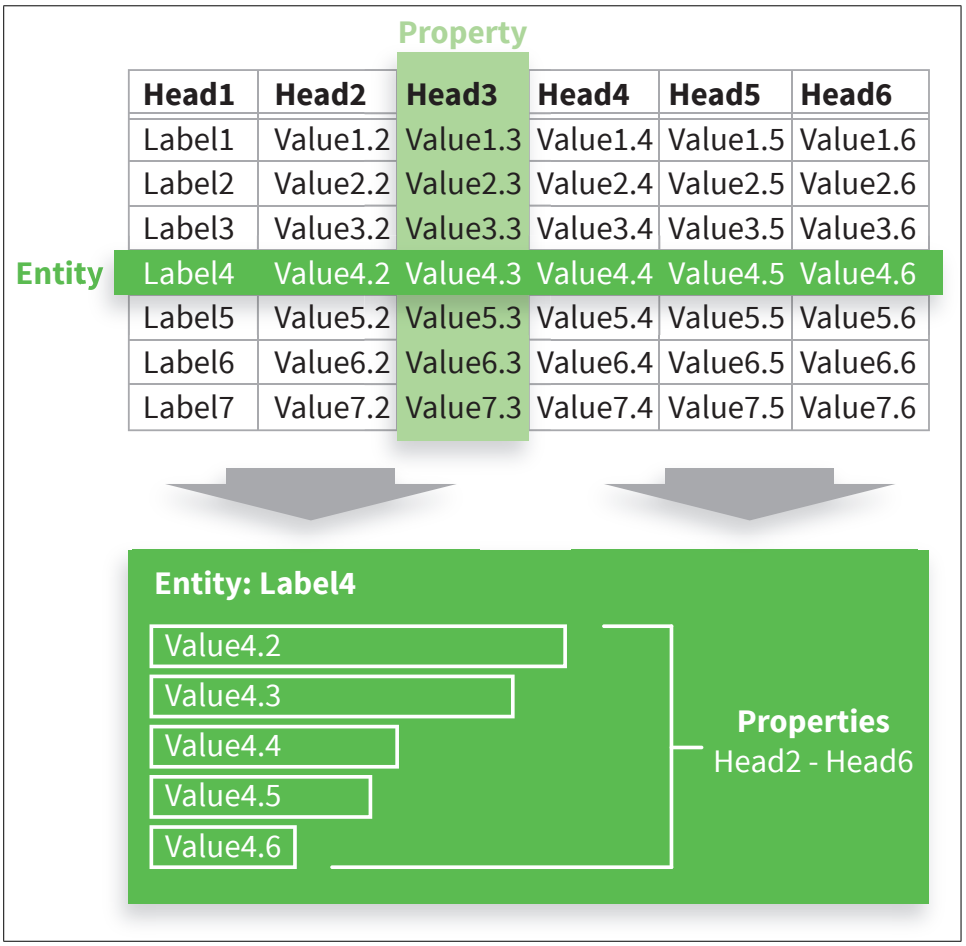


Abbildung 4.5: Die Reihen einer Tabelle werden als Entität abgespeichert. Eine Entität setzt sich aus mehreren Eigenschaften zusammen, welche als Balken dargestellt werden und gemeinsam die Repräsentation der Entität bilden.

pro Entität anzeigen kann. Die Entitäten rücken dabei näher zusammen, damit wird eine weitere Platzersparnis erreicht. Ist keine Filteroption ausgewählt, werden wieder alle Eigenschaften dargestellt und der Abstand der Repräsentationen der Entitäten wird wieder vergrößert.

Wird ein Balken einer Entität mit einer Tap Geste berührt, können Statistik-Funktionen der betreffenden Eigenschaft präsentiert werden. Dazu zählt das Maximum, das Minimum, der Median und der absolute Wert der Eigenschaft. Die Ergebnisse der Funktionen werden mithilfe einer Anzeige dargestellt. Die Anzeige erscheint nach einer Tap Geste auf einen entsprechenden Button. Tap Gesten auf andere Eigenschaften berechnet, die entsprechenden Statistik-Funktionen der Eigenschaften.

Das Editieren von Werten wird in einem Bearbeitungsmodus ermöglicht. Durch eine Long press Geste auf eine Entität oder ihren Namen, wechselt der Betrachtungsmodus in den Bearbeitungsmodus. Ein Anfasser wird am Ende eines Balkens angezeigt. Dieser Anfasser wird mit einer Drag Geste in horizontaler Richtung verschoben und ändert somit den Wert der Eigenschaft. Wird ein anderer Balken selektiert, springt der Anfasser auf diesen über und ermöglicht dessen Wert umzustellen. Während der ausgeführten Drag Geste wird der absolute Wert der Eigenschaft eingeblendet.

Im Bearbeitungsmodus wird mit einer Double touch Geste auf die Entität, das Zurücksetzen des Datensatzes erlaubt. Der Datensatz wird auf den Zustand zurückgesetzt, der bei dem Wechsel in den Bearbeitungsmodus aktuell war. Das Zurücksetzen der Daten ist mit dem Beenden des Bearbeitungsmodus verbunden.

Das Bestätigen der Änderungen wird mit einer Long press Geste auf die Entität durchgeführt. Gleichzeitig wird damit ebenfalls der Bearbeitungsmodus beendet und wechselt zurück in den Betrachtungsmodus.

4.4.3 Restriktionen

Da das automatische Parsen von Informationen aus Tabellen ein großes Themengebiet ist, werden im Umgang mit Responsive Web Widgets einige Parameter vorausgesetzt. Um das Funktionieren der prototypischen Entwicklung des Responsive Web Widgets zu gewährleisten, muss die Beschaffenheit einer HTML-Tabelle bestimmte Kriterien erfüllen:

- Die Tabelle muss semantisch korrekt aufbereitet sein.
- Die Tabelle muss dem HTML-Standard entsprechen.
- Zellen dürfen keine anderen HTML-Elemente beinhalten.
- Es dürfen keine Zellen miteinander verbunden sein.
- Jede Zelle muss befüllt sein.
- Es muss ein Tabellenkopf verfügbar sein, da dieser ausschlaggebend für die Bezeichnung der Werte in einer Spalte ist.
- Es sollen nur nummerische Werte in der Tabelle eingetragen sein. Ausnahmen sind die erste Spalte und der Tabellenkopf, hier sind aussagekräftige Bezeichner erwünscht.
- Die nummerischen Werte müssen positiv sein.
- Die Datenlage ist zweidimensional.
- Der Webentwickler ist verantwortlich für den Gebrauch des richtigen Responsive Web Widgets für die vorhandene Datenlage.

Kapitel 5

Umsetzung

Dieses Kapitel handelt von der Entwicklung der Responsive Web Widgets. Es werden die Anforderungen an Bibliotheken beschrieben, danach folgen Erklärungen zu den Entscheidungen über die verwendeten JavaScript-Bibliotheken. Des Weiteren werden die Konzepte bei der Umsetzung der Responsive Widgets erläutert. Es wird der Code in verschiedene Aufgabenbereiche eingeteilt, die das Implementieren der Responsive Web Widgets erleichtern sollen.

Danach werden der Initialisierungsprozess und das Zusammenwirken der Bibliotheken untereinander ausgeführt. Ein zentraler Punkt dieses Kapitels ist die Rolle der Entitäten, die für den Ansatz der Diagrammdarstellung wichtig sind. Codestrukturen im Zusammenhang mit Benachrichtigungen zum Informationsaustausch zwischen einzelnen Komponenten werden ebenfalls abgehandelt.

5.1 Architektur eines Responsive Web Widgets

Die Umsetzung eines Responsive Web Widgets ist in verschiedene Aufgabenbereiche unterteilt. Zum einen ist auf die Strukturierung des Programmes wert zu legen. In Scriptsprachen wie JavaScript entsteht schnell langer, unübersichtlicher Quellcode, da keine Klassen verfügbar sind. Ist keine Struktur vorhanden, sind Fehler erschwert aufzuspüren und auszubessern.

Ein anderer Aufgabenbereich behandelt Eingabemethoden. Diese müssen auf verschiedenen Geräten Aktionen auslösen, unabhängig von den vorhandenen Eingabemöglichkeiten des Gerätes. Die Darstellung des Responsive Web Widgets hat die Aufgabe ein umfassendes Datenaufkommen möglichst auflösungsunabhängig zu repräsentieren. Dies wird über Vektorgrafiken erreicht, die an die Maße des Ausgabegerätes anpassbar sind. Das *MVVM*-Paradigma wird für die Umsetzung des Responsive Web Widgets verwendet, um diese Aufgabenbereiche abzubilden.

5.1.1 Code-Strukturierung

Der Programmcode soll übersichtlich und wiederverwendbar sein, damit das Erweitern von Funktionen und das Weitergeben von Codeabschnitten gewährleistet werden kann. Aus diesem Grund ist der Code entsprechend seiner Verwendungszwecke aufzuteilen. Eine gute Strukturierung erleichtert das nachträgliche Implementieren von Zusatzfunktionen, die erst mit fortschreitendem Verlauf eines Projektes notwendig werden.

Im Idealfall sind die Kompetenzen der Programmteile voneinander abgekapselt. Das macht die Komponenten austauschbar und ein auftretender Fehler kann in der Applikation einfacher lokalisiert werden. Die Fehlersuche wird dadurch erleichtert, dass bei einer strengen Einteilung der Verantwortlichkeiten, die Codekomponenten in ihrem Aufbau besser lesbar sind.

Die Strukturierung soll die drei Codebereiche Gestaltung, Verhalten und Datenaufbereitung klar voneinander abgrenzen. Die Responsive Web Widgets sollen in ihren Funktionalitäten erweiterbar sein und eine Anbindung für eigene Gestaltungskonzepte bieten. Trotz der losen Kopplung der Komponenten muss die Kommunikation zwischen diesen Bereichen gewährleistet sein.

Ein gängiges Architektur-Paradigma, das diese Anforderungen erfüllt, ist die *Model-View-Controller*-Architektur. Für interaktionslastige Applikationen ist jedoch das *Model-View-Viewmodel*-Paradigma vorzuziehen, da es die Präsentationsschicht nicht stark bindet. MVVM ist mit der MVC-Architektur verwandt, bietet aber eine strengere Trennung der einzelnen Komponenten, was einen leichteren Austausch der View-Komponente ermöglicht.

5.1.2 Model-View-Viewmodel (MVVM)

Das Konzept Model-View-Viewmodel ist eine Abwandlung des MVC- und des MVP[1]-Paradigmas. MVVM wird dazu verwendet, Softwarekomponenten voneinander zu entkoppeln und dadurch wiederverwendbar zu gestalten. Durch die Anwendung des MVVM Konzeptes wird der Programmcode strukturiert aufgebaut, die Komponenten des MVVMs sind voneinander unabhängig austauschbar, solange die Schnittstellen eingehalten werden. Das MVVM besteht aus den drei Komponenten Model, View und Viewmodel und wird hauptsächlich für Benutzerschnittstellen-Programmierung angewendet und nach [41] und [5, Kap. 11] beschrieben.

Die Model-Komponente

Die Komponente beinhaltet die Business-Logik einer Applikation. In der Model-Komponente werden die Daten abgespeichert, die die Applikation

[1] **M**odel-**V**iew-**P**resenter: Weiterentwicklung von MVC

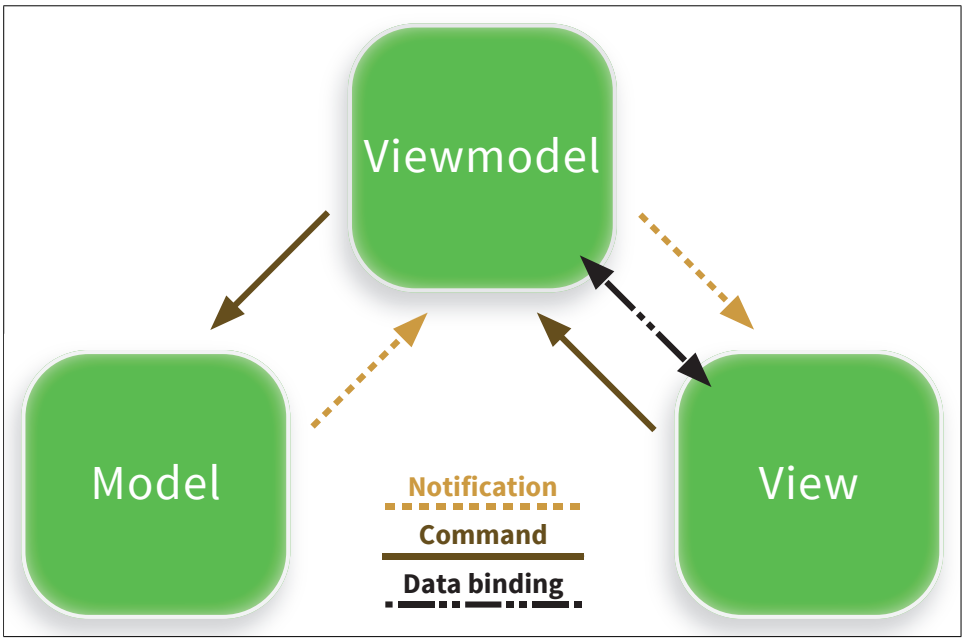


Abbildung 5.1: Schematische Darstellung des MVVM-Konzeptes. Die View-Komponente gibt Benutzeraktionen als *Command* an die Viewmodel-Komponente weiter. Die Viewmodel-Komponente entscheidet, welche Änderungen an die Model-Komponente weitergegeben werden. Ändert die Model-Komponente den Status, wird die Viewmodel-Komponente über eine *Notification* verständigt. Die Änderung wird von der Viewmodel-Komponente ebenfalls über eine *Notification* an die View-Komponente weitergegeben, welche nun den aktuellen Datenbestand für den Benutzer darstellt. Daten, die nur Viewmodel und View teilen, werden über *Data-Bindings* miteinander synchronisiert.

verwendet und von dem Rest der Applikationskomponenten gekapselt. Für den Zugriff auf die Model-Komponente werden Schnittstellen definiert. Die Verwendung einer Model-Komponente stellt sicher, dass die Daten in einem konsistenten Zustand verwaltet werden und immer den neuesten Zustand wiedergeben.

Wird der Datenbestand einer Model-Komponente verändert, verständigt die Model-Komponente jene Komponenten, die die Präsentationsschicht der Applikation bilden, damit diese die Model-Repräsentation auf den aktuellen Stand ändern. Das MVVM-Konzept sieht die Validierung von Objekteigenschaften als Aufgabe der Model-Komponente an. In der MVC-Architektur wäre der Controller für die Validierung zuständig. .

Die View-Komponente

Sie bildet die Präsentationsschicht in dem Entwurfsmuster. Sie zeigt die Daten der Model-Komponente an. Jede Model-Komponente kann mehrere View-Komponenten haben. Benutzer können über die View-Komponente mit dem Datenmodell interagieren. Die View-Komponente dient als Schnittstelle um den Datenbestand der Model-Komponente zu ändern und hat somit Zugriff auf die Eigenschaften einer Model-Komponente, die das Lesen und Schreiben von Daten erlauben. Benutzereingaben werden von der View-Komponente an die Viewmodel-Komponente weitergegeben. Die View-Komponente ist für eine aktuelle Repräsentation des Datenmodells verantwortlich.

Die Viewmodel-Komponente

Die Viewmodel-Komponente dient als spezieller Controller zwischen der Model-Komponente und der View-Komponente. Die Viewmodel-Komponente erledigt den Datenaustausch von View zu Model und vice versa. Sie ist als direkte Abstraktionsschicht hinter der View-Komponente anzusehen und verarbeitet die Benutzeraktionen. Die View-Komponente und die Viewmodel-Komponente kommunizieren über Events und Data-Bindings.

Der Unterschied zwischen Viewmodel- und Model-Komponente ist, die Zwischenspeicherung der Daten, mit denen der Benutzer gerade arbeitet, in der Viewmodel-Komponente. Die Viewmodel-Komponente ist weder ein reines Datenmodell, da das Verhalten der View-Komponente gesteuert wird, noch eine reine Repräsentation, da keine grafische Darstellung zur Verfügung gestellt wird. Die Viewmodel-Komponente wird als Container, von Daten und Operationen, die auf eine Benutzerschnittstelle angewendet werden können, angesehen.

5.1.3 Benutzeraktionen

Für die Umsetzung der Responsive Web Widgets ist die Verarbeitung von verschiedenen Eingabemöglichkeiten unentbehrlich. Vor allem das Umwandeln von Mausklicks in Touch-Gesten ist von Bedeutung bei der Entwicklung von Responsive Web Widgets. Die Umwandlung dieser Eingabemethoden kann auf verschiedenen Geräten einige Schwierigkeiten mit sich bringen, da Touch-Gesten und Mausklicks nicht dieselben *Eventphasen* haben, wie bereits in Abschnitt 4.2.2 beschrieben wurde. Da mittlerweile schon Geräte auf dem Markt sind, die sowohl Touch-Gesten als auch Mausklicks verarbeiten können, ist es umso wichtiger auf eine Bibliothek zu setzen, die mit diesem Verhalten plattformunabhängig umgehen kann.

Das Erkennen von etablierten Touch-Gesten, wie der Long press Geste, Swipe Geste und Double touch Geste muss unterstützt werden. Außerdem soll gewährleistet werden, dass das An- und Abmelden von *Eventlisteners*

in einer simplen Art und Weise vonstattengehen kann. Die Bibliothek sollte nicht zu viel Speicherplatz verwenden, da sie nur für das Erkennen von Benutzeraktionen verantwortlich sein soll. Das Umwandeln von Eingaben in Klick- oder Touch-Events soll, je nach verwendeter Plattform, automatisch vorgenommen werden. Die Bibliothek muss mit SVG-Elementen kompatibel sein. Sie muss als Standalone-Variante vorliegen, da sie keine Abhängigkeiten zu anderen Bibliotheken aufweisen darf, weil das die Funktionalitäten des Responsive Web Widgets einschränken könnte.

5.1.4 Darstellung

Viele Bildschirme von Smartphones und Tablets haben verschiedene Auflösungen, Seitenverhältnisse und Bildpunkte. Da die Darstellung mit festen Maßeinheiten aus diesem Grund nicht ratsam ist, soll die Datenrepräsentation auf kleinen Bildschirmen mit SVG erfolgen. SVG unterstützt, als ein vektorbasierendes Grafikformat, die Verwendung von Prozent- und `em`-Angaben. Die gezeichneten Elemente können durch JavaScript angesprochen und verändert werden. Dadurch sind sie für die Erstellung von konfigurierbaren Interaktionselementen brauchbar, welche für den Einsatz auf verschiedenen Plattformen geeignet sind. Da SVGs aus Text bestehen, ist der Speicherbedarf nicht sehr hoch und für mobile Geräte geeignet. Deswegen sind SVGs gut in bestehendes HTML-Markup einzubetten.

Die Bibliothek soll den SVG-Standard implementieren und eine für sich sprechende API vorweisen. Eine Fallbackfunktion für ältere Internet Explorer Versionen wäre wünschenswert, da Versionen vor IE9 den SVG-Standard nicht unterstützen. Das Erstellen von grundlegenden Geometrien wie Kreise und Rechtecke soll mit einem Aufruf erledigt werden können. Die API soll schnelle Zugriffsmethoden für gezeichnete Elemente bieten und das Gruppieren von Elementen ermöglichen. Die API muss relative Maßeinheiten zulassen können, um den Vorteil von Vektorgrafiken anwenden zu können. Das An- und Abmelden von *Eventlisteners* für Mausklicks und Touch-Gesten muss möglich sein. Funktionen, die das Löschen eines Elementes erlauben, müssen ebenfalls zur Verfügung gestellt werden.

5.1.5 Unobtrusive JavaScript

Mit diesem Begriff wird das Konzept beschrieben, die Benutzererfahrung von Webseiten mit JavaScript aufzuwerten. Jedoch sollte dies in einem Umfang erfolgen, in welchem JavaScript nicht zwingend für den Gebrauch der Webseite notwendig ist. JavaScript kann von manchen Webbrowsern nicht oder nur teilweise interpretiert werden, oder der Benutzer hat die Ausführung von JavaScript untersagt, oder die Applikation kann nicht bedient werden, weil die angedachten Eingabemethoden nicht vorhanden sind. Deswegen ist es in der Webentwicklung empfehlenswert die HTML-Struktur von

JavaScript-Code zu trennen und den Code in eine externe Javascript-Datei abzulegen und mit einem `<script>`-Element im HTML-Dokument zu referenzieren [34].

Für den Fall, das JavaScript ausgeschalten ist, sollten Ersatzmaßnahmen getroffen werden, die ohne JavaScript auskommen. Dabei ist es wichtig, dass die Webseite nicht die Funktionalitäten des JavaScript-Codes zu imitieren hat, sondern einen geregelten Ablauf der Restfunktionen gewährleisten soll. Dies kann zu Leistungseinbußen und zur Verringerung von Benutzerfreundlichkeit führen. Inhalt und Navigation einer Webseite sollen stets ohne JavaScript-Funktionen ersichtlich bzw. ansteuerbar sein. Die Anzeige von Inhalten sollte nicht von JavaScript abhängig gemacht werden. Das Laden von Webseiten sollte ohne AJAX[2] ebenfalls funktionieren. Des Weiteren ist zu beachten, dass das HTML-Markup semantisch korrekt aufgearbeitet ist, um den Aufwand bei der JavaScript-Entwicklung möglichst gering zu halten. Eine hohe Webbrowserkompatibilität ist mithilfe von JavaScript-Bibliotheken zu erreichen. Die Verwendung von Bibliotheken erleichtert den Umgang mit Eigenheiten der Webbrowser, da die Bibliotheken in den meisten Fällen ausgiebig getestet wurden.

5.2 Bibliotheken Auswahl

Im Folgenden werden Javascript Bibliotheken vorgestellt, die in der Implementierung der Responsive Web Widgets verwendet werden und den oben angeführten Anforderungen entsprechen. Diese Auswahl wurde basierend auf Kapitel 3 sowie dem passenden Anforderungsprofil der Bibliotheken getroffen.

5.2.1 Knockout JS

Der Quellcode des Responsive Web Widgets benötigt eine klare Trennung der Bereiche Datensatz, Präsentation und Verhalten. Für diese Trennung ist Knockout verantwortlich. Die Bibliothek ist eine JavaScript Implementierung des MVVM-Paradigmas und als freie Software verfügbar. Knockout bietet die Möglichkeit den Quellcode zu strukturieren, indem es Vorgaben für Model, View und Viewmodel stellt. Dadurch ist eine lose Kopplung der Komponenten möglich. Die View-Komponente wird mittels deklarativen Anweisungen an die Viewmodel-Komponente gebunden. Das minimiert den Codeaufwand, macht aber die Fehlersuche etwas undurchsichtiger.

Mit Knockout ist es möglich *Templates* anzulegen, die für wiederkehrende Benutzerschnittstellen-Komponenten nützlich sind. Außerdem ist es durch die geringe Dateigröße für den mobilen Gebrauch geeignet.

[2] **A**synchronous **J**avaScript **a**nd **X**ML

Das MVVM-Paradigma hat den Nachteil, dass es für kleine Benutzerschnittstellen-Projekte überdimensioniert ist, und für zu große Projekte zu viel Entwicklungsaufwand darstellt. Die Responsive Web Widgets erfüllen in ihrem Ausmaß die Anforderungen, sodass das MVVM-Paradigma eine gute Entscheidung in Hinblick auf Code-Strukturierung und Implementierungsaufwand ist.

5.2.2 Hammer JS

Hammer ist eine Bibliothek, die das Auslösen von Touch-Gesten erkennen kann und verarbeiten kann. Es werden Single-Touch-Gesten und Multi-Touch-Gesten erkannt. Sie ist als Standalone Bibliothek konzipiert, ist aber auch als jQuery Plug-in erhältlich. Die Dateigröße benötigt nicht sehr viel Speicherplatz, was ein schnelleres Laden der Seite ermöglicht. Das An- und Abmelden von *Eventlisteners*, die auf Maus- oder Touch-Events hören, ist mit wenig Code zu realisieren. Es werden Elemente mit CSS-Selektoren angesteuert um *Eventlistener* anzumelden. Die Hammer Bibliothek erzeugt für jedes Element, das auf Touch-Events reagieren soll, eine neue Instanz. Die Instanz ist für das Verwalten von angemeldeten *Eventlisteners* eines Elementes verantwortlich.

5.2.3 Raphaël JS

Raphaël ist eine JavaScript-Bibliothek, deren Aufgabe die dynamische Erstellung von SVG-Markup ist. Der Initialisiserungsprozess erstellt ein JavaScript-Objekt, welches als `paper` bezeichnet wird. Die `paper`-Variable ist der zentrale Zugriffspunkt auf das `<svg>`-Element im HTML-Markup. Die `paper`-Variable stellt Funktionen bereit, die das Erzeugen von Elementen und das Einfügen der Elemente in den DOM übernehmen. Die Repräsentationen der Elemente sind als Unterelemente des `<svg>`-Elementes in den DOM eingehängt.

Raphaël bietet die Möglichkeit, Elemente in Sets zu gruppieren. Sets sind Datenstrukturen, die sich ähnlich verhalten wie Arrays, jedoch einige spezifische Funktionen der Raphaël-Bibliothek bieten. Funktionen, die üblicherweise für Elemente vorgesehen sind, können durch Sets aufgerufen werden und werden auf die im Set gespeicherten Elemente angewandt. Raphaël kann relative Maßeinheiten verarbeiten.

Nicht alle Standard-Elemente können mit einem Befehl erzeugt werden. Es ist z. B. nicht möglich, `<g>`-Elemente zu erzeugen. Eine Linie kann nicht als Element vom Typ `line` angelegt werden, sondern muss über ein `<path>`-Element erzeugt werden. `<path>`-Elemente können im SVG-Standard jedoch nicht mit relativen Maßeinheiten umgehen, d. h., eine Linie mit relativen Angaben muss als dünnes Rechteck gezeichnet werden. Für Internet Explorer Versionen, die keinen SVG-Standard implementieren, wird im VML-Format

Programm 5.1: Die `viewport`-Metainformation definiert den Ausschnitt der abzubildenden HTML-Seite. In diesem Fall in Originalgröße, ohne Standard-Zoomfunktion.

```
1 <head>
2 ...
3   <meta name="viewport" content="width=device-width, initial-scale=1.0,
      minimum-scale=1.0, maximum-scale=1.0 , user-scalable=no" />
4 ...
5 </head>
```

gezeichnet. Außerdem ist Raphaël als einer der wenigen SVG-Bibliotheken durchgehend dokumentiert, und mit vielen Beispielen versehen.

5.3 Responsive Web Widget Implementierung

Dieser Abschnitt beschreibt die technischen Herausforderungen bei dem Entwickeln der Responsive Web Widgets. Die Umsetzung erfolgt durch die Webtechniken HTML, CSS und JavaScript, wobei die semantische Auszeichnung der Inhalte von HTML übernommen wird. CSS ist für die flexible Layoutanpassung und die Umsetzung der gestalterischen Komponenten verantwortlich. Die Benutzeraktionen werden mit JavaScript verwaltet, außerdem werden die SVGs dynamisch mit JavaScript erstellt. Der Aufbau des JavaScript-Codes wird mit dem MVVM-Paradigma strukturiert.

5.3.1 HTML-Markup

Das HTML-Markup ist simpel gehalten. Zu Beginn des HTML-Markups wird in der `<head>`-Sektion der Viewport, wie in Prog. 5.1 abgebildet, definiert. Der Viewport wird, wie in Abschnitt 3.3.1 beschrieben, von Webbrowsern auf Desktop-Computern ignoriert. Auf mobilen Geräten werden die Informationen verwendet um die Eigenschaften des Sichtbereiches im Webbrowserfenster zu definieren. Die Einstellungen für die Responsive Web Widgets sind:

1. das Standardverhalten des Webbrowser-Zoomes zu unterbinden,
2. die Breite der Webseite auf die Bildschirmbreite anzupassen und
3. den Vergrößerungsfaktor auf Originalgröße zu setzen.

Prog. 5.2 zeigt das Grundgerüst des Responsive Web Widgets. Als Widget-Container dient ein `<div>`-Element; als Wurzel-Element beinhaltet es alle anderen Elemente, die für den Aufbau des Responsive Web Widgets nötig sind. Die Grundstruktur teilt sich in Navigationsbereich, Datenbereich und Zeichenbereich.

Programm 5.2: Die Grundstruktur eines Responsive Web Widgets. Die Zuständigkeiten des Responsive Web Widgets sind in die Bereiche Navigation, Datenbereitstellung und Datendarstellung aufgeteilt.

```
<body>
...
  <div class="widget">
    <div id="widgetContent">
      <nav>
        <h1 class="stats">Bar Chart</h1>
      </nav>
      <div id="filterArea"></div>
      <div id="drawArea"></div>
    </div>
    <div class="dataContent">
    </div>
  </div>
...
</body>
```

Der Navigationsbereich ist mit dem `<nav>`-Element ausgezeichnet und dient dazu, Elemente wie `<a>`-Elemente zu enthalten. Im `<div>`-Element, welches im Beispiel als `drawArea` bezeichnet ist, werden auf mobilen Geräten die Grafiken dargestellt. Hierzu werden SVGs erstellt und dynamisch in den DOM geladen. Die `drawArea` ist das Elternelement der Interaktionsflächen.

Die `filterArea` ist dafür bestimmt, eine Benutzerschnittstelle bereitzustellen, die Aktionen für das Aus- und Einblenden von Informationen bietet. Das `<div>`-Element, mit dem `id`-Attribut `dataContent`, ist das Element, worin sich die Daten befinden werden. In den exemplarischen Responsive Web Widgets müssen diese Daten in HTML-Tabellen vorliegen, siehe Prog. 5.3.

Dabei müssen die Tabellen einen bestimmten Aufbau erfüllen. In Abschnitt 4.4.3 sind die Voraussetzungen für die Tabellenaufbereitung und deren Restriktionen beschrieben.

Die Daten der Tabelle werden nach dem Laden der Seite analysiert und mit JavaScript in Entitäten umgewandelt. Die Entitäten bilden den Datensatz in objektorientierter Form ab. Dieser Vorgang wird in Abschnitt 5.3.3 behandelt.

Der Kopf einer Tabelle stellt die Bezeichner für die Namen der Entitäts-Attribute bereit. Der Tabellenkörper ist die wesentliche Datenquelle. Die Zellen müssen nummerische Inhalte enthalten, welche als Werte der Entitäts-Attribute dienen. Die erste Spalte der Tabelle legt die Namen der Entitäten fest, jede Zeile in der Tabelle bildet eine spätere Entität ab.

Programm 5.3: Der Code zeigt den Aufbau einer HTML-Tabelle, wie sie für das Responsive Web Widget geeignet ist. Die erste Kopfzelle beinhaltet den Typ der Entitäten, alle weiteren Kopfzellen bilden die Namen der Entitäts-Attribute ab. Im Tabellenkörper folgen die Bezeichner der Entitäten mit den Werten der Entitäts-Attribute.

```
<div class="widget">
...
  <div class="dataContent">
    <table id="dataSet">
      <thead>
        <tr>
          <th>Team</th><th>Platz</th><th>Spiele</th>
          <th>Sieg</th><th>Unentschieden</th><th>Niederlage</th>
          <th>Tore Geschossen</th><th>Tore Bekommen</th><th>Punkte</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>FK Austria Wien</td><td>1</td><td>25</td>
          <td>19</td><td>4</td><td>2</td>
          <td>61</td><td>18</td><td>61</td>
        </tr>
        ...
      </tbody>
    </table>
  </div>
...
</div>
```

5.3.2 Präsentation

Das Layout der Responsive Web Widgets ist mit CSS realisiert. Die CSS-Regeln sind so ausgerichtet, dass sie von modernen Webbrowsern unterstützt werden. Vor den, für die Responsive Web Widgets, relevanten Regeln wird ein CSS-Reset durchgeführt. Der CSS-Reset stammt von Eric Meyer [13] und ist dafür verantwortlich, dass das Standard-Layout des Webbrowsers zurückgesetzt wird.

Die gängigen Webbrowser haben ähnliche Standard-Layouts, diese können sich aber im Detail unterscheiden, oder sind durch Benutzereinstellungen verfremdet. Der CSS-Reset setzt diese Unterschiede zurück und macht es möglich, ausgehend von einer einheitlichen Ausgangsbasis, neue CSS-Regeln zu definieren. CSS-Resets sind sparsam einzusetzen, da sie Leistungseinbußen verursachen, weil sie die Eigenschaften eines jeden Elements editieren.

Das Layout soll sich je nach Bildschirmgröße an dessen Dimensionen anpassen. Aus diesem Grund werden im gesamten Stylesheet nur relative Maßeinheiten für Angaben verwendet. Das Verhalten von Elementen mit relati-

Programm 5.4: Abgebildet ist ein Ausschnitt eines CSS-Resets. Der CSS-Reset von Eric Meyer [13] trägt dazu bei, dass die Standard-Layouts der Webbrowser zurückgesetzt werden. CSS-Resets können Ladezeiten einer Webseite negativ beeinflussen.

```
1 html, body, div, span, applet, object, iframe,
2 h1, h2, h3, h4, h5, h6, p, blockquote, pre,
3 a, abbr, acronym, address, big, cite, code,
4 del, dfn, em, img, ins, kbd, q, s, samp,
5 small, strike, strong, sub, sup, tt, var,
6 b, u, i, center,
7 dl, dt, dd, ol, ul, li,
8 fieldset, form, label, legend,
9 table, caption, tbody, tfoot, thead, tr, th, td,
10 article, aside, canvas, details, embed,
11 figure, figcaption, footer, header, hgroup,
12 menu, nav, output, ruby, section, summary,
13 time, mark, audio, video {
14     margin: 0;
15     padding: 0;
16     border: 0;
17     font-size: 100%;
18     font: inherit;
19     vertical-align: baseline;
20 }
21 ...
```

ven Angaben ist abhängig von den Eigenschaften ihrer Vorgänger-Elemente. Eine Breite von `100%` eines Elementes bedeutet, dass es die Weite seines Eltern-Elementes annimmt und nicht die Breite des Bildschirmes. Aus diesem Grund ist es notwendig die Höhe, Weite und Schriftgröße des `<html>`- und `<body>`-Elementes auf `100%` zu setzen. Angaben in der `em`-Maßeinheit beziehen sich auf die Schriftgröße, Prozentangaben auf die Ausmaße des Vorgängerelementes. Bei variierenden Dimensionen der Anzeigegeräte können die Elemente so ihre Abmessungen dynamisch ändern.

```
1 ...
2 html, body{
3   font-size: 100%;
4   height: 100%;
5   width: 100%;
6 }
7 ...
```

5.3.3 Verhalten

Die Responsive Web Widgets sind, unter der Verwendung der oben genannten Bibliotheken, mit JavaScript umgesetzt. Im Sinne der Unobtru-

sive JavaScript-Programmierung wird darauf verzichtet, im HTML-Markup JavaScript-Code zu verwenden. Der gesamte Code, der für die Generierung des Responsive Web Widgets zuständig ist, befindet sich in ausgelagerten JavaScript-Dateien, welche innerhalb des HTML-Markups referenziert werden. Im folgenden Abschnitt wird die Funktionsweise von wichtigen Methoden erklärt und beschrieben, wie die verschiedenen Bibliotheken miteinander kommunizieren.

Initialisierung

Das Responsive Web Widget wird über den Aufruf `RWW.createWidget()` erzeugt. Der Konstruktor benötigt für die Generierung des Responsive Web Widgets den Namen zweier `id`-Attribute. Die erste `id` referenziert auf die Daten, die das Responsive Web Widget verwenden soll. Diese Daten müssen in einer HTML-Tabelle abgelegt sein, und entsprechend der Restriktionen in Abschnitt 4.4.3 aufbereitet sein.

Die zweite `id` benennt das HTML-Element, in welchem das Responsive Web Widget gezeichnet werden soll. Für dieses Element bietet sich ein `<div>`-Container an. Die Anforderungen, die die Implementierung des Responsive Web Widgets voraussetzt, sollte jedoch von jedem Block-Element, welches andere Blockelemente beinhalten darf, erfüllt werden können.

In dem Konstruktor wird die Viewmodel-Komponente erzeugt, welches wiederum für die Erstellung von Entitäten und Repräsentationen verantwortlich ist. Danach werden die Repräsentationen der Entitäten in der `draw`-Funktion gerendert. Dafür wird eine Funktion der Knockout Bibliothek verwendet, die die Möglichkeit bietet, *Templates* zu verwenden.

Die Templates sind Codestücke, die das Aussehen und die HTML-Struktur der Repräsentationen beeinflussen. Jede Repräsentation wird mithilfe des Templates dargestellt, und durch die Viewmodel-Komponente mit den spezifischen Eigenschaften ausgestattet. Die Werte dieser Eigenschaften der Repräsentationen sind wiederum abhängig von den Werten der zugehörigen Entitäten.

Durch `ko.applyBindings()` wird die Knockout Bibliothek aktiviert und der eben beschriebene Vorgang ausgelöst. Die Knockout Bibliothek erkennt dann Abhängigkeiten von Variablen, Objekten und Funktionen und löst diese automatisch aus. Außerdem können *Data-Bindings* verwendet werden, um weitere Eigenschaftsabhängigkeiten darzustellen. Diese werden zum Teil im Template-Markup eingetragen, oder dynamisch mit JavaScript Code erzeugt.

Die Funktion `applyEventHandlers()` wird nach Abschluss der Knockout Initialisierung aufgerufen. In dieser Funktion werden sämtliche Interaktionsmöglichkeiten angemeldet, die für das Bearbeiten der Daten mit dem Responsive Web Widget benötigt werden.

Entität – die Model-Komponente

Die Entität ist die Model-Komponente des MVVM-Paradigmas. In ihr sind die relevanten Daten gespeichert, die zuvor von der Viewmodel-Komponente geparst werden. Eine Entität bekommt eine `id` zugewiesen, die der Reihe im Tabellenkörper entspricht. Außerdem besteht jede Entität aus einem Array von Attributen. Die Attribute sind mit Eigenschaften der Entität gleichzusetzen. Das Attribut-Array beinhaltet eine Liste von JavaScript-Objekten.

Jedes JavaScript-Objekt ist ein Attribut, das aus einem Bezeichner und einem Wert besteht. Der Bezeichner enthält den Namen der Eigenschaft, d. h. den Spaltennamen aus dem Tabellenkopf. Die Höhe des Wertes wird von dem Zelleninhalt bestimmt, welcher aus dem Schnittpunkt von Entität und Eigenschaft resultiert.

Im weiteren Verlauf werden die JavaScript-Objekte in dem Attribut-Array nach Größe sortiert. Als Referenz wird der Wert des Attributes verwendet. Die Sortierung erfolgt über die Viewmodel-Komponente, da die Entität nur als Datenspeicher dienen soll und über keine Funktionen verfügen soll, die den Datenbestand ändern könnten. Die Entitäten kennen ihre Repräsentationen nicht, es ist keine Referenz zu einer zugehörigen Repräsentation angelegt, da es Aufgabe der Viewmodel-Komponente ist, Repräsentationen mit den Daten der Entitäten zu bestücken.

Verwaltung – die Viewmodel-Komponente

Die Viewmodel-Komponente ist für die Verwaltung des Responsive Web Widgets zuständig. Sie verwaltet die Daten aus den Tabellen und hält die Repräsentationen der Entitäten auf dem aktuellen Stand. Im Verlauf der Erstellung der Viewmodel-Komponente wird die Tabelle, die anfangs dem Responsive Web Widget bekannt gemacht wurde, verarbeitet. Dabei wird der Kopf der Tabelle als Bezeichner für die Entitäts-Eigenschaften verwendet und die Reihen des Tabellenkörpers als Entitäten interpretiert. Sind die Werte in den Zellen der Tabellenreihe zulässig, erzeugt die Viewmodel-Komponente eine Entität. Dieser Vorgang wird bis zum Ende der Tabelle wiederholt.

Die Viewmodel-Komponente merkt sich eine Liste von Entitäten, damit im Laufe der Verarbeitung weiterhin der Zugriff auf die Eigenschaften der Entitäten möglich ist. Nach Abschluss des Vorganges und der Speicherung der Entitäten sind der Viewmodel-Komponente Anzahl und Aufbau der Entitäten bekannt.

Die Entitäten werden nun von der Viewmodel-Komponente dazu verwendet, die Repräsentationen zu erzeugen. In der `setupDrawObjects`-Methode wird durch die Liste der Entitäten iteriert und in jedem Schritt wird eine Repräsentation für die jeweilige Entität erzeugt. Die Viewmodel-Komponente kann durch gegebene Größen den Platzbedarf berechnen.

Da die View-Komponente für die Balkendarstellung verschiedene Zustände (Aktiv, inaktiv, ausgeklappt, Filter aktiv, etc.) annehmen kann, muss die Zeichenfläche in ihrer Dimension angepasst werden. Die verschiedenen Größen für die Zustände, werden ebenfalls vorberechnet und gespeichert um einen schnelleren Zugriff zu gewährleisten und Berechnungen einzusparen.

Nach der Aktivierung von Knockout werden die berechneten Eigenschaften für die Darstellung am Bildschirm an die Repräsentationen übermittelt. Der `Template`-Mechanismus greift auf die Liste der Repräsentationen in der Viewmodel-Komponente zu und legt für die Repräsentation das HTML-Markup an.

Nachdem das HTML-Markup angelegt ist, wird über Raphaël das SVG erzeugt. Raphaël verwendet dazu die Darstellungsdaten, die zuvor von der Viewmodel-Komponente berechnet worden sind. Die Repräsentation wird, wie die Entitäten, in einer Liste der ViewModel-Komponente abgespeichert.

Entitäten und Repräsentationen, die über die Viewmodel-Komponente verbunden werden, sind auf demselben Index in der jeweiligen Liste abgelegt. Aus den Ids der Repräsentationen oder der Entitäten, kann der Index erzeugt werden, über den sie in der Viewmodel-Komponente ansteuerbar sind. Die Viewmodel-Komponente stellt mehrere Funktionen zur Verfügung, die das Bearbeiten des Datenbestandes (Entitäten) über die View-Komponente (Repräsentation) ermöglichen.

Benutzerschnittstelle – die View-Komponente

Die Repräsentations-Objekte stellen die Benutzerschnittstelle dar. Nach abgeschlossenem Initialisierungsprozess der Viewmodel-Komponente und der Aktivierung von Knockout, werden die Repräsentationen auf dem Bildschirm angezeigt. In der Funktion `setupCanvas` wird eine neue Raphaël `paper`-Variable angelegt, welche der Erstellung eines `<svg>`-Elementes entspricht. Die Viewmodel-Komponente übergibt in diesem Vorgang die berechneten Werte für die Darstellungsformen der Repräsentation.

Diese werden nun zum Zeichnen der Repräsentation verwendet und dem Raphaël-Objekt zur Generierung von SVG-Elementen übergeben. Als Resultat wird von Raphaël eine Liste der generierten SVG-Elemente in Form von Raphaël-Element-Objekten zurückgegeben. Diese Liste wird in dem Repräsentations-Objekt als `graphics` abgespeichert, da der spätere Zugriff gewährleistet werden muss. Die Raphaël-Element-Objekte repräsentieren die Attribute einer Entität.

Um dies Knockout mitzuteilen, müssen zusätzliche *Data Bindings* zwischen den SVG-Elementen und der Viewmodel-Komponente erstellt werden. Da die Knockout-Bindings nur über den DOM-Zugriff angemeldet werden können, müssen die Raphaël-Elemente über deren `node`-Eigenschaft bekannt gemacht werden. In jedem Raphaël-Element werden Informationen mitgespeichert, die den Namen des Entitäten-Attributes beinhalten und die Po-

Programm 5.5: Funktion zum Auslesen der Tabelle.

```
1 ...
2 ViewModel.prototype.parseTable = function(table){
3
4   this.cache = {};
5
6   for(var r = 0 , rr = table.rows.length; r < rr; ++r){
7     for(var c = 0 , cc = table.rows[r].cells.length; c < cc; ++c){
8
9       var propertyName;
10
11      if((r == 0)){ // head data
12        var content = table.rows[r].cells[c].textContent.trim();
13        propertyName = this.cache[c] =
14                       this.cache[c] ||
15                       this.stringConverter(content);
16
17        (c == 0) ? this.entityLabel = propertyName
18                 : this.filter.push({  value : propertyName,
19                                       origin : content        });
20      }
21      else{
22        propertyName = this.cache[c] ||
23                       this.stringConverter(
24                         table.rows[r].cells[c].textContent );
25
26        var propertyValue;
27        if((c == 0)){
28          this.entities.push(new Entity((r - 1));
29
30          var label = table.rows[r].cells[c].textContent;
31
32          propertyValue = (label.length <= 15) ? label
33                            : label.substring(0, 16) + "...";
34
35          this.entities[r - 1].label = { property : label,
36                                         value: propertyValue};
37        }
38        else{
39          propertyValue = parseFloat(
40                            table.rows[r].cells[c].textContent);
41          this.entities[r - 1].entityAttributes.push(
42                    { property : propertyName,
43                         value : ko.observable(propertyValue) });
44        }
45      }
46    }
47  }
48 };
49 ...
```

sition des Elementes innerhalb der Zeichenfläche. Auf die Position wird zurückgegriffen, wenn sich die Höhe der Repräsentation ändert, was den Vorteil hat, dass die Berechnungen nur einmal durchgeführt werden.

Eventanmeldung

Nachdem sämtliche Objekte erstellt wurden und das HTML-Markup dynamisch mit Attributen und SVG-Elementen erweitert wurde, sind die Elemente bereit Benutzereingaben aufzunehmen. Für diesen Zweck wird die Bibliothek Hammer verwendet, da sie die üblichen Touch-Gesten implementiert. Der Aufbau der Bibliothek macht es erforderlich, für jedes interaktive Element ein neues Hammer Objekt zu instanziieren.

Mit diesem Ansatz müsste z. B. bei dem Balkendiagramm Widget, für eine Repräsentation eine Instanz für den Button, eine für das SVG-Element und jeweils eine Instanz für jedes Attribut erzeugt werden. Ein kleiner Tabellenkörper mit zwei Reihen und drei Spalten würde bei dieser Verwendung bereits acht Hammer Instanzen benötigen. Der Bedarf nach mehr Instanzen würde mit der Anzahl an Repräsentationen rapide ansteigen, und es könnte sein, dass viele dieser Instanzen nicht gebraucht werden, da mit dem Element nicht interagiert wird.

Der Ansatz, der in den Responsive Web Widgets zum Einsatz kommt, instanziiert die Hammer Bibliothek nur einmal. Das `<body>`-Element des HTML-Dokumentes wird für das Erzeugen der Hammer Instanz verwendet. Dies hat den Vorteil, dass der Speicherverbrauch für die *Eventlistener* konstant bleibt. Jedoch sind innerhalb der *Eventhandler* Unterscheidungen der einkommenden Events zu treffen.

Diese Unterscheidungen können anhand des Event-Typs getroffen werden. Events, die von Hammer erzeugt werden, liefern außerdem Referenzen zu dem `currentTarget` (dem `<body>`-Element), sowie dem `target` (Element das die Interaktion entgegen nimmt). Mit diesen Informationen kann der Index der aktivierten Repräsentation extrahiert werden und der Viewmodel-Komponente übergeben werden. Im Eventhandler wird die Funktion, die die Benutzereingaben bearbeiten soll, aufgerufen. Durch den übergebenen Index findet die Viewmodel-Komponente die dazugehörige Entität und kann diese auf den neuesten Stand bringen.Die Knockout Bibliothek kümmert sich dann um die Aktualisierung der View-Komponente.

Zusätzlich zu den Interaktionselementen, die sich mit dem Wert der Entität ändern, ist eine unsichtbare Interaktionsfläche eingezogen worden. Sollten die vorgesehenen Interaktionsflächen zu klein werden, registrieren die unsichtbaren Interaktionsflächen die Fingergesten und versuchen sie dem Interaktionselement, welches in unmittelbarer Nähe ist, zuzuordnen. Dies soll die Benutzerführung vereinfachen und einen reibungslosen Ablauf des Bedienungskomforts ermöglichen. Für jede Repräsentation steht eine unsichtbare Interaktionsfläche zur Verfügung, welche sich über jedes der dargestellten

Entitäts-Attribute spannt. Mit dieser Methode ist es möglich, das nächstgelegene Attribut anzusteuern.

Nachdem Änderungen am Datenbestand durchgeführt wurden, können diese Änderungen verworfen oder gespeichert werden. Je nach zugrunde liegender Interaktionsmethode wird ein Bestätigungs-Event abgesendet oder ein Abbrechen-Event, welches bekannt gibt, dass die Änderungen zurückgesetzt wurden. Damit ist es möglich auf die Events zu warten und in Callback-Funktionen Verbindungen zu einem Server aufzubauen und Datenbestände zu aktualisieren.

Kapitel 6

Evaluierung

In diesem Kapitel ist die kritische Betrachtung des Responsive Web Widgets festgehalten. Funktionale sowie nichtfunktionale Eigenschaften der Implementierung werden mit den Anforderungen aus Kap. 4 verglichen. Danach werden Lösungen aus Kap. 2 dem umgesetzten Ansatz gegenübergestellt. Das Responsive Web Widget wurde Personen vorgelegt, die die Benutzerschnittstelle testeten. Die Ergebnisse des praktischen Tests mit Verbesserungsvorschlägen und Zustimmungen zu verwendeten Konzepten sind am Ende des Kapitels aufgelistet.

6.1 Anforderungsreflexion

Dieser Abschnitt beschreibt, in welchem Ausmaß die Anforderungen an das Responsive Web Widget erfüllt werden. Zu Beginn werden die Funktionalen Anforderungen diskutiert. Hierbei werden zwischen den explizierten und implizierten Anforderungen aus Kap. 4 unterschieden.

Danach werden nichtfunktionale Anforderungen behandelt. Diese legen die Eigenschaften des Responsive Web Widgets fest. Der Grad der Erfüllung von funktionalen als auch nichtfunktionalen Anforderungen ist maßgebend für die Bewertung der Arbeit.

6.1.1 Funktionale Anforderungen

Als funktionale Anforderungen werden jene Anforderungen des Responsive Web Widgets verstanden, die von einem Benutzer erwartet werden. Die explizierten Anforderungen sind in Abschnitt 4.3 beschrieben. Dieser Abschnitt behandelt vorwiegend das Umwandeln einer HTML-Tabelle von einem passiven Element in ein aktives Element. Die implizierten Anforderungen sind in Abschnitt 4.2 erklärt. Zu diesen Funktionen gehören: das Berechnen von Werten, das Sortieren oder das Filtern von Attributen oder die flexible Anpassung des Layouts.

Aktives Tabellenelement

Diese explizierte Anforderung stellt den Anspruch an das Responsive Web Widget, eine HTML-Tabelle von einem passiven Element in ein aktives Element umzuwandeln. HTML-Tabellen sind im HTML-Standard als Container von Daten festgelegt. Für die beinhalteten Daten ist keine Editierungsmöglichkeit vorgesehen. Das Responsive Web Widget liest die Daten ein und macht sie dem Benutzer zugänglich.

Zum einen ändert das Responsive Web Widget die Datenaufbereitung, sodass es auf kleinen Bildschirmen zu einer übersichtlicheren Darstellung kommt. Als komprimierte Darstellungsform der Daten werden Diagramme verwendet.

Zum anderen werden Interaktionsmethoden hinzugefügt, welche das Erkunden des Datenaufkommens und das Bearbeiten von nummerischen Daten ermöglichen. Die HTML-Tabelle wird zu einem aktiven Element, indem sie sich sofort an die Bildschirmgegebenheiten anpasst. Die Tabelle ist außerdem ein aktives Element, da sie Benutzereingaben verarbeiten kann.

Flexibles Layout

Das Layout der Responsive Web Widgets kann sich an mehrere Bildschirmgrößen anpassen. Dabei werden Media Queries verwendet, welche die Fenstergröße des Betrachtungsgerätes auslesen können. Sowohl im Hochformat als auch im Querformat sind die Inhalte auf modernen Smartphones in ausreichender Größe dargestellt und gut lesbar. Auf Desktop-Computern werden die Tabellen in ihrer Ursprungsform angezeigt, wenn die vorhandene Fenstergröße ausreichend dimensioniert ist. Hochauflösende Tablet-PCs können im Querformat ebenfalls, die Ursprungstabelle anzeigen. Im Hochformat-Modus wechselt das Layout von der Tabellenanzeige in die, für Touch-Gesten optimierte, grafische Anzeige.

Das flexible Layout kann sowohl im Vollbildmodus als auch mit Statusanzeigen und Adressleisten, welche standardmäßig in den Webbrowsern verwendet werden, betrieben werden. Die unterschiedlichen Höhenabmessungen werden durch das Verwenden von relativen Größenangaben in den Layoutdefinitionen kompensiert.

Eindimensionales Scrollen

Die Responsive Web Widgets sind in ihrem Aufbau nach dem Konzept arrangiert, dass auf kleinen Bildschirmen nur eine Scroll-Richtung genutzt wird. Dies soll sicherstellen, dass Benutzer Zusammenhänge von Betrachtungsobjekten nicht aus den Augen verlieren, weil sie keine zweidimensionalen Positionswechsel zu bewältigen haben.

Die Einschränkung der Scroll-Richtung trägt dazu bei, dass Repräsentationen von Entitäten schneller gefunden werden können. Der Benutzer des

Responsive Web Widgets kann sich so besser orientieren, da nur die Position einer Richtung der betrachteten Repräsentationen eingeprägt werden muss.

SVG-Darstellung

Der Aufbau des Responsive Web Widgets unter der Verwendung der SVG-Technologie als Grafikformat hat sich bewährt. Durch das Vektorformat können sich die Grafiken an den vorhandenen Platz anpassen, ohne an Darstellungsqualität einzubüßen. Die Grafiken passen sich den Gegebenheiten ihres Eltern-Elementes automatisch an. Die Möglichkeit relative Längenmaße in SVGs zu verwenden ist unentbehrlich für die Entwicklung unterschiedlicher Bildschirmgrößen, da viele Berechnungen intern vom Webbrowser übernommen werden.

Für SVG existiert bereits ein gültiger Standard des W3C, das Canvas-Element wird erst mit der Veröffentlichung der HTML5-Recommendation zum Standard erhoben. Die grafischen Repräsentationen können mit der Knockout Bibliothek verbunden werden. Die Koppelung von Knockout und einem Canvas Element wäre in dieser Form nicht möglich, da Knockout Zugriff auf DOM-Elemente haben muss.

Das Interagieren mit den Grafiken auf einem Touchscreen ist ebenfalls gut durchführbar. Die Touch-Events werden im Fall der Responsive Web Widgets zuverlässig abgesendet. Bei Applikationen mit schnelleren Interaktionsabläufen können die Statusänderungen der SVGs zu langsam sein. Die Konzeption der Responsive Web Widgets vermeidet, dass schnellere Bedienungsabläufe, wie sie z. B. in Spielen vorkommen, nötig werden.

Wenn die SVG-Grafiken sehr umfangreich ausfallen, kann dies den Ablauf von Applikationen verlangsamen, da für jede Grafik ein Element erzeugt wird. Dies ist bei verdeckten Elementen problematisch, da sie nicht sichtbar sind, aber trotzdem Speicherplatz verbrauchen. Bei pixelbasierenden Techniken würden die verdeckten Bereiche von Elementen mit den sichtbaren Elementen überschrieben werden und der Speicherbedarf würde annähernd gleich bleiben. Animationen sind mit SVG möglich, sind aber auf mobilen Geräten sehr langsam und werden nicht immer flüssig abgespielt.

Filtern und Sortieren

Der Aufbau der Responsive Web Widgets impliziert, dass die Daten aus der Tabelle bereits gefiltert oder sortiert sind. Diese Funktionen sind ausschlaggebend, damit die Daten aus den HTML-Tabellen auf kleinen Bildschirmen platzsparender dargestellt werden können.

Das Kreisdiagramm Widgets filtert nach dem Typ der Attribute und kann nur dieses Attribut der jeweiligen Entitäten auf dem Bildschirm anzeigen. Das Balkendiagramm Widget zeigt die gesamte Anzahl an Attributen der Entitäten, und sortiert diese nach Größe. Dem Benutzer ist im Bal-

kendiagramm Widget möglich, zusätzlich nach dem Typ des Attributes zu filtern.

Eine Sortierung der Attribute erfolgt im Balkendiagramm Widget automatisch nachdem die Entität in platzsparender Form angezeigt wird. Die Sortierung basiert auf den Werten der Attribute. Die Sortierung nach Entitäten kann in beiden Responsive Web Widgets nicht durchgeführt werden.

Ein zusammengesetztes Objekt, wie es die Entitäten sind, kann nicht durch automatisch festgelegte Kriterien sortiert werden, dies ist für einen Benutzer zu unübersichtlich. In der Benutzerbefragung ist der Wunsch aufgetreten, Entitäten nach eigener Ermessung zu sortieren.

Statistikfunktionen

Das Balkendiagramm Widget bietet die Möglichkeit, die gesamten Werte der Attribute der Entitäten anzuzeigen. Dazu wird eine Anzeige eingeblendet, die die Werte aufschlüsselt. Ist ein Attribut ausgewählt, können die Statistiken zu diesem Typ über die Anzeige eingeblendet werden. Die Anzeige beinhaltet die Werte der Statistikfunktionen Maximum und Minimum.

Als Mittel wird der Median berechnet. Handelt es sich um eine gerade Anzahl von Entitäten, werden ein Ober- und Untermedian berechnet.

6.1.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen betreffen vorwiegend Anforderungen, die sich auf die Benutzerfreundlichkeit einer Applikation beziehen. Im Fall des Responsive Web Widgets sind es jene Anforderungen, die stark mit der Benutzerschnittstelle und der Handhabung verknüpft sind. Dazu zählen die verschiedenen Eingabemethoden und die grafischen Aktionsauslöser.

MVVM-Paradigma in der Praxis

Die Anwendung des MVVM-Paradigmas in den Responsive Web Widgets hat zu einer objektorientierten Programmierweise im JavaScript-Code geführt. Der Code wirde in kleine Teilaufgaben zerlegt, welche in Funktionen abgelegt sind. In der Praxis sind in manchen Fällen die Grenzen der Aufgabengebiete zwischen Model, View und Viewmodel undeutlich abgesteckt. In dem Responsive Web Widget Code ist die Viewmodel-Komponente für sämtliche Zugriffe auf die Model-Komponente verantwortlich. Es wäre durchaus möglich Methoden für Abfragen von Model-Eigenschaften in der Model-Komponente bereit zustellen. Diese Abfragen werden im Responsive Web Widget von der Viewmodel-Komponente getätigt.

Als weiteres Beispiel dienen die Aktivitätsmarkierungen der View-Komponente im Balkendiagramm Widget. Ob eine Repräsentation aktiv oder inaktiv ist, wird in der Repräsentation selbst gespeichert. Die Model-Komponente kommt für diese Aufgabe nicht infrage, da diese nur für die Tabellen-

daten verantwortlich ist. Die Viewmodel-Komponente ist für die Steuerung der Repräsentationen verantwortlich und müsste deswegen eine Datenstruktur bereitstellen, die den Status einer Repräsentation verwaltet.

Diese Datenstruktur müsste zusätzlich verwaltet werden und regelmäßig über den Zustand der View-Komponente informiert werden, was einen zusätzlichen Mehraufwand bedeutet. Das Ablegen des Aktiv/Inaktiv-Zustands im Repräsentations-Objekt hat den Vorteil abgekapselt zu sein und ohne zusätzliche Datenstruktur über die Viewmodel-Komponente erreichbar zu sein.

Das Verwenden der Knockout-Bibliothek im Speziellen hat für den prototypischen Entwurf einen Mehrwert gebracht, da es schnell zu stabilen Ergebnissen führt. In der Implementierung der Responsive Web Widgets wurde auf viele zu observierende Variablen verzichtet, um überschüssige Berechnungen zu vermeiden. Im Gegenzug dazu sind mehr Objekt-Koppelungen entstanden, als es unter der Verwendung von observierenden Variablen gekommen wäre. Mit der automatischen Erkennung von Abhängigkeiten von Knockout musste teilweise Code hinzugefügt werden um redundante Funktionsaufrufe, die von der Bibliothek ausgelöst wurden, zu unterbinden.

Knockout verwendet für die Kommunikation von JavaScript Funktionen und View-Elementen data*-Attributes. Diese sind HTML5 konform, jedoch noch nicht im SVG-Standard vorgesehen. Die Kommunikation zwischen SVG-Elementen und der Logik funktioniert einwandfrei, jedoch ist die Verwendung dieser Attribute nicht valide mit dem SVG-Standard.

Die nächste Version von Knockout soll das Erstellen von Data Bindings anstatt mit Markup-Attributen mit einer reinen JavaScript-Lösung unterstützen. In Kombination mit der Raphaël Bibliothek kann es zu Schwierigkeiten kommen, wenn die SVG-Nodes von Knockout verändert werden und nicht über die Zugriffsmethoden der Raphaël Bibliothek.

Da Raphaël den Zustand der SVG-Elemente beim Zeitpunkt des Erstellens in Raphaël-JavaScript-Objekten abspeichert, sollten alle Veränderungen der SVG-Elemente über die Zugriffsmethoden der Raphaël-Javascript-Objekte stattfinden. Dies ist notwendig damit die Zustände von dem Raphaël-JavaScript-Objekt und dem SVG-Element konsistent bleiben. Hat das SVG-Element einen anderen Zustand als das Raphaël-JavaScript-Objekt, werden SVG-Elemente durch das Data-Binding von Knockout verändert. Dies kann zu Problemen führen, wenn die Raphaël-Bibliothek für weitere Berechnungen, wie z. B. Animationen, herangezogen werden. Die Responsive Web Widgets kapseln alle relevanten Daten zur Visualisierung selbst, sie sind nicht auf die Raphaël-JavaScript-Objekte angewiesen. Die Raphaël Bibliothek wird nur zum Initialisieren der SVG-Elemente und zum Abspeichern von benutzerdefinierten Daten verwendet.

Touch Interface

Das Interface für mobile Geräte mit Touch-Eingabe ist so gestaltet, dass es den Anforderungen der Interface-Guidelines der Hersteller entspricht. Die Dimensionen der Interaktionsflächen sind so gewählt, dass sie selbst auf Geräten mit einer Fenstergröße von 320 px noch mit dem Finger bedienbar sind. Der Bedienungsablauf der Responsive Widgets ist so gewählt, dass es nicht nötig ist, Geräte mit Multi-Touch-Funktion zu verwenden. Jede Aktion ist mit einer Single-Touch Geste durchführbar.

Obwohl die Responsive Web Widgets insgesamt nur wenig Aktionen zum Betrachten und Editieren von Daten bieten und das Platzangebot auf Smartphones sehr begrenzt ist, sind diese Funktionen über mehrere Interaktionsflächen aufrufbar. Viele der Interaktionsflächen liegen in einem Bereich, der mit dem Daumen gut erreichbar ist. Das bedeutet, dass es möglich ist, mit einer Hand die Steuerung des Responsive Web Widgets zu übernehmen. Die Bibliothek Hammer zeigt auf allen Versuchsgeräten eine sehr gute Leistung. Die Erkennung der Touch-Gesten funktioniert bei allen Geräten auf Anhieb. Die Gesten werden ohne Verzögerung an die Viewmodel-Komponente des Responsive Web Widget weitergegeben und verarbeitet.

Aktionsaufrufe

In den Responsive Web Widgets sind die Touch-Gesten nach den Prinzipien von Abbschnitt 4.2.3 umgesetzt. Das Wechseln von dem Betrachtungsmodus in den Editiermodus erfolgt über eine Long press Geste. Die Long press Geste wird nach längerem Verweilen auf einer Position auf dem Bildschirm ausgeführt. Dies verhindert dass unbeabsichtigte Modiwechsel, während andere Touch-Gesten im Gange sind, stattfinden.

Außerdem werden *mouseover*-Events angemeldet welche es ermöglichen Filter- und Statistik-Interaktionsflächen für die aktuellen Repräsentationen einzublenden. *mouseover*-Events werden auf Geräten mit Touch-Eingabe ausgelöst, wenn diese angesteuert werden. Durch die Drag Geste auf aktiven Elementen werden die Werte der Entitäten verändert. Diese Eingabemethode kommt vor allem den Touch-Geräten entgegen, ist aber auch auf Desktop-Computern praktikabel.

Ist genügen Platz vorhanden um eine Tabelle in ihrer ursprünglichen Gestalt zu präsentieren, werden andere Eingabetechniken verwendet. Auf einem Desktop-Computer mit einer Maus als Eingabegerät kann durch einen Doppelklick auf eine Zelle mit nummerischem Inhalt in den Bearbeitungsmodus gewechselt werden. Der Wert der Zelle kann dann mit der Tastatur bestimmt werden und gespeichert werden. Bestätigt werden die neuen Werte mit einem Doppelklick.

Unobtrusive JavaScript

Das Responsive Web Widget ist so konzipiert, dass, unter der Verwendung von JavaScript, Interaktionsmöglichkeiten zur Betrachtung und Bearbeitung der Quelldaten gegeben sind. Ist JavaScript nicht vorhanden, können – bis auf die Standardaktionen des Webbrowsers – keine Aktionen durchgeführt werden, da die Interaktionsmethoden JavaScript voraussetzen.

Die grafischen Abbildungen des Responsive Web Widgets werden nicht angezeigt. Als Fallback des Responsive Web Widgets wird die Tabelle, in der die Daten vorliegen, dargestellt. Die Tabelle ist in ihrer Ursprungsform, als passives Element ohne Interaktionsmöglichkeiten vorhanden.

6.2 Benutzerstudie

Die Testpersonen gehören der Altersgruppe der Personen von 20–30 Jahren an und haben Erfahrungen mit Touch-Gesten und diversen nativen Applikationen auf Smartphones und Tablet-PCs sowie fundierte PC-Kenntnisse. Es handelt sich um drei Probanden, die im Vorfeld der Benutzerstudie nicht mit der Bedienung der Responsive Web Widgets vertraut wurden. Die Probanden erhielten Informationen über die Funktionen, die mit dem Responsive Web Widget möglich sind, jedoch nicht, wie diese Funktionen ausgelöst werden. Dies soll dem Zweck dienen, Erkenntnisse über die Gestaltung der Benutzerschnittstelle zu gewinnen.

Die Probanden wurden, angelehnt an den Think-Aloud-Test, während der Benutzerstudie beobachtet, und waren aufgefordert, ihre Intentionen bei der Bedienung des Responsive Web Widgets laut auszusprechen. Die Äußerungen wurden durch Mitschrift dokumentiert und dienen als Rekonstruktionshilfe der Bedienungsabläufe. Damit ist nachvollziehbar was sich ein Benutzer bei der Interaktion mit dem Responsive Web Widget erwartet. Der Fokus der Benutzerstudie liegt auf dem Bedienungskonzept der Responsive Web Widgets und ob die Interaktionsflächen so gestaltet sind, dass eine intuitive Steuerung der Widgets möglich ist.

6.2.1 Testgeräte

Als Testgeräte dienten die privat genutzten Geräte mit berührungssensitiven Bildschirmen der Testteilnehmer, mitsamt deren benutzerdefinierten Einstellungen und Präferenzen. Die Tabelle 6.1 zeigt ein Aufschlüsselung der Verwendeten Modelle und ihrer Software. Eine interessante Erkenntnis zu Beginn der Befragung war, dass die Geräte ein unterschiedliches Verhalten in der Darstellung der Responsive Web Widgets zeigten, obwohl alle Geräte von demselben Hersteller stammen. Da der Standardwebbrowser des Samsung Galaxy Ace 2 keine SVG-Anzeige unterstützt, wurde das Responsive Web Widget nicht angezeigt. Für diese Fälle sollte als Fallback die

Tabelle 6.1: Überblick der Testgeräte mit verwendetem Webbrowser und Versionen der Betriebssystem.

Modell	*Betriebssystem*	*Webbrowser*
Samsung Galaxy Tab 2 10.1	Android 4.0.3	Google Chrome
Samsung Nexus S	Android 4.1.2	Google Chrome Modzilla Firefox
Samsung Nexus S	Android 4.2.2	Standard Browser
Samsung Galaxy S 2	Android 4.1.2	Google Chrome
Samsung Galaxy Ace 2	Android 2.3.6	Standard Browser (nicht ausführbar)

HTML-Quelltabelle angezeigt werden, wie es für Geräte ohne JavaScript Unterstützung implementiert ist. In den folgenden Beschreibungen der Evaluierung wird das Galaxy Ace 2 aufgrund dieser Inkompatibilität nicht mehr berücksichtigt.

6.2.2 Ergebnisse der Benutzerstudie

In der folgenden Liste werden Kritikpunkte und Zusprüche am Benutzerschnittstellen-Design aufgelistet. Der Gesamteindruck ist durchweg positiv, trotzdem haben die Testpersonen Vorschläge, basierend auf ihrer subjektiven Einschätzung, zur einfacheren und schnelleren Bedienung der Responsive Web Widgets eingebracht.

Ladezeit Alle Befragten sind zufrieden mit der Ladezeit und dem Auslösen des Responsive Web Widgets, bei den Beteiligten hatten zwei Personen Internetzugang über einen WLAN-Router und eine Person Zugang über mobiles Internet.

Touch-Input Die Probanden empfinden die Annahme der Touch-Gesten als sehr gut. Das Responsive Web Widget reagiert schnell auf die Eingaben und die Erkennung der Gesten erfolgt reibungslos.

Die Probanden sind der Ansicht, dass die Dimensionen der Interaktionsflächen und der Schriftgröße ausreichend hoch sind.

Alle drei Befragten sind sehr zufrieden mit der Selektion von visuell kleinen Elementen. Ist der Wert eines Attributes niedrig, ist auch die Repräsentation des Attributes schmal. Unter Verwendung einer unsichtbaren Interaktionsfläche wird die Touch-Geste an das nächstgelegene Element weiter gegeben.

Jeder der Probanden ist mit der Sensibilität der Drag Geste zum Einstellen der Attribut-Werte sehr zufrieden. Es ist selbst auf den Smart-

phones leicht möglich, Werte um nur eine Einheit zu verschieben. Außerdem wird die Drag Geste nicht nur auf dem dafür vorgesehenen Angreifer erkannt, sondern auf der gesamten Repräsentation. Damit ist eine einfache Bedienung möglich.

Ein Proband würde mit der Double touch Geste auf einen Balken in den Filtermodus wechseln wollen.

Entitäten Repräsentationen Alle Probanden empfinden das Betrachten und Bearbeiten von Tabellen mit dem Responsive Web Widget als Mehrwert und eindeutige Verbesserung zum Standardverhalten der Webbrowser.

Alle Befragten wünschen sich bei der angezeigten Tabelle auf großen Bildschirmen, denselben Funktionsumfang wie auf den mobilen Geräten mit kleinen Bildschirmen.

Einer der Befragten wünscht sich bei dem Button, der die Entitäten bezeichnet, dass dieser die Anzeige mit allen Attribut-Werten auslöst.

Ist ein Attribut selektiert, werden der Name und der Wert des Attributes über dem Balken angezeigt. Wenn das Attribut deselektiert wird, verschwindet die Anzeige. Zwei Probanden bemängeln, dass bei aktivem Filter das angezeigte Attribut in jeder Entität selektiert sein soll und die Werte automatisch eingeblendet werden sollen, da bereits nach Interesse gefiltert wurde.

Zwei Probanden wollen das im aktiven Filter-Modus nur die Statistikkalkulationen über den Stats-Button aufrufbar sind und das Aufrufen des Gesamtüberblicks der Attribute unterbunden wird.

Einer der Befragten empfindet es als hilfreich, dass ein selektiertes Attribut deselektiert wird, nach dem die Entität in ihren platzsparenden Zustand übergeht und eingeklappt wird.

Einer der Befragten äußerte sich positiv zu dem versetzten Erscheinungsbild der Balken im eingeklappten Zustand der Entität.

Ein Proband würde es bevorzugen, dass die Tabelle in den Farben der Attribut-Repräsentation eingefärbt ist, wenn sie auf größeren Bildschirmen angezeigt wird.

Editiermodus Alle Probanden würden es gut heißen, wenn der Editiermodus nicht nur für eine Entität aufrufbar ist, sondern für mehrere gleichzeitig.

Ein Proband ist der Meinung, dass beim Starten des Editiermodus ein Attribut vorausgewählt sein sollte.

Alle Probanden wünschen sich ein besseres Feedback, wenn sie den Editiermodus aktivieren. Das derzeitige Feedback kann zu leicht übersehen werden und in manchen Fällen ist es schwer ersichtlich, welche Entität sich gerade im Editiermodus befindet.

Ein Befragter hat Hilfe benötigt um den Editiermodus zu öffnen, da

die Long press Geste nicht intuitiv war.

Zwei der Befragten wünschen sich bei dem Button, der die Entitäten bezeichnet, dass dieser den Editiermodus beendet und die Änderungen speichert.

Filter Zwei Probanden würden ein Icon anstatt der Filter-Selectbox bevorzugen, da der Inhalt der Selectbox nicht immer lesbar ist.

Anzeige Wenn die Anzeige der Statistiken oder die Anzeige der Attributwerte einer Entität eingeblendet ist, drückte jeder der Probanden den Back-Hardwarekey des Android-Gerätes, da kein anderer Button in dem Widget ersichtlich war. Als Lösung wurde das Einbringen eines Schließen-Buttons in die Anzeige vorgeschlagen, der die Anzeige beendet.

Toolbar In der Toolbar befinden sich der Stats-Button und das Filter Drop Down. Ist ein Attribut einer Entität ausgewählt, löst ein Drücken des Stats-Buttons die Statistikkalkulation aus. Ist kein Attribut ausgewählt, werden sämtliche Attribute und deren Werte der Entität angezeigt. Alle Probanden sind der Meinung, dass der Stats-Button sein Aussehen je nach Selektion ändern soll.

Alle drei Befragten empfinden es als nützlich, wenn die Toolbar zu der ausgewählten Entität springt. Dies spart Scrollaufwand ein, und da sie mit dem Daumen erreichbar ist, ist sie einhändig bedienbar.

6.2.3 Verbesserungsmöglichkeiten

Speziell für den getesteten Prototyp fließen die Ergebnisse der Benutzerstudie in die Verbesserungsvorschläge ein. Für Texte, die auf kleinen Bildschirmen nicht vollständig lesbar sind, sollen Icons verwendet werden. Der größte Kritikpunkt war, dass zu wenig Feedback über die Aktionen der Responsive Web Widgets gegeben wird. Vor allem das Erscheinungsbild des Editiermodus muss sich stärker ändern, wenn der Zustand gewechselt wird.

Das Sortieren und Filtern von Attributen der Entitäten wird bereits ermöglicht, doch in manchen Fällen ist es wünschenswert, dass die Sortierung nach anderen Kriterien als an- und absteigende Zahlenwerte erfolgen könnte. In weiterer Folge sollte es ermöglicht werden, dass die Reihenfolge der Entitäten verändert werden könnte. Dies ist nützlich, damit der Benutzer die Entitäten nach persönlichen Präferenzen ordnen kann.

Die Darstellung der Responsive Web Widgets auf kleinen Bildschirmen wurde mit Funktionen erweitert, um durch Sortier- und Filtermechanismen platzsparend gestaltet zu werden. Diese Zusatzfunktionen werden von den Benutzern auch erwartet, wenn die Tabelle in vollem Umfang auf großen Bildschirmen dargestellt wird.

6.2.4 Erweiterungsmöglichkeiten

In Abschnitt 4.4.3 sind einige Punkte aufgelistet, welche die Limitierungen der Funktionalitäten des Responsive Web Widgets aufzeigen. Eine wesentliche Verbesserung wäre, wenn die Responsive Web Widgets mit jeder Art von Daten umgehen könnten. Im Moment sind nur nummerische positive Werte zur Aufbereitung durch die Responsive Web Widgets erlaubt. Für mehrere Datentypen müssten viele Widget Arten implementiert werden, die mit der Datenfülle umgehen können.

Ein weiterer Denkanstoß in diese Richtung sind Fälle wie: verbundene Zellen, verschachtelte Tabellen, Tabellen die Literale gemischt mit anderen HTML-Elementen beinhalten usw. Diese Liste an Tabellenvariationen kann sehr lange fortgeführt werden, deswegen wird immer ein Verbesserungspotenzial oder ein Responsive Web Widget Bedarf vorhanden sein. Mit der Verwendung von SVGs kann eine barrierefreie Gestaltung erzielt werden, da die SVG-Elemente mit WAI-ARIA Auszeichnungen bestückt werden können. Speziell für Menschen mit Sehschwächen können die Grafiken nicht nur in unterschiedlichen Farben dargestellt werden, sondern auch in verschiedenen Füllmustern.

Für eine verbesserte Verwendung der Responsive Web Widgets in Projekten kann die Aufbereitung des Codes in eine Bibliotheks-Architektur umgesetzt werden. Durch die Bibliotheks-Architektur ist der Code besser wiederzuverwenden, und kann benutzerdefinierter gestaltet werden.

6.3 Fazit

Responsive Web Widgets sind entstanden, um Datentabellen für Benutzer zugänglicher zu machen. Dabei werden Diagramme als Darstellungsformen verwendet, um die starre Struktur der Tabellen in grafische Elemente umzuwandeln. Die grafischen Elemente können auf kleinen Bildschirmen als Interaktionsflächen dienen. Die Umsetzung der Responsive Web Widgets entspricht den technischen Anforderungen und Erwartungen. Responsive Web Widgets verknüpfen Eingabemethoden von Desktop-Computern mit Gesten-Erkennung auf mobilen Geräten und machen sich Eigenheiten der Gerätearten zu nutzen.

Die verwendete Gesten-Erkennung reagiert sehr gut und löst die richtigen Touch-Events aus. Die Berührungssensitivität kommt auf Android-Geräten nativen Applikationen sehr nahe. Zusätzlich ist das auf eine Richtung limitierte Scrollen auf kleinen Bildschirmen ein wesentlicher Bestandteil der Konzeption um die Übersichtlichkeit von Tabellen von großen Bildschirmen auf kleine Bildschirme zu transferieren. Die Verwendung von SVGs bewährt sich in diesem Fall, da die Vektorgrafiken auflösungsunabhängig ohne Qualitätsverlust angezeigt werden können.

Wie viel Zeit das Erstellen der Responsive Web Widgets auf der Client-

Seite bei großen Datenmengen in Anspruch nimmt, muss zukünftig getestet werden. Für eine HTML-Tabelle mit zehn Spalten und zehn Zeilen ist der Ladevorgang auf mobilen Geräten nicht von langer Dauer und zu vernachlässigen.

Das Responsive Web Widget wurde durch die Benutzerstudie einem kritischen Benutzerschnittstellen-Test unterzogen, der einige Vor- und Nachteile der prototypischen Implementierung aufgeworfen hat. Die Probanden haben sich gut in der Benutzerschnittstelle des Responsive Web Widgets zurechtgefunden. Aufgrund der Tatsache, dass die Probanden nicht sehr viel Vorinformation erhalten haben und bereits Erfahrungen mit Touch-Gesten gesammelt haben, kann davon ausgegangen werden, dass das Responsive Web Widget die erwarteten Aktionen auslöst.

Kapitel 7

Schlussbemerkungen

Unterschiedliche Geräte mit Internetzugang werden in verschiedenen Alltagssituationen benutzt. Die Entwicklung nativer Applikationen ist sehr zeitintensiv, da für die verschiedenen Betriebssysteme unterschiedlicher Hersteller mehrere Applikationen geschrieben werden müssen. Die Applikationen bieten denselben Funktionsumfang, was zu einer redundanten Applikationsentwicklung führt. Dazu kommt, dass die Leistung von Tablets und Smartphones steigt und viele neue Produkte mit neuen Features auf den Markt drängen. All diese unterschiedlichen Gerätetypen, ältere und neuere Modelle müssen ebenfalls durch native Applikationen bedient werden.

Webapplikationen laufen in einem Webbrowser. Sie haben nicht den umfangreichen Hardwarezugriff, wie native Applikationen und sind deswegen in ihrem Funktionsumfang etwas eingeschränkt. Gängige Web-Techniken unterliegen Standards, die vom W3C kontrolliert werden. Dies bedeutet, dass die Entwicklung einer Webapplikation herstellerübergreifend durchführbar ist, solange die verwendeten Webbrowser die Standards implementieren. Auf älteren Geräten ist ältere Webbrowsersoftware zu finden, welche nicht die neuesten Webstandards interpretieren können. Diese Defizite müssen von Webapplikations-Entwicklern beachtet werden und mit geeigneten Fallbacks behandelt werden. Die Problematik des unterschiedlichen Verhaltens von Webbrowsern bei der Anzeige von Webseiten ist weit bekannt. Das Herstellen eines konsistenten Verhaltens der Webseite in verschiedenen Webbrowserversionen ist eine der Hauptaufgaben in der Webprogrammierung.

Nicht nur die Anzeige von Webseiten ist auf den unterschiedlichen Webbrowsern problematisch, sondern auch die Interaktionsverarbeitung, da die Webbrowser verschiedene JavaScript-Engines verwenden. Hinzukommt, dass die Bedienungsmethoden von berührungssensitiven Geräten und Desktop-Computern, ebenfalls beachtet werden müssen. Anwender erwarten bei der Verwendung derselben Webapplikationen auf Desktop-Computern und auf mobilen Geräten denselben Funktionsumfang.

In dieser Arbeit wird der Ablauf zur Erstellung eines Responsive Web

Widgets beschrieben. Widgets sind Elemente einer Webseite, die komplexe Aufgaben – vor allem das Verarbeiten von Benutzereingaben – erfüllen. Die Arbeit handelt von dem Abbilden von Darstellungs- und Interaktionsmethoden auf verschiedenen Geräten. Hauptaugenmerk liegt bei der Darstellung von Tabellen auf kleinen Bildschirmen, da das Standardverhalten der Webbrowser auf mobilen Geräten in diesem Bereich unzureichend ist. Am Anfang dieser Arbeit werden einige Ansätze zur Behebung dieses Problems genannt. Einige ältere Lösungsvorschläge, die vor der Verbreitung von Smartphones entstanden sind, haben weiterhin noch Gültigkeit. Andere Ansätze versuchen mit neuen CSS-Definitionen, wie Flexible Boxes, Tabellen lesbarer zu gestalten. Zum Teil werden Filterfunktionen bereitgestellt, damit die Größe der Tabelle kleiner wird und besser auf kleinen Bildschirmen Platz findet. Ein weiterer interessanter Lösungsvorschlag ist die Aufbereitung der Daten in Diagrammformen, damit durch die Abstraktion der Daten eine platzsparende Version der Tabelle generiert werden kann. Eine andere Möglichkeit ist ein Webbrowser-Plug-in zu installieren, das sich um die Aufbereitung der Tabellendaten kümmert.

Danach werden Webtechniken vorgestellt, die im Responsive Web Design Anwendung finden. Es werden SVG-Grafiken mit dem Canvas-Element verglichen und Vor- und Nachteile dieser Technologien erläutert. Außerdem werden Metainformationen aufgelistet, die die Konfiguration des Viewports auf mobilen Geräten ermöglichen. Media Queries sind für die Anpassung von Webseitenlayouts auf unterschiedlichen Bildschirmgrößen verantwortlich. Sie werden dazu benutzt, entsprechend den Gegebenheiten des Betrachtungsgerätes, CSS-Definitionen anzuwenden.

Basierend auf den Erkenntnissen der untersuchten Webtechnologien werden Anforderungen ausgearbeitet und Konzepte zur Realisierung der Responsive Web Widgets erstellt. Es werden die unterschiedlichen Verhaltensweisen von Touch-Gesten und Maus-Eingaben beschrieben. Zudem sind Betrachtungsweisen angeführt, die die Problematik von Tabellendarstellung und die Bearbeitung der beinhaltenden Daten beleuchten.

Die Umsetzung dieser Anforderungen erfolgt mittels JavaScript, CSS, SVG und HTML unter Verwendung von JavaScript Bibliotheken. Für die Code-Strukturierung wird das MVVM-Paradigma durch Knockout JS angewendet. Hammer JS ist für die Erkennung der Eingabemethoden verantwortlich und mit Raphaël JS werden die SVG-Abbildungen erzeugt.

Der entstandene Prototyp wird durch eine Benutzerstudie evaluiert. Es wird der Think-Aloud-Test angewendet. Dabei sind die Probanden angehalten ihre Erwartungen und Intentionen, während der Bedienung des Prototyps laut kundzutun. Die Äußerungen der Probanden werden schriftlich festgehalten, um den Bedienungsablauf wiederherstellen zu können.

Die Fülle an Geräten, die für die Nutzung des Internets verwendet werden, nimmt stetig zu. Neben Desktop-Computern, Tablet-PCs und Smartphones, werden Smart-TVs und Spielkonsolen für den Internetzugang ver-

wendet. Die Geräte unterstützen verschiedene Eingabeformen, die mit den Benutzerschnittstellen der konventionellen Desktop-Entwicklungen unzureichend bedienbar sind. Der Benutzerkomfort soll bei der Betrachtung und Interaktion von Applikation unabhängig von dem verwendeten Gerät bestehen bleiben. Der Entwicklungsaufwand von nativen Applikationen, die auf all jenen Geräten laufen sollen, ist enorm. Zu schnell ändern sich gerätespezifische APIs oder Hardwarekonfigurationen.

Die Webentwicklung bietet eine sinnvolle Alternative. Durch die Standardisierung vom W3C-Konsortium bieten Technologien in Web Projekten eine gemeinsame Schnittmenge, die auf internetfähigen Geräten anwendbar sind. Die Verwendbarkeit der Web Widgets hängt von den Standard-Implementierungen der benutzen Webbrowser ab. Für ältere Geräte müssen Fallbacks bereitgestellt werden. Trotzdem wird die Verbreitung von Webanwendungen und Web Widgets aufgrund ihrer Plattformunabhängigkeit weiterhin Einzug in der Applikationsentwicklung halten.

Anhang A

Inhalt der CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 Masterarbeit

Pfad: /

Schwoiger_Marvin_2013.pdf Masterarbeit (Gesamtdokument)

A.2 Online-Quellen

Pfad: /Literatur

*.pdf . . . . . Kopien der Literatur und Online-Quellen

Internet Access_Households and Individuals.xls Statistik des *Office for National Statistics*

Pfad: /Fussnoten

*.pdf . . . . . Ausgewählte Webseiten aus Fußnoten als Beispiel für Tabellendarstellungen

*.png . . . . . Screenshots von Webseiten zur Veranschaulichung der alternativen Darstellung von Webseiten auf kleinen Bildschirmgrößen

A.3 Abbildungen

Pfad: /Abbildungen

*.pdf . . . . . Vektorgrafiken

A.4 Projektdateien

Pfad: /Projekt

ResponsiveWebWidget.zip Prototyp Responsive Web Widget

Quellenverzeichnis

Literatur

[1] J. Allsopp. *Developing with Web Standards*. Voices That Matter. Pearson Education, 2009.

[2] Michael Derntl, Stephan Erdtmann und Ralf Klamma. „An embeddable dashboard for widget-based visual analytics on scientific communities“. In: *Proceedings of the 12th International Conference on Knowledge Management and Knowledge Technologies*. i-KNOW '12. ACM, 2012, 23:1–23:8.

[3] Jeffrey Heer und Ben Shneiderman. „Interactive Dynamics for Visual Analysis“. In: *Queue* 10.2 (Feb. 2012), 30:30–30:55.

[4] Ethan Marcotte. *Responsive Web Design*. A Book Apart, 2011.

[5] A. Osmani. *Learning JavaScript Design Patterns*. JavaScript and jQuery developer's guide. O'Reilly Media, Incorporated, 2012.

[6] Keishi Tajima und Kaori Ohnishi. „Browsing large HTML tables on small screens“. In: *Proceedings of the 21st annual ACM symposium on User interface software and technology*. UIST '08. ACM, 2008, S. 259–268.

[7] Carolyn Watters, Rui Zhang und Jack Duffy. „Comparing table views for small devices“. In: *Proceedings of the 2005 ACM symposium on Applied computing*. SAC '05. ACM, 2005, S. 975–980.

[8] L. Wroblewski. *Mobile First*. A book apart. A Book Apart, 2011.

[9] Wenchang Xu, Xin Yang und Yuanchun Shi. „A new mode of browsing web tables on small screens“. In: *Proceedings of UCS* (2009), S. 75–79.

[10] Jeffrey Zeldman und Ethan Marcotte. *Designing with Web Standards*. New Riders, Okt. 2009.

Online-Quellen

[11] Android Developers. *Gestures*. URL: http://developer.android.com/design/patterns/gestures.html (besucht am 05.05.2013).

[12] Android Developers. *Metrics and Grids*. URL: http://developer.android.com/design/style/metrics-grids.html (besucht am 07.05.2013).

[13] *CSS Tools: Reset CSS*. URL: http://meyerweb.com/eric/tools/css/reset/ (besucht am 03.06.2013).

[14] Marcos Caceres. *Widgets 1.0: The Widget Landscape (Q1 2008), W3C Working Draft 14 April 2008*. URL: http://www.w3.org/TR/2008/WD-widgets-land-20080414/#differences (besucht am 09.04.2013).

[15] *Can I use SVG?* URL: http://caniuse.com/svg (besucht am 08.04.2013).

[16] *Can I use the HTML5 Canvas element?* URL: http://caniuse.com/canvas (besucht am 05.04.2013).

[17] Dev.Opera. *Comparison of SVG and Canvas*. URL: http://dev.opera.com/articles/view/svg-or-canvas-choosing-between-the-two/#comparison-of-svg-and-canvas (besucht am 04.05.2013).

[18] Dev.Opera. *Love your devices: adaptive web design with media queries, viewport and more*. URL: http://dev.opera.com/articles/view/love-your-devices-adaptive-web-design-with-media-queries-viewport-and-more/ (besucht am 09.04.2013).

[19] Erwin Ebermann. *Die grafische Darstellung statistischer Ergebnisse*. URL: http://www.univie.ac.at/ksa/elearning/cp/quantitative/quantitative-114.html (besucht am 05.05.2013).

[20] *HTML 5.1 Nightly*. URL: http://www.w3.org/html/wg/drafts/html/master/embedded-content-0.html#the-canvas-element (besucht am 09.04.2013).

[21] HTML5 Rocks. *"Mobifying"Your HTML5 Site*. URL: http://www.html5rocks.com/en/mobile/mobifying/ (besucht am 04.05.2013).

[22] HTML5 Rocks. *Touch And Mouse*. URL: http://www.html5rocks.com/en/mobile/touchandmouse/ (besucht am 05.05.2013).

[23] Mozilla Developer Network. *CSS Reference Length Property*. URL: https://developer.mozilla.org/en-US/docs/CSS/length (besucht am 07.05.2013).

[24] Mozilla Developer Network. *CSS media queries*. URL: https://developer.mozilla.org/en-US/docs/CSS/Media_queries#Logical_operators (besucht am 06.05.2013).

[25] Mozilla Developer Network. *Tables*. URL: https://developer.mozilla.org/en-US/docs/HTML/Element/table#Examples (besucht am 09.04.2013).

[26] Mozilla Developer Network. *Touch events*. URL: https://developer.mozilla.org/en-US/docs/Web/Guide/DOM/Events/Touch_events (besucht am 07.05.2013).

[27] Mozilla Developer Network. *Using the viewport meta tag to control layout on mobile browsers.* URL: https://developer.mozilla.org/en-US/docs/Mobile/Viewport_meta_tag#Background (besucht am 06.05.2013).

[28] Office for National Statistics. *Internet Access – Households and Individuals, 2012 Part 2.* URL: http://www.ons.gov.uk/ons/publications/re-reference-tables.html?edition=tcm%3A77-289719 (besucht am 05.05.2013).

[29] Safari Developer Library. *Supported Meta Tags.* URL: http://developer.apple.com/library/safari/#documentation/appleapplications/reference/SafariHTMLRef/Articles/MetaTags.html (besucht am 09.05.2013).

[30] W3C Candidate Recommendation. *HTML5.* URL: http://www.w3.org/TR/html5/document-metadata.html#the-meta-element (besucht am 06.05.2013).

[31] W3C Candidate Recommendation. *Pointer Events.* URL: http://www.w3.org/TR/2013/CR-pointerevents-20130509/ (besucht am 05.05.2013).

[32] W3C Recommendation. *Media types.* URL: http://www.w3.org/TR/2011/REC-CSS2-20110607/media.html (besucht am 04.05.2013).

[33] W3C Recommendation. *Scalable Vector Graphics (SVG) 1.1 (Second Edition).* URL: http://www.w3.org/TR/SVG/struct.html#NewDocumentOverview (besucht am 09.04.2013).

[34] W3C Wiki. *The principles of unobtrusive JavaScript.* URL: http://www.w3.org/wiki/The_principles_of_unobtrusive_JavaScript (besucht am 07.05.2013).

[35] WHATWG. *HTML Living Standard.* URL: http://www.whatwg.org/specs/web-apps/current-work/#dom-canvas-getcontext (besucht am 06.05.2013).

[36] Geir Wavik. *Table Layouts vs. Div Layouts: From Hell to... Hell?* URL: http://coding.smashingmagazine.com/2009/04/08/from-table-hell-to-div-hell/ (besucht am 01.06.2013).

[37] Web Platform. *Touch Input Considerations.* URL: http://docs.webplatform.org/wiki/concepts/mobile_web/touch (besucht am 05.05.2013).

[38] Windows Phone Dev Center. *Interactions and usability with Windows Phone.* URL: http://msdn.microsoft.com/en-us/library/windowsphone/design/hh202889%28v=vs.105%29.aspx (besucht am 09.05.2013).

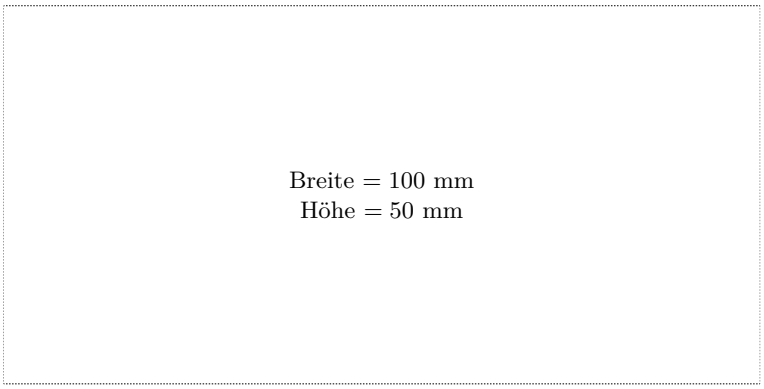
[39] iOS Developer Library. *Handling Events*. URL: http://developer.apple.com/library/ios/#documentation/AppleApplications/Reference/SafariWebContent/HandlingEvents/HandlingEvents.html (besucht am 07.05.2013).

[40] iOS Developer Library. *iOS UI Element Usage Guidelines*. URL: http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/UIElementGuidelines/UIElementGuidelines.html#//apple_ref/doc/uid/TP40006556-CH13-SW1 (besucht am 07.05.2013).

[41] msdn. *Implementing the MVVM Pattern*. URL: http://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx (besucht am 07.05.2013).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —