Ein Layoutsystem zur Erstellung von geräte- und auflösungsunabhängigen Webanwendungen

Christoph Zeller

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Jänner 2013

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 24. Januar 2013

Christoph Zeller

Inhaltsverzeichnis

Kurzfassung

Die Nutzung des Internets per *Smartphone* oder *Tablet* wird immer beliebter. Viele weitere Geräte des täglichen Lebens sind mittlerweile mit einer Schnittstelle zur Verbindung mit dem Internet ausgestattet. Spielkonsolen sowie eine Vielzahl moderner Fernseher laden den Benutzer förmlich ein, gemütlich von der Couch aus im Internet zu surfen. Diese Tatsache bringt mit sich, dass sich Webanwendungen nachhaltig verändern. Diese können einerseits in Form von sogenannten *Apps* für die jeweilige Plattform, oder andererseits optimiert für die Darstellung der jeweiligen Endgeräte umgesetzt werden. Mit dem Konzept *Responsive Webdesign* können effizient bildschirmauflösungs- und geräteunabhängige Webanwendungen realisiert werden.

Im Zuge dieser Arbeit wurde ein *CSS-Framework* entwickelt, das auf dem Konzept von *Responsive Webdesign* aufbaut. Es soll Webentwickler dabei unterstützen, schneller und effizienter zu einer fertigen Webanwendung zu gelangen. Das *Framework* stellt einen Grundstock an Funktionalität bereit. Es bietet einen flexiblen Gestaltungsraster, der es ermöglicht, die Webanwendung an verschiedene Bildschirmauflösungen automatisch anzupassen. Zusätzlich wurde auch großer Wert auf die Kompatibilität verschiedener Browser, gelegt. Mittels Modulen lässt sich das *Framework* für individuelle Anforderungen erweitern.

Abstract

Using the Internet on smart phones or tablet PCs is gaining in popularity. In addition, many other tools of everyday life already offer interfaces to establish an Internet connection. Video game consoles as well as various modern TV screens invite user to surf the Internet in a most convenient way while they are sitting on their couch.

Thus, web applications changed radically. They are either implemented as so-called "apps" for each particular platform. Or developer optimize the applications to be displayed appropriately on each device. With responsive web design the developer handle this challenge in a most efficient way. Responsive web design is a concept to create applications independently from the final screen resolutions or devices.

The author of this paper developed a CSS framework based upon responsive web design. The CSS framework supports web developer to reach their finished web application faster and more efficiently. It provides a basic range of functions. With a flexible basic grid the web application can be adjusted to different screen resolutions automatically. Additionally, the compatibility to different browsers was considered a crucial requirement for the framework. The features of the framework can be adapted for individual purpose by expanding it with flexible modules.

Kapitel 1

Einleitung

1.1 Motivation

Viele Informationen sind in einem riesigen Netzwerk 24 Stunden pro Tag, sieben Tage die Woche, frei verfügbar. Diese Tatsache begeistert immer wieder aufs Neue. Mit meiner Arbeit habe ich die Möglichkeit, mich detailliert mit dem Thema *Responsive Webdesign* und der Zukunft von flexiblen Webanwendungen auseinanderzusetzen.

Mit dem Einzug der *Smartphones* hat eine neue Ära der Internetnutzung begonnen. Für die Nutzer ergeben sich viele neue Anwendungsmöglichkeiten, da die *Smartphones* immer dabei und Informationen ständig verfügbar sind. Durch die einfache Bedienung mittels *Touchscreen* und zusätzlich eingebauter Sensoren sind viele neue, zum Beispiel orts- und zeitabhängige Anwendungen möglich. Nicht nur für die Nutzer sondern auch für die Herausgeber von Inhalten im WWW[1] ergeben sich neue Möglichkeiten, ihre Informationen zu verbreiten. Mittels kleiner Anwendungen, kurz *Apps*, die direkt auf dem *Smartphone* installiert sind oder einer mobilen Webanwendung, können Herausgeber ihre Informationen gezielt den Nutzern zur Verfügung stellen.

1.2 Problemstellung

Internetfähige Geräte werden immer vielfältiger, wie zum Beispiel das *Smartphone* oder ein *Tablet* für unterwegs, der Laptop oder Desktop in der Arbeit sowie der Fernseher für die gemütlichen Stunden nach Feierabend. All diese Geräte werden täglich von vielen Menschen für ein und die selbe Sache verwendet – um im Internet zu surfen. Das Größenverhältnis der verschiedenen Geräte könnte unterschiedlicher nicht sein. Ein *Smartphone* passt in eine Hosentasche und viele Fernseher entfalten erst ihre Leistung, wenn mindestens vier Meter Abstand eingehalten werden. Für Webentwickler ist diese Tatsache eine enorme Herausforderung, Inhalte so aufzubereiten und zu gestalten,

[1] World Wide Web

dass alle Geräte gleich oder zumindest ähnlich bedient werden können.

Die Webbrowser der einzelnen Geräte sind so ausgeführt, dass diese standardmäßig Inhalte auf die Größe des jeweiligen Gerätes skalieren. Eine optimale Darstellung ist dies leider nicht. Auf *Smartphones* wird eine Webanwendung meist kleiner skaliert, um diese vollständig anzuzeigen. Damit die Inhalte lesbar werden, muss der Benutzer diese von Hand größer skalieren. Umgekehrt verhält es sich bei Fernsehern. Die Inhalte werden größer dargestellt, lesbar sind sie aufgrund der Entfernung dennoch nicht, ohne sie größer zu skalieren. Es gibt verschiedene Möglichkeiten all diese Geräte mit optimiert dargestellten Inhalten zu bedienen.

Pro Plattform eine *App* zu entwickeln ist die erste Möglichkeit. Diese wird für die Darstellung jeder Plattform individuell optimiert. Dies bringt jedoch enorme Entwicklungskosten mit sich, was sich wiederum nachteilig auswirkt. Für kleinere Projekte und ein einziges Projektteam ist diese Möglichkeit wirtschaftlich nicht durchführbar.

Eine zweite Möglichkeit ist, jeweils eine eigene Version einer Webanwendung für verschiedene Bildschirmauflösungen zu entwickeln und diese für die Darstellung entsprechend zu optimieren. Vorteile sind die Flexibilität und ein überschaubarer Wartungsaufwand. Probleme bei dieser Methode sind, dass die Struktur der einzelnen Versionen oft unterschiedlich umgesetzt wird und dadurch der Wartungsaufwand wieder steigt.

Die dritte Möglichkeit ist das Konzept *Responsive Webdesign*. Dabei passt sich die Website automatisch an das jeweilige Gerät an. Wenn ein Benutzer mit einem *Smartphone* die Website aufruft, wird diese optimiert für das *Smartphone* dargestellt, genauso für Fernseher und andere Geräte. Die Struktur der Webanwendung bleibt immer gleich, es ändert sich nur die Darstellung der einzelnen Elemente, was von großem Vorteil ist. *Responsive Design* ist im Gegensatz zu den anderen Möglichkeiten weniger zeitaufwändig umzusetzen.

Responsive Webdesign ist eine Herangehensweise, geräte- und auflösungsunabhängige Webanwendungen zu gestalten bzw. eine Technik, um Webanwendungen generell benutzerfreundlicher zu machen. *Responsive Webdesign* wird oft mit dem Design mobiler Websites gleichgesetzt. Dies ist nur ein Teilaspekt. *Responsive Design* beinhaltet wesentlich mehr. Mit der Vielzahl verschiedener mobiler Endgeräte, steigt auch der Frust jener Webentwickler, die für ihre Kunden geräteabhängige Anpassungen von deren Webanwendung zur Verfügung stellen müssen. Meist wird das mittels einer *Subdomain* gelöst und für jedes Gerät eine eigene Version realisiert. Das ist zwar eine mögliche Lösung, der Wartungsaufwand wächst jedoch mit der Anzahl der verschiedenen Versionen.

Es ist also nicht die Frage ob, sondern wann Webentwickler beginnen, eine dieser Methoden für die Entwicklung zukünftiger Projekte einzusetzen. Dass der Trend in Richtung *Mobile* geht, zeigt auch die Statistik von Gartner (Abbildung 1.1) sowie die Statistik von Canalys (Abbildung 1.2).

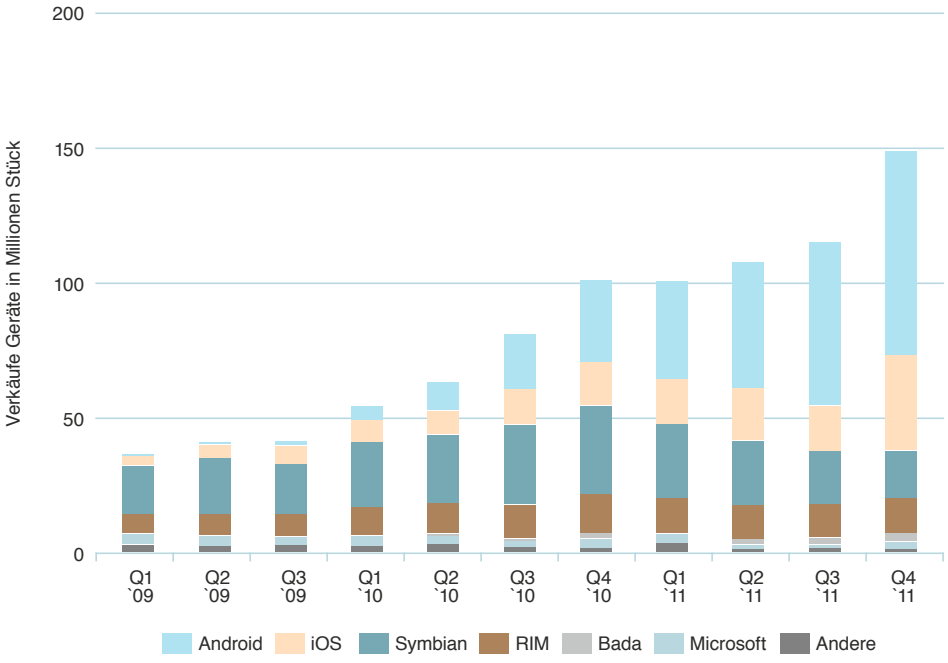


Abbildung 1.1: Verkaufszahlen von *Smartphones* weltweit nach Betriebssystem [10].

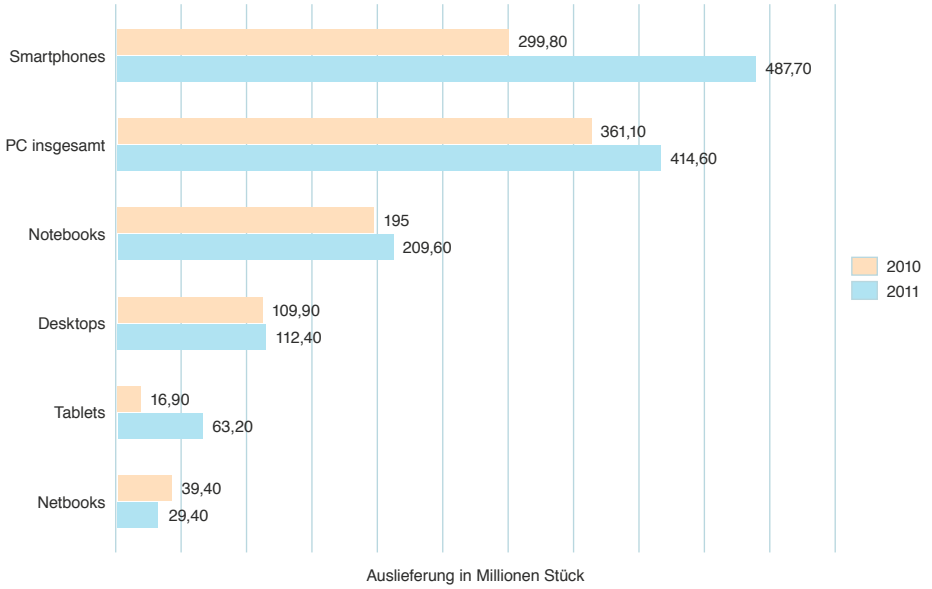


Abbildung 1.2: PC- und *Smartphone*-Auslieferung im Jahr 2010 und 2011 [20].

1.3 Zielsetzung

Aus der Sicht der Herausgeber ist es häufig sehr aufwändig bzw. teuer, ihre Informationen für alle Nutzer gleichermaßen zur Verfügung zu stellen. Es muss vor jedem Projekt abgewogen werden, welche Variante die richtige Realisierung darstellt. Es gibt kein Patent, das alle Anforderungen abdeckt. Eine *App* für jede Plattform plus einen Online-Dienst wäre sicher die performanteste Lösung, aber durch den Mehraufwand die teuerste.

Im Zuge dieser Arbeit wird ein *CSS-Framework* im Sinne des *Responsive Webdesign* und *Mobile First* realisiert. Das Ziel ist, Webdesigner dabei zu unterstützen, geräte- und plattformunabhängige Webanwendungen schneller und effizienter zu entwickeln.

1.4 Gliederung

Diese Arbeit setzt sich aus insgesamt sieben Kapiteln zusammen. In der Einleitung und auch in weiterer Folge dieser Arbeit wird auf die Probleme und möglichen Lösungsansätze eingegangen, Inhalte so aufzubereiten und zu gestalten, dass alle internetfähigen Geräte wie *Smartphones*, *Tablets*, Laptops, Desktops oder Fernseher gleich oder ähnlich bedient werden können.

Im Kapitel 2 werden die Grundlagen, welche für den weiteren Verlauf der Arbeit wichtig sind, defniniert und kurz beschrieben. Unter anderem werden die Neuerungen und Verbesserungen von *HTML5* und *CSS3* gegenüber den vorangegangenen Versionen erläutert.

Kapitel 3 widmet sich verwandten Ansätzen zum Entwickeln einer Webanwendung, die es Entwicklern ermöglichen, Webanwendungen mit flexiblen Benutzeroberflächen für verschiedene Endgeräte zu implementieren.

Kapitel 4 beschäftigt sich mit einem eigenen Lösungsansatz. Dieses Kapitel soll einen Einblick in die grundlegenden eigenen Überlegungen zum Thema dieser Arbeit geben. Ziel ist es, Webentwickler dabei zu unterstützen, schneller und effizienter zu einer geräte- und plattformunabhängigen Webanwendung zu gelangen. Für den Endnutzer soll höchstmögliche Benutzerfreundlichkeit und Barrierefreiheit erreicht werden.

Im Kapitel 5 wird die praktische Umsetzung des im vorigen Kapitel theoretisch beschriebenen Lösungsansatzes erläutert.

Kapitel 6 dieser Arbeit beschäftigt sich mit der Evaluierung des Systems. Bei dieser wird neben den Standardkriterien, die ein *CSS-Framework* erfüllen sollte, zusätzlich eine Beispielanwendung mit dem jeweiligen System umgesetzt. Diese dient dem Vergleich der unterschiedlichen *Workflows* und gleichzeitig der Browserkompatibilität.

Im abschließenden Kapitel 7 wird ein Résumé über die gesamte Arbeit gezogen. Weiters wird ein kurzer Einblick in die Möglichkeiten zur Weiterentwicklung des eigenen Systems gezeigt.

Kapitel 2

Grundlagen

Um die Ausführungen im weiteren Verlauf dieser Arbeit zu verstehen, ist es wichtig, einige grundlegende Begriffe zu kennen. Diese werden im folgenden Kapitel erörtert.

Die Definition von Webdesign ist durch die mehrdeutige Verwendung des Begriffes nicht einfach zu erfassen. Im deutschen Sprachgebrauch wird der Begriff „Design“ mit der Gestaltung gleichgesetzt. Webdesign versteht sich aber nicht als Gestaltung des Erscheinungsbildes einer Webanwendung sondern vielmehr als Prozess der Entstehung. Es verbirgt sich hinter dem Begriff viel mehr, wobei die Gestaltung nur ein kleiner aber nicht unwichtiger Teil ist. Webdesign wie es mittlerweile praktiziert wird oder praktiziert werden sollte geht einher mit Projektmanagement, verteiltes Arbeiten im Team und diversen Optimierungsprozessen. Die Definition aus Wikipedia[1] lautet wie folgt:

> Webdesign (auch Webgestaltung) umfasst die Gestaltung, den Aufbau und die Nutzerführung von Websites für das World Wide Web und das *Interface-Design* in diesem Bereich. Der „Webdesigner“ hat dabei die Aufgabe, die Kommunikationsziele des Auftraggebers mit Hilfe der technischen Gegebenheiten umzusetzen.

2.1 Technologien

Durch neue Technologien im Bereich des Webdesigns ergeben sich für Entwickler viele neue und spannende Möglichkeiten Webanwendungen zu entwickeln. Mit *CSS3*, *HTML5* und den zur Verfügung stehenden *APIs*[2] können in Zukunft bessere Anwendungen mit weniger Zeitaufwand umgesetzt werden. Die nachfolgenden Abschnitte bieten einen Überblick über die Neuerungen

[1] http://de.wikipedia.org/wiki/Webdesign
[2] engl. application programming interface

sowie Verbesserungen von *HTML5* und *CSS3* gegenüber den vorangegangenen Versionen.

HTML ist eine Auszeichnungssprache, mit der Webseiten im Internet textuell strukturiert werden. Im folgenden Abschnitt findet sich die Definition aus Wikipedia [8]:

> *HTML5* ist eine textbasierte Auszeichnungssprache zur Strukturierung und semantischen Auszeichnung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten. Die Sprache befindet sich zurzeit noch in der Entwicklung, es liegen aber bereits recht ausgereifte Entwürfe zweier Entwicklerteams vor.

Mittels *CSS* können Aussehen und Layout einer Webseite definiert werden. Wikipedia definiert *CSS3* folgendermaßen [9]:

> Die *Cascading Style Sheets Level 3* steht für stufenförmige oder hintereinander geschachtelte Gestaltungvorlagen, kurz *CSS3* genannt, sind quasi eine deklarative Sprache für Stilvorlagen von strukturierten Dokumenten wie *HTML*.

2.1.1 HTML5

HTML5 wird vom *W3C*[3] und der *WHATWG*[4] entwickelt. Der *HTML5* Standard ist noch nicht verabschiedet, das heißt, dass sich manche Definitionen noch ändern können. Dieser Umstand bedingt, dass die meisten Browserentwickler vorerst ihre eigenen Vorstellungen implementieren und sich somit Inkonsistenzen im Markup sowie in der Darstellung durch die unterschiedlichen Browser ergeben. Erschwerend hinzu kommt, dass sich aus der gemeinsamen Arbeit des *W3C* und der *WHATWG* zwei parallele Entwicklungen gebildet haben. Das *W3C* ist zuständig für die Standardisierung im Web und die *WHATWG* will mit einem lebendigen Standard die Entwicklung und Umsetzung schneller vorantreiben. Mehr dazu findet sich in [18].

Die Definition von *HTML5* ist nicht eindeutig. Es wird von verschiedenen Autoren sowie verschiedenen Gruppen jeweils anders definiert. Allgemein geht die Definition über eine reine Textauszeichnungssprache hinaus. Es ist eine neue Webtechnologie, die mehrere Technologien beinhaltet. *CSS3* sowie *JavaScript* sind ebenfalls Technologien, die in *HTML5* beinhaltet sind. Die Abbildung 2.1 zeigt einen groben Überblick der Bestandteile von *HTML5*

Neue Elemente

HTML5 bringt viele neue Elemente für die Strukturierung einer Webanwendung und somit einige Vorteile mit sich. Der Programmcode wird dadurch

[3] World Wide Web Consortium, http://www.w3.org

[4] Web Hypertext Application Technology Working Group, http://www.whatwg.org

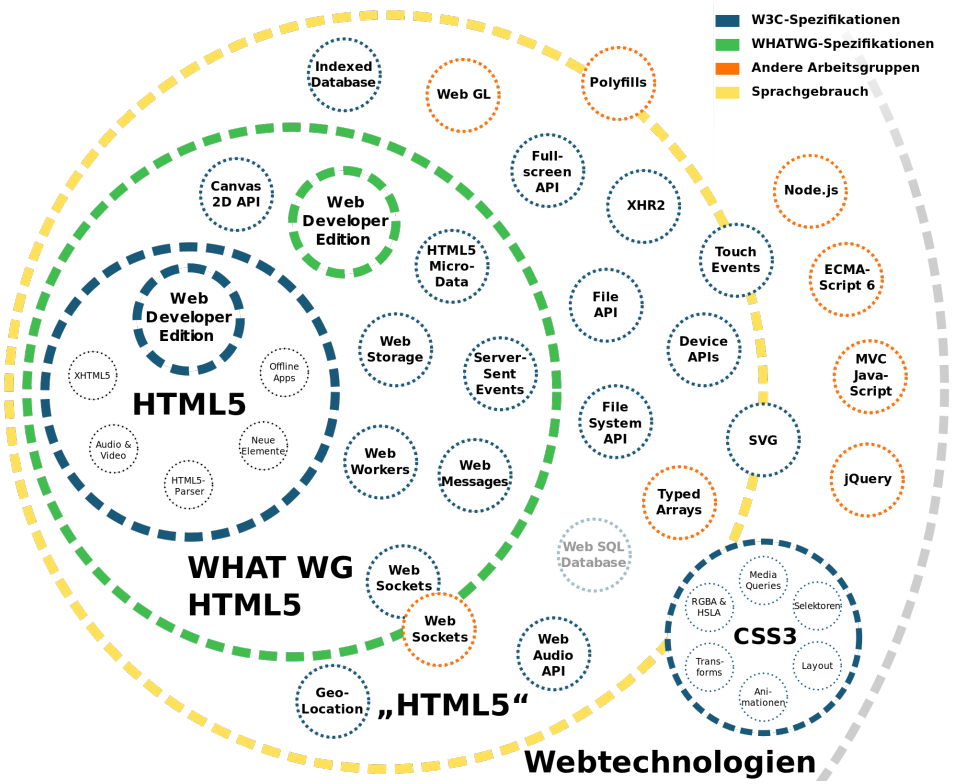


Abbildung 2.1: Verhältnis von diversen *HTML5* Webtechnologie-Spezifikationen zueinander [16].

reduziert, dass die semantische Auszeichnung dieser Elemente bereits einer Bedeutung zugeordnet werden kann. Suchmaschinen können durch diese Bedeutungen relevantere Ergebnisse liefern. Zusätzlich kann dieser Programmcode einfacher von Entwicklern gelesen werden.

Auch die Barrierefreiheit im Internet kann aus diesen neuen Elementen einen Vorteil ziehen. *Screenreader* beispielsweise können die Inhalte relevanter Elemente wie `article` vorlesen und unwichtigere wie `aside` außen vor lassen.

Attribute

HTML5 bietet die Möglichkeit, einfach ein benutzerdefiniertes Attribut in einem Element zu erstellen. Dieses kann mittels *JavaScript* ausgelesen und verarbeitet werden.

Formulare

Auch für Formulare gibt es durch *HTML5* einige Neuerungen. Es wurden eine Vielzahl neuer Eingabetypen im Standard definiert. Diese neuen Typen haben jeweils eine semantische Bedeutung. Durch diese semantische Bedeutung wird die Validierung erleichtert. Es muss nicht jedes Feld einzeln auf alle möglichen Varianten von Mustern abgefragt werden. Viele aktuelle Webbrowser haben bereits eine clientseitige Validierung inkludiert – das heißt jedoch nicht, auf eine serverseitige Validierung komplett zu verzichten.

Die möglichen Eingaben können mit neuen Attributen zusätzlich eingeschränkt werden. Das `min`- bzw. `max`-Attribut beispielsweise ermöglicht die Eingaben einer Zahl in einem gewissen Bereich einzuschränken. Mittels Attribut `required` wird ein Formularfeld als Pflichtfeld definiert. Was früher mit einem aufwändigen Workaround meist in *JavaScript* realisiert wurde, wird heute mit dem Attribut `placeholder` realisiert. Dabei wird der Platzhaltertext des Formularfeldes automatisch ausgeblendet, wenn dieses den Fokus erhält.

Rich Media

Das *Canvas*-Element ist ein in *HTML* definierter Bereich, der mittels Angabe der Breite bzw. Höhe definiert wird. Es ist eine Art Leinwand, die mittels *JavaScript* „bemalt“ werden kann. Neben normalen Linien- sowie Rechteckzeichenfunktionen gibt es noch eine Reihe weiterer Zeichenfunktionen, die beispielsweise Transparenzen erlauben. Neben dem simplen Zeichnen können auch einzelne Objekte verschoben, skaliert und rotiert werden. Damit können Animationen direkt und nativ im Browser realisiert werden. Eine genaue Beschreibung findet man in der Referenz [14] des *W3C*.

Eine weitere Verbesserung durch den neuen *HTML5*-Standard ist die native Einbindung von Videos bzw. Sounds in eine Webanwendung. Das heißt im Wesentlichen, dass der Entwickler sich nicht mehr um die Einbindung eines Abspielgeräts kümmern muss. Es muss einzig und allein das Material im richtigen Dateiformat vorliegen. Leider gibt es kein Format, das durchgängig von allen neueren Browsern unterstützt wird. Somit müssen diese in mehreren Formaten vorliegen. Der Player ist bei neueren Browsern eingebettet. Mittels Attributen kann der Player parametriert werden, um zum Beispiel ein Video oder einen Sound direkt nach dem Laden der Webanwendung zu starten. Es gibt auch für ältere Browser, die diese neuen Elemente noch nicht unterstützen, eine Fallback-Variante, indem innerhalb des Elements ein eigener Player, beispielsweise über Flash, zur Verfügung gestellt wird.

Weitere HTML5 Technologien

Wie aus Abbildung 2.1 ersichtlich, gibt es noch eine Reihe weiterer Technologien die mit *HTML5* entwickelt wurden. Zwei Technologien, die für die

mobile Webentwicklung relevant sind, werden an dieser Stelle vorgestellt.

Einerseits *Geolocation*[5], die es ermöglicht, über eine Webanwendung die Position des Benutzers zu lokalisieren. Hierbei werden verschiedene Faktoren zur Lokalisierung herangezogen. Es wird über die IP-Adresse, WLAN, GPS und andere Quellen der Standort mit einer Genauigkeit übermittelt, die von den verfügbaren Quellen abhängt.

Andererseits die Technologie für *Web Storage*[6], die es ermöglicht, Webanwendungen direkt auf dem Endgerät laufen zu lassen, ohne eine Verbindung zum Internet aufzubauen. Dies funktioniert, indem die gesamte Webanwendung auf den Client übertragen und gespeichert wird. Die nötigen Daten bzw. die eingegebenen Daten werden über eine Art Datenbank, die Bestandteil des Browsers ist, gespeichert. Es unterscheiden sich zwei Varianten wie lange die eingegebenen Daten gespeichert werden. Bei der Variante *Local Storage* werden Usereingaben unbegrenzt gespeichert. Es verhält sich somit ähnlich wie eine native *App*. Bei der Variante *Session Storage* werden Benutzereingaben für die Dauer einer aufrechten *Session* gespeichert. Bei einem Neustart des Endgerätes sind die Daten weg.

2.1.2 CSS3

Auch *CSS3* bringt wie *HTML5* einige Neuerungen, die sich positiv auf die Entwicklung von Webanwendung, besonders in Bezug auf mobile Webentwicklung, auswirken. Das wichtigste Instrument für die mobile Webentwicklung sind die sogenannten *media queries*, die in Abschnitt 2.2.3 beschrieben sind.

Mittels neuer Selektoren, die in *CSS3* eingeführt wurden, können Elemente noch gezielter über ihre Position in der Element-Struktur angesprochen werden. Damit wird die Vergabe von Klassen und IDs deutlich reduziert. Daraus resultiert ein schlanker und sauberer Code. Beispielsweise können mit Attribut-Selektoren auf einfache Art und Weise für verschiedene Link-Typen verschiedene *Icons* definiert oder mittels Struktur-Pseudoklassen das erste oder das letzte Kind-Element einfach gefiltert und anders gestylt werden.

Ein weiteres Beispiel, das in den Vorversionen nicht so leicht umzusetzen war, ist, abgerundete Ecken für Block-Elemente zu definieren. Dies erreicht man mittels einfacher `Eigenschaft: Wert` Kombination wie zum Beispiel `border-radius: .3em`.

Schriften zu verwalten, die nicht auf dem Client installiert sind, war bis *CSS3* fast unmöglich. Mittels der Neuauflage von `@font-face` bei der Definition von *CSS3* wurde ein neuer Weg in Richtung Typografie im Internet geebnet. Schriften können nun einfach eingebetet werden. Die Technik der Webfonts ist mittlerweile auch in verschiedenen Browsern richtig umgesetzt.

[5] http://www.w3.org/TR/2010/CR-geolocation-API-20100907/
[6] http://www.w3.org/TR/webstorage/

Mit *CSS3* ist der Weg für die kreative Gestaltung von mobilen Webanwendungen geebnet worden. Es gibt eine Reihe weiterer Neuerungen, auf die in [12] näher eingegangen wird.

2.1.3 Apps

Zum Grundverständnis wird der Begriff *App* definiert. Nachfolgend werden einige Techniken, die für mobile Webentwicklung wichtig sind, genauer unter die Lupe genommen.

Der Begriff *App* kommt vom englischen Wort „Application“ – was soviel bedeutet wie Computer- bzw. Anwendungsprogramm. Dieser wurde mit der Einführung des *App-Stores*[7] von Apple stark geprägt. Laut Duden[8] ist eine App eine zusätzliche Applikation, die auf bestimmte Mobiltelefone heruntergeladen werden kann. *Smartphones* sind, Mobiltelefone mit größerem Bildschirm, die Toucheingaben per Finger oder Stift ermöglichen. Der Begriff *App* wird im allgemeinen Sprachgebrauch aber anders verwendet. Eine *App* zeichnet sich dadurch aus, dass diese unkompliziert auf einem Endgerät mittels einem vom Betriebssystem mitgelieferten *Store* (iOS: „AppStore“[9], Android: „Google Play Store“[10], Windows Phone 8: „Marketplace“[11]) installiert werden kann. Die meisten *Apps* werden heute zusätzlich für *Tablets* angeboten.

Microsoft[12] und Apple[13] als Hersteller von klassischen Desktop-Betriebssystemen gehen hier noch einen Schritt weiter und weiten die Store-Infrastruktur für zusätzliche Anwendungsprogramme in ihre Desktop-Betriebssysteme aus.

Apps lassen sich grundsätzlich in zwei Arten aufteilen: Native *Apps* und *Web-Apps*. Hybride *Apps* sind eine dritte Art, welche eine Mischung aus den beiden vorangegangen ist. Hierbei stellt ein *Framework*, wie zum Beispiel *PhoneGap*[14] eine Schnittstelle zur nativen Anwendung bietet. Die Entwicklung erfolgt dabei mittels den herkömmlichen Techniken der Webentwicklung.

Native Apps

Native *Apps* werden deswegen *native* genannt, weil diese in der Sprache für die jeweilige Plattform programmiert sind. Beispielsweise werden *Apps* für

[7] http://www.apple.com
[8] http://www.duden.de/rechtschreibung/App
[9] https://itunes.apple.com/de/genre/ios/id36?mt=8
[10] https://play.google.com
[11] https://www.windowsphone.com/de-de/store
[12] http://windows.microsoft.com/de-AT/windows-8/apps
[13] https://itunes.apple.com/de/genre/mac/id39?mt=12
[14] http://phonegap.com

iOS in der Sprache *Objective-C*[15] und *Apps* für Android in *Java*[16] geschrieben.

Diese *Apps* werden direkt auf dem Endgerät installiert. Dadurch sind sie auch bei einer fehlenden Mobilfunk- bzw. Internetverbindung mittels Wireless LAN einsatzbereit.

Die jeweiligen Plattformen stellen viele Komponenten und Funktionen für die Entwickler bereit. Beispielsweise werden Schnittstellen zu den gerätespezifischen Sensoren direkt bereitgestellt. Durch die Richtlinien, die einem Entwickler vorgeschrieben werden und die gerätenahe Umsetzung der Anwendungen, sind diese gegenüber den anderen Arten viel performanter. Diese Performance wird meist bei grafisch aufwändigen Anwendungen wie beispielsweise Spielen benötigt.

Um eine *App* anzubieten reicht es, diese über die jeweilige Plattform in deren *Store* zu stellen. Der Entwickler muss hierfür einen kostenpflichtigen *Account* für den jeweiligen *Store* erwerben. Dafür stellt die jeweilige Plattform die gesamte Infrastruktur für den Vertrieb einer *App* zur Verfügung. Diese wird kategorisiert und über den jeweiligen *Store* angezeigt. Lediglich ein Teil des Umsatzes sowie die jährliche Gebühr für den Account werden dem Entwickler in Rechnung gestellt. Die Aufnahme einer *App* in die verschiedenen *Stores* unterliegt einer Prüfung der jeweiligen Plattform. Viele Richtlinien sowie Bedingungen können jedoch zu einem langen und aufwändigen Aufnahme- bzw. Updateprozess führen.

Die Vorteile einer nativen *App* sind einerseits die Performance, die über eine *Web-App* nicht erreicht werden kann und andererseits der für die Plattform eigene *Store*, über diesen viele Nutzer erreicht werden und der dadurch den Vertrieb enorm erleichtert. Der Nachteil liegt auf der Hand, eine *App* die auf mehreren Plattformen Einzug finden soll, muss somit für mehrere Plattformen entwickelt werden. Dies vervielfacht die Entwicklungs- wie auch die Wartungskosten.

Web-Apps

Web-Apps werden plattformunabhängig mittels Web-Technologien (*HTML*, *CSS* und *JavaScript*) entwickelt. Diese werden im Browser des jeweiligen Endgerätes ausgeführt und unterliegen somit lediglich den Grenzen, den die Browser bieten.

Meist sind die Browser nicht so performant, wie Benutzer es von nativen *Apps* gewohnt sind. Die Entwickler bzw. die Herausgeber ersparen sich die Prüfung durch die *Stores*, weil diese Anwendungen erst gar nicht in einem *Store* aufscheinen können. Die plattformabhängigen *Stores* sind den nativen *Apps* vorbehalten. Der Vertrieb der *App* oder des Dienstes muss somit eigens organisiert und entwickelt werden. Ein weiterer Nachteil, den *Web-Apps* mit

[15] https://developer.apple.com/library/mac/navigation/
[16] http://developer.android.com/reference/packages.html

sich bringen, ist die mangelnde Unterstützung für die Hardware und Sensorik der einzelnen Geräte. Beispielsweise kann nicht direkt auf die Kamera oder den Kompass zugegriffen werden.

Web-Apps haben trotzdem viele Vorteile für die mobile Webentwicklung. Sie können plattform-, geräte- sowie auflösungsunabhängig entwickelt werden. Das heißt es gibt eine einzige *App* für alle Plattformen sowie Geräte. Angefangen vom *Smartphone* über *Tablets* bis hin zum Desktop PC. Diese Tatsache erspart sehr viel an Entwicklungskosten und auch laufende Kosten bei der Wartung.

Entwickler müssen sich klar über die Grenzen von *Web-Apps* bewusst sein und je nach Anforderung abwägen, welche Art dafür besser geeignet ist.

2.2 Webdesign

2.2.1 Methode der Fehlertoleranz

State of the Art im Webdesign ist die Methode der Fehlertoleranz[17]. Dieses Konzept beschreibt den Weg der Entwicklung einer Website. Hierbei orientiert sich das Webdesign hauptsächlich an den möglichen Webtechnologien. Im ersten Schritt wird die Webanwendung mit allen Funktionalitäten und dessen Design für neueste Webbrowser entwickelt, ohne Rücksicht auf Abwärtskompatibilität. Im zweiten Schritt wird versucht, einerseits die Funktionalitäten und andererseits die Darstellung für ältere Browser zu integrieren. Die Missstände die hierbei entstehen beschreibt Aaron Gustafson in seinem Buch *Adaptive Web Design* [2, Kap. 1].

Umgelegt auf die Entwicklung mobiler Webanwendungen heißt das, dass eine fertige Desktop-Version einer Webanwendung sukzessive zur mobilen Version hin optimiert wird. Dies ist derzeit für Webanwendungen, die vor zwei oder mehr Jahren erstellt wurden und noch immer laufen, oft der Fall. Dass hierbei oft das Nutzererlebnis darunter leidet, wird durch geringere Kosten gerechtfertigt.

Dieses Konzept hat auch in der heutigen Zeit noch seine Berechtigung. Es kommt immer auf den Fokus der Webanwendung an. In dieser Arbeit wird die Methode der Fehlertoleranz nicht als die Abwärtskompatibilität in Form von technologischer Barrierefreiheit bezeichnet. Es bezeichnet die optische Anpassung, ausgehend von der Version für Desktop Computer zu der Version mobiler Endgeräte.

2.2.2 Methode der progressiven Verbesserung

Ein eher neueres Konzept in der Webentwicklung spiegelt das Konzept der progressiven Verbesserung[18] wider. Hierbei wird darauf geachtet, dass jeder

[17] engl. graceful degradation
[18] engl. progressive enhancement

Benutzer den gleichen Inhalt erhält, egal welche Technologie dieser nutzt. Auch die Funktionalität einer Webanwendung ist für jeden Anwender gleich. Einzig die Präsentation sowie die Benutzerfreundlichkeit wird zusätzlich für verschiedene Technologien optimiert.

Der Inhalt steht im Vordergrund der Anwendung, dessen Strukturierung über standardkonformes *HTML* erfolgt. Durch das Markup kann jede Webtechnologie den Inhalt und die Struktur wiedergeben. Dem Benutzer werden keine Barrieren in den Weg gestellt. Aufbauend auf der einfachen Aufbereitung des Inhaltes wird sukzessive für neuere Webtechnologien optimiert. Es werden Layout und Design mittels *CSS* hinzugefügt. Zusätzlich werden über *JavaScript* weitere Optimierungsschritte durchgeführt, um das Benutzererlebnis weiter zu steigern. Der Kern wird aber nicht verändert und der Inhalt ist somit für alle Technologien gleich.

Ziel ist, dass die Inhalte nicht verfälscht werden, diese aber aufbereitet ein besseres Benutzererlebnis bieten können. Durch die unterschiedlichen Browser und deren unterstützte Technologien ist es von Vorteil, die Barrieren schon im Vorhinein auszuschließen. Es hilft nicht, eine top moderne und ansehnliche Webanwendung zu entwickeln, wenn die Hauptzielgruppe diese nicht verwenden kann, da die Hälfte der Inhalte bzw. Funktionalitäten nicht funktionieren, weil ein veralteter Browser im Einsatz ist. In dem Buch *Designing with Progressive Enhancement* [6] werden die Methoden für die Entwicklung von barrierefreien Webanwendungen ausführlich beschrieben.

2.2.3 Responsive Webdesign

Responsive Webdesign ist keine neue Technologie, es ist vielmehr eine Idee bzw. ein Prozess beim Webdesign, der eine einzige Webanwendung für eine Vielzahl von Geräten mit verschiedenen Auflösungen unterstützt. Ethan Marcotte beschreibt in seinem Buch *Responsive Web Design* [4] den Weg zur geräte- bzw. auflösungsunabhängigen Webanwendung durch bestehende Webtechniken sowie Prinzipien. Diese sind:

- flexible rasterbasierte Layouts,
- flexible Bilder und andere *Rich Media* Formate,
- *media queries*, Bestandteil der *CSS3*-Spezifikation.

Layout mittels Gestaltungsraster

Der Gestaltungsraster[19] ist ein Ordnungssystem in der visuellen Kommunikation. In der modernen Kunst der 1960er experimentierten Künstler mit reduzierten Ausdrucksformen. Es kamen einfache Objekte wie Linien, Flächen und Farben zum Einsatz. Ziel dabei war, ein harmonisches Gesamtbild in der Gestaltung zu schaffen. Josef Müller-Brockmann beschreibt in sei-

[19] engl. grid-system

nem Buch *Grid systems in graphic design* [5] einen Prozess zum Layouten von Typografie auf einer Seite. Dabei wird die Proportion des Gestaltungsrasters mit dem Format und der Größe eines Blatt Papiers in Verbindung gebracht. Ein Gestaltungsraster ist ein rationales System aus Spalten und Zeilen. Daraus ergeben sich Flächen, die mit Inhalten wie Texte und Bilder gefüllt werden können.

Dieses Konzept lässt sich leider nicht eins zu eins auf die Gestaltung einer Webanwendung übertragen. Ein Schlüsselelement, das Blatt Papier, fehlt völlig. Die Bühne für die Gestaltung einer Webanwendung ist das Browserfenster. Das Browserfenster ist aber ein flexibles Element, das sich vom Benutzer in Größe und Form jederzeit verändern lässt. Dieser Umstand verlangt, dass ein Gestaltungsraster innerhalb eines definierten Bereiches fixiert wird.

Aus diesem Grund findet sich in der Praxis häufig ein Konstrukt, bestehend aus einem Container, der mittels *CSS*-Anweisung in der Mitte des Browserfensters angezeigt wird und eine fixe Breite aufweist. Folgendes Beispiel zeigt eine rudimentäre Umsetzung eines solchen Konstrukts mittels *CSS*.

```
1 #wrapper {
2   width: 960px;
3   margin: 0 auto;
4 }
```

Damit kommt man dem Blatt Papier und dem Format wieder näher, wenn auch nur in der Breite. Der Grundstein für ein *Grid*-System ist nun gelegt.

Gestaltungsraster ermöglichen die Anordnung von Informationen nach einem bestimmten Schema. Die Unterteilung erfolgt nach grafisch ansprechenden Strukturen. Moderne Webanwendungen basieren meist auf einem ein-, zwei- bzw. dreispaltigen Layout. Sie bestehen aus einer definierten Anzahl von Spalten, die über *CSS*-Klassen im Markup angesprochen werden können.

Vorteile von Gestaltungsrastern:

- Raster ermöglichen Ordnung, Kontinuität und Harmonie von bereitgestellten Informationen.
- Raster ermöglichen den Usern die strukturierten Informationen leichter zu finden und unterstützen somit die Informationsvermittlung.
- Raster erleichtern das Hinzufügen neuer Informationen in konsistenter Art und Weise, ohne das Layout zu zerstören.

Fester Raster

Als feste Raster werden Rastersysteme mit absolut definierten Größen bezeichnet. Diese haben eine definierte Anzahl an Spalten mit einer fixen Brei-

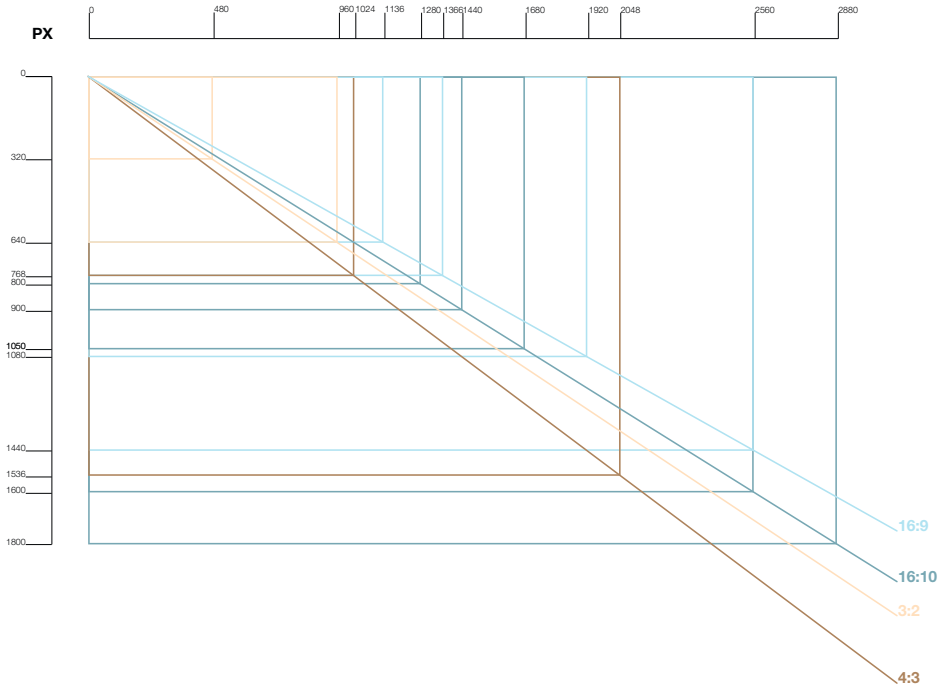


Abbildung 2.2: Auflösungsunterschiede aktueller Darstellungsgeräte, von *Smartphones* bis zu gängigen Desktop Monitoren.

te und einem Abstandhalter, die jeweils in Pixel angegeben sind. Populäre Beispiele für Rastersysteme mittels festem Raster sind *960.gs*[20] oder *Blueprint*[21]. Feste Raster orientieren sich meist am mittleren Wert der Bildschirmauflösungen. War 1024px in der Breite ein guter Mittelwert in den letzten Jahren so ist dies heute mit der Vielzahl an verschiedenen Geräten mit verschiedenen Auflösungen nicht mehr gegeben wie in Abbildung 2.2 gezeigt wird.

Flexibler Raster

Anstelle von festen Werten, die herkömmlicherweise in Pixel angegeben werden, wird ein flexibler Raster mittels relativen Prozentwerten definiert. Somit wird das Layout der Webanwendung dynamisch und reagiert auf Veränderungen der im Browser verfügbaren Fläche (*Viewport*). Um eine korrekte Darstellung wie sie gewünscht wird zu erreichen, müssen die festen Werte umgerechnet werden. Ein Element auf einer Seite muss relativ zum Elternelement berechnet werden. Die Größe des *Viewports* ist die maximale Aus-

[20] http://960.gs
[21] http://www.blueprintcss.org

dehnung, die ein Element annehmen kann. Ethan Marcotte beschreibt in seinem Artikel *The fluid Grid* [17] die Methode, wie aus einem festen Rastersystem ein flexibles Rastersystem wird. Die Umrechnung erfolgt mit einer einfachen Formel:

$$\frac{\text{Ziel}}{\text{Kontext}} = \text{Ergebnis in Prozent.}$$

Flexible Bilder

Innerhalb der flexiblen Layouts sind auch flexible Bilder notwendig. Diese müssen sich ebenfalls automatisch der Größe des *Viewports* anpassen. Aufgrund einer fehlenden standardisierten Lösung, werden Bilder einfach relativ auf die richtige Größe skaliert. Dies hat den Nachteil, dass bei großen Bildern die Bandbreite leidet. Es gibt zwar einige clientbasierte Lösungen mit *JavaScript* und auch serverbasierte Lösungen für dieses Problem. Eine standardisierte Lösung wird aber noch kontrovers diskutiert[22] und entwickelt.

media queries

Letzter Bestandteil des Konzepts vom *Responsive Webdesign* sind die im *CSS3* definierten *media queries*[12]. Die *media queries* erlauben dem Entwickler anhand von logischen Abfragen die Steuerung der Darstellung innerhalb von *CSS*. Mit *media queries* kann die *Viewport*-Größe abgefragt werden. So ist es möglich, nicht nur die flexiblen Layouts der Größe anzupassen sondern bedarfsgemäß einzelne Elemente der Anwendung in Form und Position zu verändern. Es können auch Bereiche zur Gänze ein- bzw. ausgeblendet werden.

Diese drei Techniken, flexibler Raster, flexible Bilder und *media queries* in Kombination ermöglichen einer Webanwendung, je nach Endgerät- bzw. *Viewport*-Auflösung verschiedene Layoutformate automatisch zur Verfügung zu stellen. Der Inhalt wird für jede Größe optimiert dargestellt. Dem Benutzer der über einen Desktop die Webanwendung ansteuert, bietet sich ein anderes Bild als jenem, der die Anwendung über ein *Tablet* oder *Smartphone* aufruft.

2.2.4 Mobile First

Bei der Entwicklung mobiler Webanwendungen gibt es zwei unterschiedliche Herangehensweisen, einerseits den Ansatz *Desktop First* und andererseits den Ansatz *Mobile First*.

Wie der Name schon sagt wird beim *Desktop-First*-Ansatz zuerst die Version für den Desktop entwickelt und nach und nach auf die mobile Version

[22] http://www.peterkroener.de/die-responsive-images-story/

abgespeckt. Vergleichbar ist dieses Vorgehen mit der Methode der Fehlertoleranz, siehe Abschnitt 2.2.1. Bei diesem Vorgehen wird keine Rücksicht auf die verfügbare Bandbreite, die unter Umständen bei mobiler Nutzung entstehen kann, genommen. Zusätzlich bietet eine Desktop-Version mehr Platz für Designelemente an, die sich beim Abspecken auf die mobile Version oft nicht leicht übertragen lässt.

Luke Wroblewski propagiert seit einiger Zeit den Begriff *Mobile First*. In dem gleichnamigen Buch [7] beschreibt er die Gründe und das Vorgehen für Entwickler für diesen Ansatz. Beim *Mobile-First*-Ansatz wird, wie der Name schon sagt, die mobile Version einer Webanwendung als erstes entwickelt. Dieses Vorgehen ist vergleichbar mit der Methode der progressiven Verbesserung, siehe Abschnitt 2.2.2. Aber nicht nur Wroblewski sondern auch andere Größen im Web-Business wie beispielsweise Eric Schmidt[23] und Kate Aronowitz[24] vertreten diesen Ansatz. Mehr zu den Aussagen in [11] und [19].

Die Gründe sieht Wroblewski in der steigenden Verbreitung von *Smartphones*. Die Statistiken von Gartner (Abb. 1.1) sowie die Statistik von Canalys (Abb. 1.2) belegen diese Beweggründe. Daneben zeigt er noch eine Reihe weiterer Gründe auf, wie beispielsweise steigende Umsätze, die durch getätigte Käufe über *Smartphones* entstehen sowie steigende Netzwerkverkehrsdaten über UMTS, uvm.

Durch die Beschränkungen, denen mobile Geräte unterworfen sind, gewinnt eine Webanwendung für Desktops den Fokus zurück und dieser wird sich auch auf die Desktop-Version übertragen. *Mobile First* zwingt die Designer und Entwickler vieles zu überdenken. Nutzlose Navigation sowie Platzfüller durch irrelevante Inhalte werden dahingehend optimiert, dass diese überarbeitet bzw. gelöscht werden.

Mobile Netzwerke sind bei der Netzabdeckung sowie der Übertragungsgeschwindigkeit relativ instabil und teilweise lückenhaft. Dies soll auch bei der Entwicklung berücksichtigt werden und es empfiehlt sich, entsprechend bandbreitenschonend zu arbeiten.

2.3 Webdesignprozess

Für viele Webdesigner ist ein Webdesign kein einzelner Arbeitsschritt sondern ein Prozess, der die Entstehung eines Webauftritts oder einer Webanwendung beschreibt. Dieser Prozess folgt oft einer strukturierten Ordnung, wo die Gestaltung am Anfang und die Umsetzung bzw. Programmierung am Ende steht.

Am Anfang steht immer eine Idee. Eine Website bzw. Webanwendung sollte sich immer mit einem einzigen Thema beschäftigen. Ob es sich dabei um eine Firma handelt, die ihre Produkte und Referenzen präsentiert oder ob

[23] http://de.wikipedia.org/wiki/Eric_Schmidt
[24] http://www.linkedin.com/in/katearonowitz

es eine Anwendung für eine bestimmte Aufgabe ist, spielt dabei keine Rolle. Nachfolgend wird ein heute gängiger Prozess im Webdesign beschrieben:

2.3.1 Inspirationen sammeln

Um mit der Gestaltung des äußeren Erscheinungsbildes der Webanwendung zu beginnen, ist es hilfreich, sich mittels ähnlich gestalteter Websites bzw. Anwendungen einen Grundstock an Gestaltungsvorgaben anzueignen.

Das Imageboard

Ein *Imageboard* oder *Moodboard* ist eine Anhäufung von Bildern aus Zeitschriften oder dem Internet, die zum Thema der zu gestaltenden Webanwendung passen. Es werden gesammelte Bilder in Form einer Fotocollage arrangiert. Durch diese Inspirationen verschiedenster Bilder wird die Gestaltung des Erscheinungsbildes der Webanwendung erleichtert, indem immer wieder Bezug darauf genommen werden kann. Wenn verschiedene Personen in einem Team arbeiten, kann ein *Imageboard* zur Vermittlung der konkreten Vorstellung, das Aussehen betreffend, verwendet werden.

Inspirierende Websites

Inspirierende Websites können, müssen aber nicht unbedingt ein ähnliches Thema der geplanten Webanwendung haben. Bei diesem Schritt holt sich der Webdesigner Inspiration bzgl. Menüstrukturen, Navigationselementen, Slideshows sowie vielen weiteren interaktiven Elementen, die ebenfalls in dieser oder ähnlicher Form bei der geplanten Webanwendung Anwendung finden können.

2.3.2 Material für die Gestaltung sammeln

Bevor der Gestaltungsprozess startet ist es hilfreich, Materialien für die Gestaltung zu sammeln. Für jedes Gestaltungsprogramm gibt es eigene Methoden verschiedene Erweiterungen zu laden.

Muster, Texturen und Photoshop Pinselsets

Wenn für die Gestaltung mit Adobe[25] gearbeitet wird, ist es sinnvoll sich Muster und Texturen für die Gestaltung der Hintergründe sowie verschiedene Pinselsets im Vorhinein zu besorgen. Hier sind beispielsweise *deviantART*[26] bzw. *myPhotoshopBrushes*[27] gute Adressen.

[25] http://www.adobe.com/products/creativesuite/design-web-premium.html
[26] http://www.deviantart.com/
[27] http://myphotoshopbrushes.com/

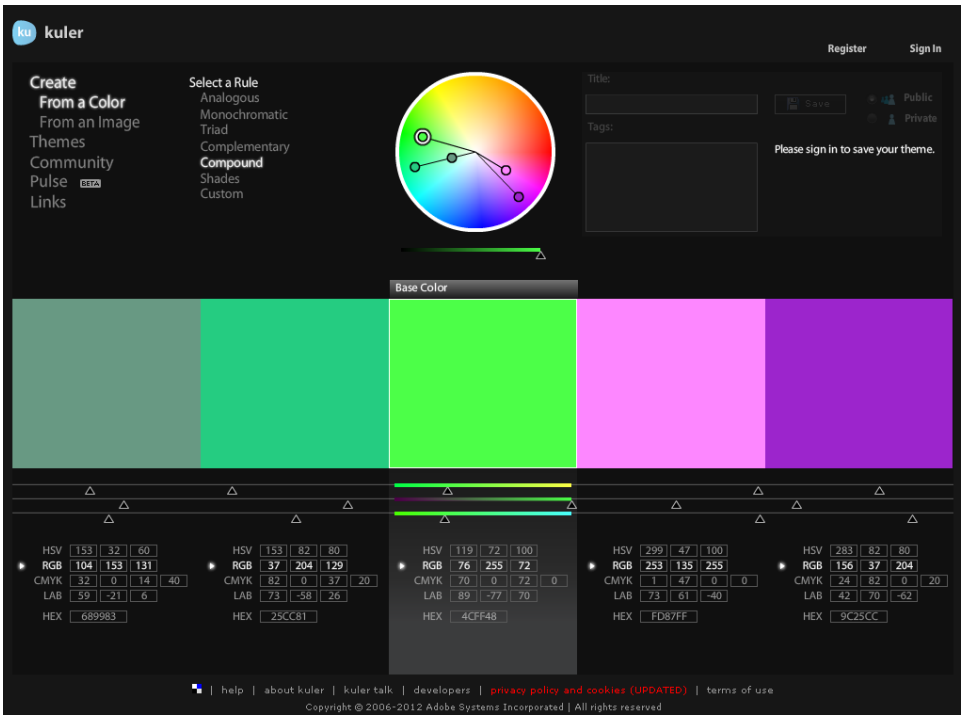


Abbildung 2.3: Mittels dem Onlinetool *kuler* von Adobe können, ausgehend von einer Hauptfarbe, verschiedene Farbschemen erstellt werden.

Auswahl der Schriften

Abgerundet wird diese Sammlung durch die Vorauswahl passender Schriftarten für Überschriften sowie den Fließtext. Natürlich sind diese Schriften nur Anhaltspunkte für die Gestaltung. Sich vorab Gedanken zu machen, erleichtert die Umsetzung der grafischen Vorlage.

2.3.3 Farben auswählen

Ausgehend vom Corporate Design ergeben sich meist ein bis zwei Hauptfarben. Diese kann man sich zu Nutze machen, um abgestufte oder kontrastreiche Farbschemen zu erstellen. Es sind verschiedene Hilfestellungen für die Erstellung von Farbschemen online verfügbar. Beispielsweise können mit *kuler*[28] von Adobe, siehe Abbildung 2.3, ausgehend von einer Hauptfarbe, verschiedene Farbschemen erstellt werden.

[28] http://kuler.adobe.com

2.3.4 Mockup

Bevor man mit der Gestaltung der Website richtig beginnen kann, ist es essenziell, sich um die Struktur der Webanwendung Gedanken zu machen. Welche Inhalte präsentieren sich wie und auf welcher Seite? Wie gestaltet sich das Menü, damit der Benutzer die richtigen Informationen schnell findet?

Sitemap

In diesem Schritt wird die Struktur der Webanwendung erstellt. Die verschiedenen Bereiche der Anwendung werden definiert und Abhängigkeiten erstellt. Dieser Schritt ist wichtig um abzugrenzen, welche Inhalte wichtig und welche nicht Teil der Anwendung sind. Zusätzlich erfährt man in diesem Schritt, welche Elemente in welchen Bereichen zu erstellen sind. Mit einer fertigen Sitemap kann abgegrenzt werden, welche Bereiche sich optisch ähneln und welche eigene Charakteristika aufweisen. Das führt zum nächsten Schritt, dem Erstellen einer groben Skizze.

Layout-Konzept

Ausgehend von der Sitemap definiert man verschiedene Layouts für die einzelnen Bereiche. Einige Layouts können für mehrere Bereiche ihre Gültigkeit haben, andere Bereiche müssen ein eigenes Layout aufweisen. Das Layout besteht wiederum aus Inhaltsbereichen, die jeweils mit Text oder Medieninhalten befüllt werden. Dieses Konzept zeigt lediglich eine grobe Skizze über Positionierung und Art der Elemente auf der jeweiligen Seite. Ein Beispiel findet sich in Abbildung 2.4.

2.4 Weitere Techniken

In diesem Abschnitt werden weitere, für die Umsetzung des eigenen Ansatzes relevante Tools und Techniken vorgestellt.

2.4.1 LESS

LESS[29] ist eine dynamische Programmiersprache die validen *CSS-Code* generiert. Die Sprache ist sehr an *CSS* angelehnt. Der Quelltext der geschrieben wird, ist erweiterter *CSS-Code*. Das hat den Vorteil, dass valider *CSS-Code* eins zu eins in eine *LESS*-Datei übernommen werden kann. Es ist eine logische Schicht, die über dem *CSS-Code* liegt und ermöglicht somit einige Mechanismen, die in der objektorientierten Programmierung angewendet werden. Es werden Variablen, Verschachtelung, Mixins, Funktionen und Operatoren als Mechanismen zur Verfügung gestellt.

[29] http://lesscss.org/

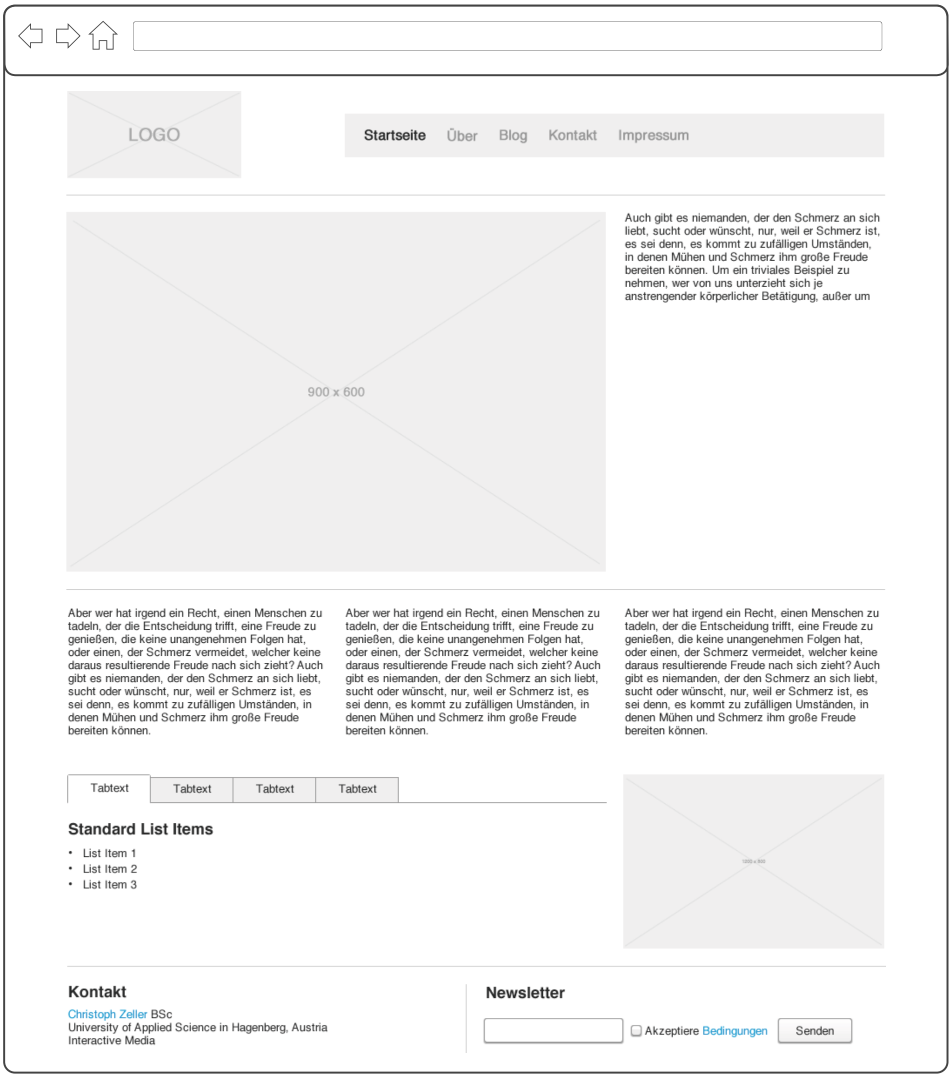


Abbildung 2.4: Ein Mockup einer Startseite als Teil des Layout-Konzepts.

Beim Kompilieren wird valider *CSS-Code* generiert. Dem Entwickler bieten sich mehrere Möglichkeiten *LESS* einzusetzen. Es kann einerseits zur Laufzeit client- bzw. serverseitig kompiliert werden oder andererseits über diverse Tools vorkompiliert werden. Beim Vorkompilieren müssen die generierten *CSS*-Dateien extra auf den Webserver geladen werden.

Folgendes Beispiel zeigt eine *LESS*-Definition eines Mixin um den Radius von abgerundeten Ecken zu definieren. Dabei ist die Übergabe-Variable mit dem Wert `0.4em` vordefiniert.

```
.border-radius (@radius: .4em) {
  -webkit-border-radius: @radius; /* Chrome, Safari */
```

```
3   -moz-border-radius: @radius; /* Mozilla */
4   border-radius: @radius; /* W3C CSS3 Deklaration */
5 }
```

Die Implementierung der oben angeführten Definition erfolgt durch den Funktionsaufruf `.border-radius`. Wie aus folgendem Beispiel ersichtlich, wird die Übergabevariable mit dem Wert `0.6em` überschrieben.

```
1 .box {
2   .border-radius(.6em);
3 }
```

Der daraus kompilierte *CSS-Code* wird in folgendem Beispiel ausgegeben.

```
1 .box {
2   -webkit-border-radius: .6em; /* Chrome, Safari */
3   -moz-border-radius: .6em; /* Mozilla */
4   border-radius: .6em; /* W3C CSS3 Deklaration */
5 }
```

2.4.2 Browserkompatibilität

Ein klassischer Arbeitsschritt beim Entwickeln einer Anwendung für das Web ist das Optimieren für einzelne Browser. Es ist immer noch so, dass einzelne Browserhersteller die Standards, die vom *W3C* definiert werden, nicht richtig interpretieren und somit sogenannte Browserweichen für die Unterstützung einzelner Browser implementiert werden müssen.

JavaScript Polyfills

Um die neuen Elemente, die *HTML5* bereitstellt sorglos verwenden zu können, müssen sie älteren Browsern wie beispielsweise *Microsoft Internet Explorer 6* erst bekannt gemacht werden. Diese Aufgabe lösen sogenannte *Polyfills*, indem die Funktionalität eines *HTML5* Features mit Hilfe von *JavaScript* nachgebildet wird. Ein Beispiel dazu findet man [3, S. 75].

Vendor-Prefix

Um nicht-standardisierte *CSS*-Anweisungen für einzelne Browser vorzeitig zur Verfügung zu stellen, können mit so genannten *vendor-prefixes* die einzelnen *Render-Engines*[30] der Browser direkt angesprochen werden. Für die Browser *Google Chrome*[31] und Apple Safari[32] wird beispielsweise `-webkit-` zum ansprechen der *Render-Engine* verwendet. Weitere Informationen über *vendor-prefixes* finden Sie in [1, S. 11–14].

[30] http://de.wikipedia.org/wiki/HTML-Rendering
[31] http://www.google.com/intl/de/chrome/browser/
[32] http://www.apple.com/de/safari/

CSS-Reset, Normalisierung

Jeder Browser hat ein Standard *Stylesheet*, das die Anzeige der einzelnen *HTML*-Elemente grundlegend definiert. Leider gibt es dafür keinen Industriestandard. Jeder Hersteller definiert die Elemente nach eigenen Vorstellungen. Somit ergeben sich in der Darstellung unterschiedlicher Browser teilweise sehr große Differenzen. Um diesen unterschiedlichen Darstellungsformen verschiedener Browser entgegenzuwirken, hat sich in der Webentwicklung eine Methode durchgesetzt, die alle Standard-Stil-Informationen verschiedener Browser auf einen gleichen Nenner bringt. Hierbei werden mittels einem *Stylesheet*, das am Anfang aller Stildefinitionen eingebunden wird, alle *HTML*-Elemente auf einheitliche Werte zurückgesetzt. Damit werden die herstellerspezifischen Formatvorlagen überschrieben und eine einheitliche Basis ist für die weitere Arbeit gegeben.

Die Normalisierung unterscheidet sich vom normalen *CSS-Reset* dadurch, dass bei dieser Methode direkt die gewünschten Werte aller Elemente definiert werden und der Webentwickler erspart sich damit den Zwischenschritt alles auf Null zu setzen. Zusätzlich werden hier alle typischen Fehler von verschiedenen Webbrowsern behoben und so eine geeignete Grundvoraussetzung für die weitere Arbeit geschaffen.

Kapitel 3

Verwandte Ansätze

In der heutigen Zeit ist das Entwickeln einer Webanwendung zu einer herausfordernde Aufgabe geworden. Es gibt viele neue Techniken, die es dem Entwickler ermöglichen, Webanwendungen mit flexiblen Benutzeroberflächen für verschiedene Endgeräte zu implementieren. Die Entwicklung einer Webanwendung, die plattform- und geräteübergreifend das gleiche Benutzererlebnis bietet, ist ein schwieriges Unterfangen. Der Entwickler ist mit verschiedenen Hindernissen konfrontiert. Es gibt unterschiedliche Browser, in denen die Webanwendung lauffähig sein soll, diese sind wiederum auf verschiedenen Geräten installiert. Zusätzlich wird die Entwicklung dadurch erschwert, dass verschiedene Auflösungen der jeweiligen Ausgabegeräte unterstützt werden sollen. Neue Eingabegeräte wie *touch-screens* erfordern die Auswahl der richtigen Elemente, erweitern dafür aber den Interaktionsspielraum. Um nicht bei jeder Webanwendung von ganz vorne beginnen zu müssen, greifen viele Webentwickler auf sogenannte *CSS-Frameworks* oder *libraries* zurück. Ob bei einem *CSS-Framework* wirklich von einem *Framework* gesprochen werden darf, ist laut Definition[1] nicht klar eruierbar.

CSS-Frameworks bieten Webentwicklern eine Grundlage, auf die ihre eigene Webanwendung aufbauen kann. Ein *CSS-Framework* stellt meist ein Layoutsystem sowie grundlegende Formatdefinitionen von einzelnen *HTML*-Elementen zur Verfügung. Der große Vorteil von *CSS-Frameworks* ist, dass diese für mehrere Browser, Geräte sowie Betriebssysteme getestet werden und somit die Implementierung von Browserweichen[2] auf das Mindeste reduziert wird. Unterschiede zwischen verschiedenen *CSS-Frameworks* liegen meist im Detail, zum Beispiel in der Umsetzung einzelner Anforderungen. In diesem Kapitel werden einige bekannte und viel genutzte *CSS-Frameworks* unter die Lupe genommen und deren Vor- und Nachteile aufgezeigt.

[1] http://de.wikipedia.org/wiki/Framework
[2] http://de.wikipedia.org/wiki/Browserweiche

3.1 Bootstrap

Bootstrap[3] ist ein Frontend-Baukasten-System und wurde ursprünglich von Twitter für die interne Oberflächenentwicklung der Analyse- und Verwaltungswerkzeuge entwickelt. Mittlerweile ist es auf *GitHub*[4] veröffentlicht und der Quellcode ist mittels *Apache License v2.0*[5] sowie die Dokumentation mittels *CC BY 3.0*[6] lizensiert. Es bietet auf *HTML* und *CSS* basierende Gestaltungsvorlagen, ein flexibles *Grid*-System, eine große Auswahl an Oberflächengestaltungselementen und optionale *JavaScript*-Erweiterungen für die Entwicklung von Webanwendungen.

3.1.1 Aufbau und Funktionen

Bootstrap ist ein modular aufgebautes System, das im Kern aus einzelnen *LESS-Stylesheets* besteht, welche die einzelnen Komponenten bilden. Ein zentrales *Stylesheet* inkludiert die einzelnen Komponenten und macht diese verfügbar. Mittels der *LESS*-Struktur hat der Entwickler einige Möglichkeiten auf die Gestaltung der Elemente Einfluss zu nehmen. Einerseits kann dies über ein zentral bereitgestelltes *Stylesheet* erfolgen, wo mittels Variablen, Farben und Schriften definiert sind und andererseits können vorhandene Klassen in einem eigenen *Stylesheet* überschrieben werden.

Die Verwendung von *LESS* als *Stylesheetsprache* ermöglicht den Einsatz von Variablen, Funktionen und Operatoren, verschachtelten Selektoren sowie von sogenannten Mixins. Mixins verwenden Klassen innerhalb anderer Klassen mit der Option, diese zu parametrisieren. Dadurch wird duplizierter Code vermieden und die Wartbarkeit der *Stylesheets* erhöht. Ein Nachteil ist die fehlende Unterstützung durch Webbrowser. Deshalb ist es erforderlich, die *Stylesheets* manuell, durch den Server oder clientseitig mit Hilfe von *Javascript* in reguläres *CSS* zu kompilieren.

3.1.2 Layout Funktionalität

Bootstrap ist mit einem 12-spaltigen *Grid*-System ausgestattet, welches standardmäßig 940px breit ist. Als Alternative bietet *Bootstrap* das gleiche *Grid*-System mit flexibler Breite an. Für beide Fälle bietet es vier Variationen im Sinne des *Responsive Designs*. Dabei passt sich die Breite einer Spalte automatisch an die Breite des *Viewports* an. Eine Optimierung für Mobiltelefone, *Tablets* sowie Desktopmonitore ist somit gegeben.

[3] http://twitter.github.com/bootstrap/
[4] https://github.com
[5] http://www.apache.org/licenses/LICENSE-2.0
[6] http://creativecommons.org/licenses/by/3.0/

3.1.3 Grundlegende Stildefinition

Bootstrap enthält eine Reihe von *Stylesheets*, welche grundlegende Stildefinitionen für alle wichtigen *HTML*-Elemente enthalten. Als *CSS*-Reset oder anders gesagt zur Normalisierung der Basis wird eine abgewandelte Form von *normalize.css*[7] verwendet. Diese gewährleistet ein browser- und systemübergreifend einheitliches, modernes Erscheinungsbild für die Textformatierung, für Tabellen sowie Formularelemente.

Zusätzlich zu den regulären *HTML*-Elementen bietet *Bootstrap* weitere Oberflächenelemente an. Es werden beispielsweise Navigationselemente in vertikaler oder horizontaler Ausrichtung, Buttons mit erweiterter Funktionalität, Formatierungen für Hinweismeldungen und viele weitere Oberflächenelemente angeboten.

3.1.4 Weitere Funktionalitäten

Abgerundet wird das System mit verschiedenen *Plugins*, die auf Basis des *JavaScript-Frameworks jQuery*[8] implementiert sind. Sie erweitern die Funktionen der Oberflächenelemente und bieten außerdem zusätzliche *User-Interface-Elemente* wie zum Beispiel Dialogfenster.

3.1.5 Verwendung

Der Webentwickler hat die Möglichkeit, eine kompilierte Version direkt von der *Bootstrap* Website oder die Version für Entwickler von *GitHub* herunterzuladen. Seit der Version 2.0 gibt es zusätzlich die Möglichkeit, über ein Formular die benötigten Komponenten auszuwählen und das System zu individualisieren.

3.2 YAML

Yet Another Multicolumn Layout[9] (kurz *YAML*) ist ein modular aufgebautes *CSS-Framework* und bietet neben verschiedenen Layout-Varianten und grundlegenden Stildefinitionen auch eine Reihe von erweiterten Oberflächenelementen an. Die erste Version von *YAML* wurde bereits im Jahr 2005 veröffentlicht und wurde somit jahrelang getestet und verbessert. Mit der aktuellen Version 4 wurde die gesamte Struktur auf *HTML5* und *CSS3* umgestellt. Diese ermöglicht laut Entwickler eine Kompatibilität zu allen gängigen Browsern sowie ein flexibles Layoutsystem um geräte- sowie plattformunabhängige Webanwendungen zu erstellen.

[7] https://github.com/necolas/normalize.css
[8] http://jquery.com
[9] http://www.yaml.de

3.2.1 Aufbau und Funktionen

YAML ist modular aufgebaut und besteht aus einer Reihe verschiedener *CSS-Stylesheets*, welche die einzelnen Komponenten definieren. Es stellt vier verschiedene Gestaltungsvorlagen als Einstieg für die eigene Umsetzung des Webentwicklers zur Verfügung. Die Kernfunktionalitäten sowie die Erweiterungen werden innerhalb des *Layout-Stylesheets* zu Beginn durch die `@import` Anweisung inkludiert, was in der Praxis zu vielen Serveranfragen führt. Zusätzlich werden einige Komponenten, welche die Oberflächenelemente erweitern, angeboten.

Entwicklern steht eine ausführliche Dokumentation zur Verfügung. Darin werden einzelne Elemente und Komponenten beschrieben und mit Beispielen hinterlegt. Die Browserkompatibilität erreicht *YAML*, indem es die browserspezifischen Stile normalisiert und zusätzlich eine Version speziell für den *Internet Explorer* bereit stellt. Um ältere Browser fit für den Einsatz von *HTML5*-Elementen zu machen, greift das *Framework* auf *html5shiv*[10] zurück.

3.2.2 Layout-Funktionalität

YAML stellt verschiedene Layout-Varianten zur Verfügung. Neben einem flexiblen *Grid-System* bietet es auch die Möglichkeit, das Layout spaltenbasiert zu implementieren. Das *Grid-System* verfügt über zwölf Spalten und ist mittels Prozentwerten implementiert. Die Klassendefinitionen des Layouts sind im *Basis-Stylesheet* implementiert. Einzig für das spaltenbasierte Layout bietet das *Framework* die Anpassung an verschiedene Auflösungen in der Standardeinstellung an. Um auch für das *Grid-System* Anpassungen im Sinne von *Responsive Design* zu erreichen, muss der Webentwickler selbst weitere Definitionen implementieren. *YAML* stellt dazu den Quellcode in der Dokumentation[11] bereit.

3.2.3 Grundlegende Stildefinition

Im *Basis-Stylesheet* des *Frameworks* werden keine grundlegenden Stile definiert, lediglich die Normalisierung des browserinternen *Stylesheets* wird hier durchgeführt. Erst durch das Einbinden eines weiteren *Stylesheets* wird eine grundlegende Stildefinition durchgeführt. Dieses Vorgehen ist bewusst gewählt worden, denn der Entwickler soll die anwendungsspezifischen Anpassungen direkt in den dafür vorgesehenen *Stylesheets* vornehmen können.

[10] http://code.google.com/p/html5shim/
[11] http://www.yaml.de/docs/index.html#yaml-grids

3.2.4 Weitere Funktionalitäten

Weitere Funktionalitäten erreicht das *CSS-Framework* über diverse *Add-Ons*. Ein Navigationsmodul stellt eine vertikale oder horizontale Navigation bereit. Ein *jQuery-Plugin*, beinhaltet zusätzlich eine *Tab-Navigation*[12]. Ein rechts-nach-links Struktur-Modul ermöglicht arabische und hebräische Schriften und Mikroformate repräsentieren Personen, Events oder Adressen. Der große Vorteil von *YAML* ist, dass es eine Vielzahl fertiger Vorlagen für diverse *Content-Management-Systeme*[13] gibt.

[12] https://github.com/ginader/Accessible-Tabs
[13] http://www.yaml.de/resources.html

Kapitel 4

Eigener Ansatz

Dieses Kapitel gibt einen Einblick in die eigenen Überlegungen zum Thema *Responsive Webdesign CSS Framework*. Ziel ist es, Webentwickler dabei zu unterstützen, schneller und effizienter zu einer geräte- und plattformunabhängigen Webanwendung zu gelangen. Im Vordergrund stehen hierbei die einfache Verwendung, die Erweiterbarkeit und maximale Flexibilität für den Entwickler. Zudem soll für den Endnutzer höchstmögliche Benutzerfreundlichkeit und Barrierefreiheit erreicht werden.

Das übergeordnete Ziel dieser Arbeit ist es, alle Elemente, die in Webanwendungen immer wieder verwendet werden, automatisch für die jeweilige Plattform oder das jeweilige Gerät so anzupassen, dass das bestmögliche Nutzererlebnis erreicht wird.

4.1 Studie Mobile-Web

Was erwarten mobile Webanwender, die eine Anwendung bzw. einen Dienst über ein mobiles Endgerät nutzen? Dieser Frage geht eine Studie [21], die im Jahr 2011 von Compuware[1] aufgetragen wurde, nach. Die Ergebnisse dieser Studie sind im folgenden Abschnitt kurz angeführt:

- 71 % aller *Smartphonebenutzer* weltweit wünschen sich eine schnellere Ladezeit von Webanwendungen. Diese soll mindestens so schnell, oder schneller laden, als am Desktop. Die Hälfte der Befragten gibt an, dass Webanwendungen, die sie regelmäßig benutzen, auf deren mobilen Endgeräten langsamer seien als am Desktop.
- Fast 50 % der befragten Benutzer erwarten sich von einer Webanwendung, dass sie in weniger als drei Sekunden zur Verfügung steht. 74 % sind bereit, bis zu fünf Sekunden zu warten, bevor eine einzelne Seite geladen ist.

[1] http://www.compuware.com/d/release/592528/new-study-reveals-the-mobile-web-disappoints-global-consumers

- 57 % der befragten Benutzer hatten vergangenes Jahr Probleme mit einem mobilen Endgerät eine Website zu öffnen. Mehr als 80 % der befragten Web-Nutzer verwenden eine Webanwendung häufiger mobil, wenn das Verhalten schnell und zuverlässig ist.
- Mobile Anwender haben nicht viel Geduld. Wenn eine Website nicht gleich beim ersten Mal richtig funktioniert, sucht sich rund ein Drittel eine alternative Website eines Mitbewerbers.
- Wird von den mobilen Anwendern einmal eine schlechte Erfahrung auf einer mobilen Website oder Webanwendung gemacht, werden diese höchstens ein zweites Mal ausprobiert (von ca. der Hälfte der Befragten). Noch unwahrscheinlicher ist es, dass diese Websites oder Webanwendungen weiterempfohlen werden (57 % keine Weiterempfehlung).

Die Kernaussage dieser Studie besagt, dass eine mobile Webanwendung schnelle Ladezeiten sowie eine optimierte Darstellung für das Nutzererlebnis voraussetzt. Diese zwei Kriterien sind neben den Standardkriterien eines *CSS-Frameworks* wichtiger Bestandteil des eigenen Ansatzes. Folgende Anforderungen werden an das System gestellt:

- höchstmögliche Flexibilität für den Webentwickler,
- einfache Erweiterbarkeit durch den Webentwickler,
- hohe Performance für den Endnutzer,
- höchstmögliche *User-Experience* für den Endnutzer.

4.2 Aufbau

Der eigene Ansatz orientiert sich grundsätzlich an dem Prinzip der progressiven Verbesserung und unterstützt die Methoden von *Responsive Webdesign* und *Mobile First*. Um die größtmögliche Flexibilität für den Entwickler zu erreichen, ist das System so aufgebaut, dass sich einzelne Komponenten oder Module einfach austauschen oder verändern lassen. Das *Framework* besteht aus einer Reihe von *LESS-Stylesheets*, die alle Module und Komponenten beinhalten. Zusammengeführt werden diese über ein zentrales *Stylesheet*, welches die benötigten Komponenten inkludiert. Die einzelnen Komponenten erhalten über eigene *jQuery Plugins* zusätzliche Funktionalität. Dieser Aufbau ermöglicht dem Entwickler das Erstellen von unterschiedlichsten Webanwendungen.

Der Aufbau des *Frameworks* mittels *CSS-Preprocessor LESS* bringt einige Vorteile mit sich. Um den Workflow eines Webentwicklers zu unterstützen und die Flexibilität zu erhöhen bietet das *Framework* zwei globale *Stylesheets* an. Im ersten, *variables.less*, sind alle Variablen definiert, welche global über die gesamte Struktur verwendet werden können. Im zweiten, *mixins.less*, sind diverse Funktionen definiert, die wiederum in verschiedenen Klassen verwendet werden können. Das zentrale *Stylesheet*, *domino.less*, ver-

waltet und inkludiert alle weiteren *Stylesheets*. Dies bringt den Vorteil, dass nur ein einziges *Stylesheet* für die produktive Umgebung kompiliert wird. Der Entwickler entscheidet selbst, welche Module und Komponenten eingebunden werden und welche nicht. Das vermeidet unnötigen Ballast für den Produktiveinsatz.

Das *Framework* erwartet standardkonformen und validen *HTML5*-Quelltext. Es unterstützt alle neuen Elemente und empfiehlt, diese auch ordnungsgemäß einzusetzen: `<header>` als Kopfbereich, `<nav>` als Navigationsbereich und so weiter. Als Konvention sollte der Inhalt der Webanwendung in strukturierter Art und Weise vorliegen, bevor mit der Umsetzung begonnen wird. Das beschreibt Mark Boulton in seinem Artikel [13].

4.2.1 Kernfunktionalität

Um für alle modernen Webbrowser ein einheitliches Benutzererlebnis zu bieten, stellt das *Framework* einige Funktionalitäten bereit. Es bietet ein eigenes *Stylesheet* an, welches grundlegende Stildefinitionen für alle *HTML*-Elemente beinhaltet. In dieses *Stylesheet* fließt ebenso die Normalisierung ein. Mittels der Normalisierung werden die verschiedenen Darstellungsformen der Browser auf gleichen Nenner gebracht und die Differenzen bereinigt. Dies basiert auf dem Projekt[2] von Nicolas Gallagher und Jonathan Neal, welches den Vorteil hat, dass eine große Gruppe von Anwendern diese ebenfalls verwendet und somit Fehler schnell erkannt und behoben werden.

Ältere *IE*-Versionen[3] bis einschließlich Version 8 können mit den neuen Elementen, die in *HTML5* definiert wurden, nicht umgehen und ignorieren diese beim Rendern der Webseite. Um diesem Problem entgegenzuwirken können die neuen Elemente per *JavaScript* auch in diesen Browsern bekannt gemacht werden. *html5shiv*[4] ist ein frei verfügbares *JavaScript*-Projekt, das genau dieses Problem löst und in diesem *Framework* angewendet wird.

4.2.2 Layout

Ebenfalls zum Kern gehört, wenn auch optional und austauschbar, ein selbst entwickeltes Rastersystem, welches nach dem Grundsatz von *Mobile First* entworfen wurde. Das Standardlayout optimiert die Darstellung bei mobilen Endgeräten, wobei nach der Methode von *Responsive Design* diese automatisch für größere Monitore angepasst wird. Standardmäßig wird für Darstellungsbreiten von 480 Pixel, 768 Pixel, 1024 Pixel und 1200 Pixel jeweils eine optimierte Darstellungsvariante bereitgestellt. Diese kann einfach über die zentral verwalteten Variablen verändert werden. Dies ist eine Variante, die für mobile Endgeräte von *Apple* optimal funktioniert aber nicht auf jedes

[2] http://necolas.github.com/normalize.css/
[3] Microsoft Internet Explorer
[4] http://code.google.com/p/html5shiv/

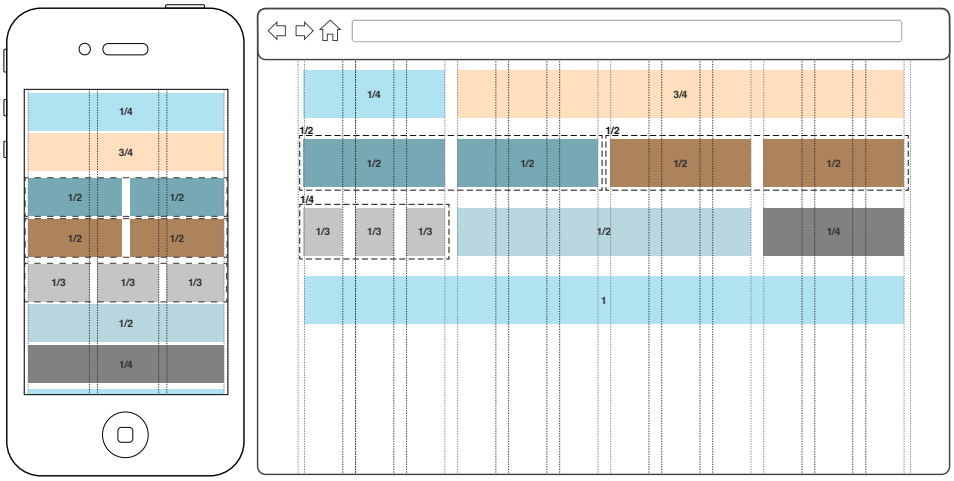


Abbildung 4.1: Rastersystem von *DOMINO* als Mockup. Links wird die Ansicht auf einem mobilen Endgerät mit einer Breite bis zu 480 Pixel dargestellt und rechts die Ansicht auf einem Desktop.

Projekt einfach angewendet werden sollte. Der jeweilige Webentwickler sollte sich über die Zielgruppe und deren Gegebenheiten, für welche die konkrete Webanwendung entwickelt wird, bewusst sein.

Der Raster besteht aus vier Spalten und kann individuell unterteilt werden. Eine Zeile kann jeweils aus vier Viertel, drei Drittel, zwei Halbe oder einem ganzen Element bestehen. Es können natürlich auch Variationen daraus gebildet werden. Um die Flexibilität noch zu steigern, können diese Zeilen und Spalten verschachtelt werden. Wie aus Abbildung 4.1 ersichtlich, wird es durch die Verschachtelung möglich, auch in der mobilen Version bis zu drei Spalten darzustellen. Diese Funktion unterscheidet sich von den meisten bekannten *Grid*-Systemen, die zwar durch mehrere Spalten für Desktops flexiblere Layouts ermöglichen, jedoch bei der mobilen Variante immer auf eine einspaltige Darstellung reduzieren.

4.3 Komponenten

Die Komponenten erweitern das System um wiederverwendbare *User-Interface*-Elemente. Im folgenden Abschnitt wird der Aufbau dieser Elemente beschrieben. Bei allen Elementen wurde bei der Konzipierung auf die Einsetzbarkeit bei unterschiedlichen Bildschirmgrößen großer Wert gelegt. Ausgehend von den kleineren Displays, wie sie bei *Smartphones* zum Einsatz kommen, wurden die Elemente zusätzlich für die Darstellung auf größeren Monitoren optimiert. Die Stile werden jeweils mit eigenen *LESS-Stylesheets*

definiert. Die Funktionalität der einzelnen Elemente wird mittels *jQuery-Plugin* implementiert. Durch den modularen Aufbau des Systems können einzelne Elemente einfach geändert, ausgetauscht oder weggelassen werden, ohne die Funktion anderer Teile zu beeinflussen.

4.3.1 Navigation

Einer der wichtigsten Elemente in einer Webanwendung ist – neben dem Inhalt – die Navigation. Diese ermöglicht es dem Benutzer einfach zu allen Informationen zu gelangen. Eine Navigation spiegelt die Struktur einer Webanwendung wider, das heißt, dass eine umfangreiche Webanwendung meist auch eine umfangreiche Navigationsstruktur aufweist. Bis heute ist es üblich, die Navigation einer Webanwendung oben als Zeile oder links in einer Spalte anzuordnen. Oft werden diese beiden Varianten auch vermischt als Haupt- und Unternavigation. Zusätzlich haben Navigationselemente oft mehrere Ebenen, von grob – Hauptebene zu fein – eine oder mehrere Unterebenen, um dem Benutzer die Filterung der Inhalte zu erleichtern.

Eine Navigationsstruktur soll platzsparend aber auch übersichtlich für den Benutzer aufgebaut sein. Ein Desktop bietet genügend Platz, um diesen mit verschiedenen Darstellungselementen vollzupacken. Es bietet sich einiges an Spielraum für die optimale Navigation für unterschiedliche Projekte an. Bei mobilen Endgeräten sieht die Sache ganz anders aus. Diese stellen einen eingeschränkten Raum zur Verfügung. Die Navigationsstruktur soll für den Benutzer aber trotzdem übersichtlich und leicht erreichbar bleiben. Diese Problematik wurde auch schon von vielen anderen diskutiert und es wurden Lösungsansätze dafür erstellt. Viele von den vorhandenen *Responsive* Webseiten basieren auf einigen wenigen Anpassungsmustern von Navigationselementen zwischen *Smartphone* und Desktop. Diese wurden von Luke Wroblewski in einem Artikel[22] gut zusammengefasst.

Eine weit verbreitete Methode für die Navigation auf mobilen Endgeräten ist, die Navigation auszublenden und per Klick mittels Animation in das Anzeigefenster zu fließen. Der Vorteil bei dieser Methode ergibt sich daraus, dass die Navigation immer nur einen Klick entfernt ist. Durch die Animation nimmt der Benutzer diese als Bestandteil der Webanwendung wahr, als ob diese nur aus dem Anzeigebereich gerutscht wäre. Um die *User-Experience* auf mobilen Geräten noch zu steigern, kann die Navigation optional mittels Fingerwisch ein- und ausgeblendet werden. Wie in Abbildung 4.2 gut ersichtlich, wird diese Variante für mobile Endgeräte in diesem System bereitgestellt. Für die Darstellung am Desktop kann der Webentwickler über das *HTML-Markup* bestimmen, ob die Navigation horizontal oder vertikal dargestellt wird.

Abbildung 4.2: Mockup der Komponente Navigation. Links wird die Navigation bei mobilen Geräten dargestellt und rechts die Anzeige der Navigation bei Desktops. Oben sieht man den Ausgangszustand ohne Interaktion des Benutzers und unten links zeigt es das Bild nachdem die Navigation eingeblendet wurde bzw. rechts das aufgeklappte Untermenü nach dem Überfahren mit der Maus.

4.3.2 Accordion zu Tabs

Ein bekannter Ansatz im *User-Interface*-Design ist, Inhaltsbereiche mittels sogenannten *Tabs* zu unterteilen, siehe [15]. Inhalte sind hierbei in verschiedene Bereiche aufgeteilt und es erfolgt jeweils nur die Anzeige eines einzigen Bereichs. Der Benutzer kann zwischen den verschiedenen Bereichen hin- und herschalten, indem er auf einen *Tab* klickt. Dieses *User-Interface*-Element kommt aus dem Bereich der Desktop-Anwendungen und wird zum Beispiel bei Webbrowsern eingesetzt, um zwischen Webseiten zu wechseln. Bei diesem Element ergibt sicht das Problem, dass bei der Darstellung auf kleinen Touchdisplays viel Platz für die *Tabs* benötigt wird, um diese einfach mit

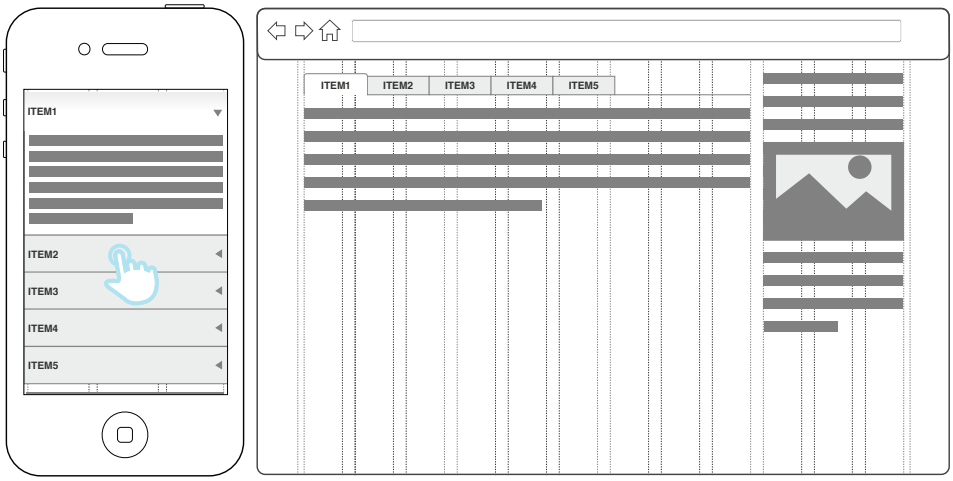


Abbildung 4.3: Mockup der Komponente *Accordion zu Tabs*. Links ist die Darstellung auf einem *Smartphone* zu sehen und rechts die Darstellung auf einem größeren Bildschirm.

dem Finger zu aktivieren. Bei der Recherche für die Umsetzung wurde ein Element gefunden, das ähnliche Ziele erreicht. Es handelt sich um ein sogenanntes *Accordion*, welches mittels einzelner Schaltflächen, Inhalte ein- bzw. ausblenden kann. Dieses Element funktioniert für kleinere Displays besser als die *Tabs*.

Es wurde also eine Komponente konzipiert, die es dem Entwickler ermöglicht mit identem Quelltext die Vorteile dieser zwei *User-Interface*-Elemente miteinander zu kombinieren. Wie in Abbildung 4.3 ersichtlich, optimiert diese Komponente automatisch die Anzeige aufgrund der Displaygröße des Benutzers.

4.3.3 Carousel

Ein Carousel ist ein weit verbreitetes *User-Interface*-Element im Webbereich. Dieses schaltet automatisch eine Reihe von Inhalten, meist animiert für einen definierten Bereich, auf der Darstellungsfläche durch. Oft wird dieses Element für eine Bilderserie oder um neue bzw. besondere Inhalte einer Webanwednung hervorzuheben, verwendet. Ähnlich wie bei Tabs hat es den Vorteil, Inhalte platzsparend in einem definierten Bereich darzustellen.

Im *Framework* wird dieses Element als eigene Komponente bereitgestellt. Wie in Abbildung 4.4 ersichtlich, stellt diese Komponente für die Anzeige auf mobilen Endgeräten, je nach Verwendung im Hoch- bzw. Querformat, zwei verschiedene Darstellungstypen bereit.

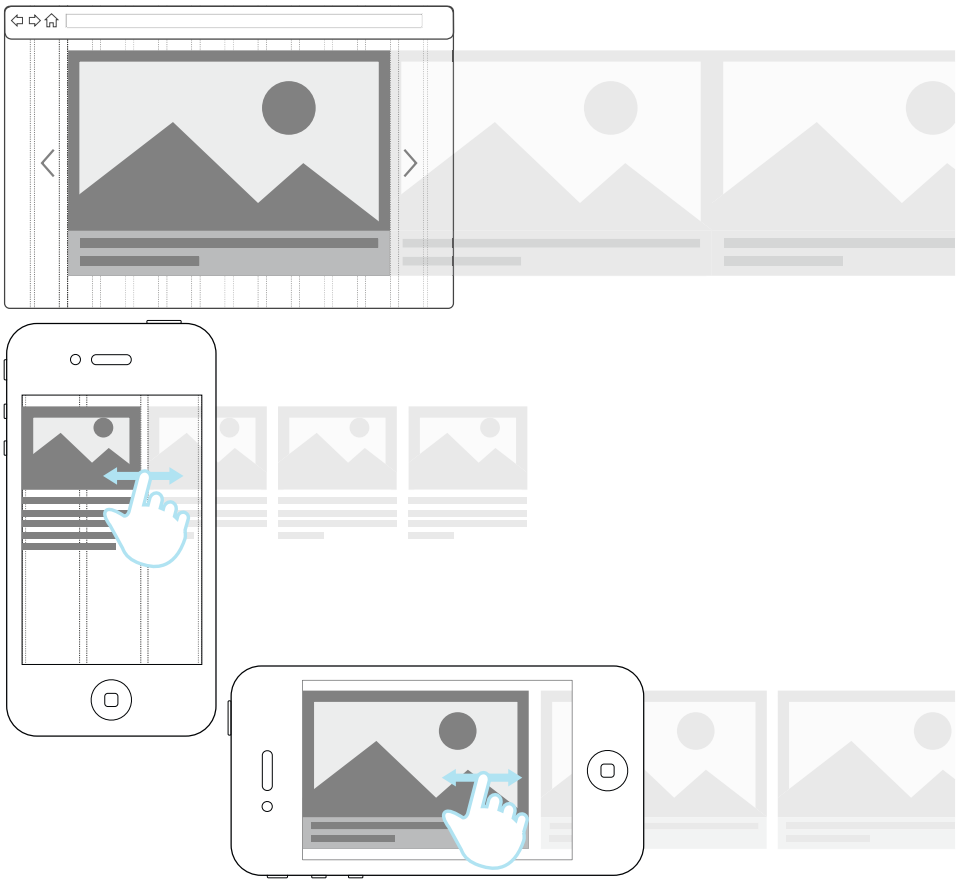


Abbildung 4.4: Verschiedene Anzeigetypen der Komponente *Carousel*.

4.3.4 Symbole

Symbole erleichtern das Mitteilen einer Information und unterstreichen den Sachverhalt des Inhalts. In diesem *Framework* kommt eine Symbolbibliothek in Form eines Schriftstils zum Einsatz. Das bringt zwei essentielle Vorteile mit sich. Erstens sind die Symbole Vektorgrafiken und können somit verlustfrei skaliert werden und zweitens wird die Bibliothek nur einmal geladen und steht dann zur Verfügung. Im Gegensatz zu anderen Varianten, wo jedes Symbol eine eigene Grafik ist und extra geladen werden muss. Es wurde auch eine Variante mittels SVG-Grafiken[5] in Kombination mit *CSS-Sprites*[6] angedacht, diese aber aufgrund der mangelnden Unterstützung des SVG-Standards diverser Browser wieder verworfen.

[5] http://www.w3.org/TR/SVG/
[6] http://css-tricks.com/css-sprites/

Abbildung 4.5: Diese Grafik zeigt eine Auswahl der insgesamt 116 verschiedenen Symbole, die über diese Komponente eingebunden werden können.

Über den externen Dienst *Fontello - icon fonts generator*[7] wurde eine große Auswahl an Symbolen für das *Framework* bereitgestellt, eine Auswahl zeigt Abbildung 4.5. Diese können unter Beachtung der beigefügten Lizenzbestimmungen von *Font-Designer* frei verwendet werden.

4.3.5 Formular

Diese Komponente erweitert das Basis-*Stylesheet* und bietet erweiterte Stile für Formulare an. Zusätzlich bietet es Klassen, die eine individuelle Layoutierung für die Formularelemente erlauben, beispielsweise zur sauberen Anordnung der *Labels*. Weiters wird die clientseitige Validierung sowie die neuen Attribute von *HTML5*-Formularen für Browser, die diese nicht von sich aus unterstützen, mittels einem *Polyfill*[8] nachgerüstet. Optional werden formulartypische *Interface*-Elemente wie zum Beispiel ein Kalender für das `date`-Feld oder ein Farbspektrum für das `color`-Feld für Browser, die diese nicht von sich aus unterstützen, mittels der *jQuery-UI-Library*[9] nachgerüstet.

4.3.6 Tabelle

Für die Ausgabe von Tabellen auf *Smartphones* reicht oft der Platz nicht aus, um diese übersichtlich darzustellen. Das *Framework* bietet mittels dem Tabellenmodul eine optimierte Variante für kleinere Ausgabegeräte an. Wie in Abbildung 4.6 ersichtlich, wird jede Zeile als zweispaltige Tabellenansicht

[7] http://fontello.com/
[8] http://www.matiasmancini.com.ar/jquery-plugin-ajax-form-validation-html5.html
[9] http://jqueryui.com/

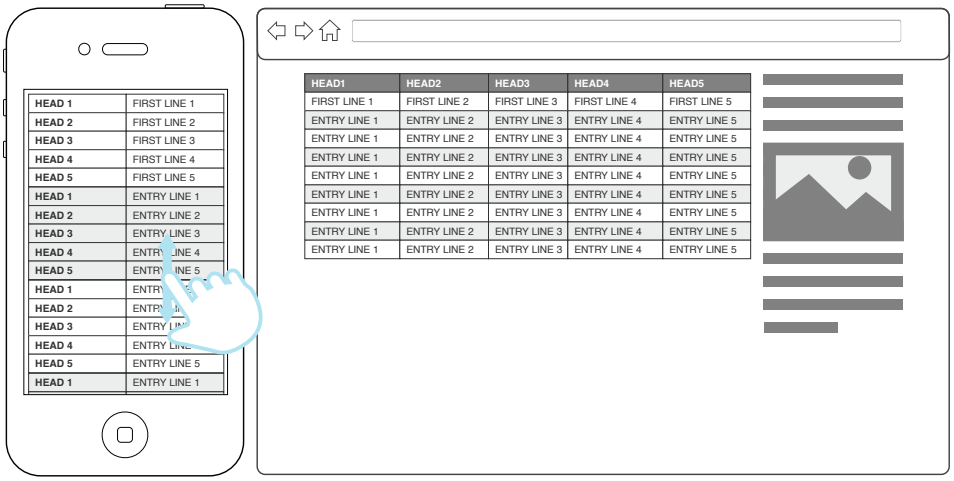


Abbildung 4.6: Mockup zur Darstellung von Tabellen mit verschiedenen *Viewports*, einerseits von einem *Smartphone* und andererseits vom Desktop.

dargestellt. Links wird die Überschrift und rechts der Inhalt angezeigt. Somit lassen sich auch größere Tabellen sauber und übersichtlich auf kleinen Ausgabegeräten darstellen.

Kapitel 5

Implementierung

Im Zuge dieser Arbeit wurde ein *CSS-Framework* auf Basis von *Responsive Design* entwickelt. Wie das System im Detail funktioniert und aufgebaut ist, wird im folgenden Kapitel erläutert.

5.1 Aufbau

Wie in Abschnitt 4.2 beschrieben, besteht das System aus einer Reihe von *LESS-Stylesheets* und *jQuery*-Plugins. Diese implementieren die einzelnen Module und Komponenten. Dem Entwickler steht es frei, Komponenten nach seinen eigenen Wünschen anzupassen oder auszutauschen. Dieser Aufbau dient der Flexibilität und Erweiterbarkeit, die der Entwickler für die Umsetzung individueller Webanwendungen benötigt. Abbildung 5.1 zeigt den Aufbau des gesamten *Frameworks*.

5.1.1 Modul

Ein Modul beschreibt im Zusammenhang mit dem *Framework* einen einzelnen Teil einer Funktionalität. Es ist eine eigene Datei im Filesystem. Bei einer Komponente ist ein Modul beispielsweise das eigene *Stylesheet*, welches alle Stil-Informationen beinhaltet und ein weiteres Modul ein *jQuery-Plugin*, welches logische Operatoren für die Komponente bereitstellt und die Funktion damit erweitert.

5.1.2 Komponente

Eine Komponente ist eine Funktionalität die im Framework zur Verfügung gestellt wird. Diese kann aus einem *Stylesheet* oder aus einer Kombination von *Stylesheet* und *jQuery*-Plugin bestehen. Ein Beispiel für eine Komponente ist die Navigation. Diese besteht wiederum aus einem Stil-Modul und einem *jQuery*-Modul. Bei einer Komponente kann es vorkommen, dass diese

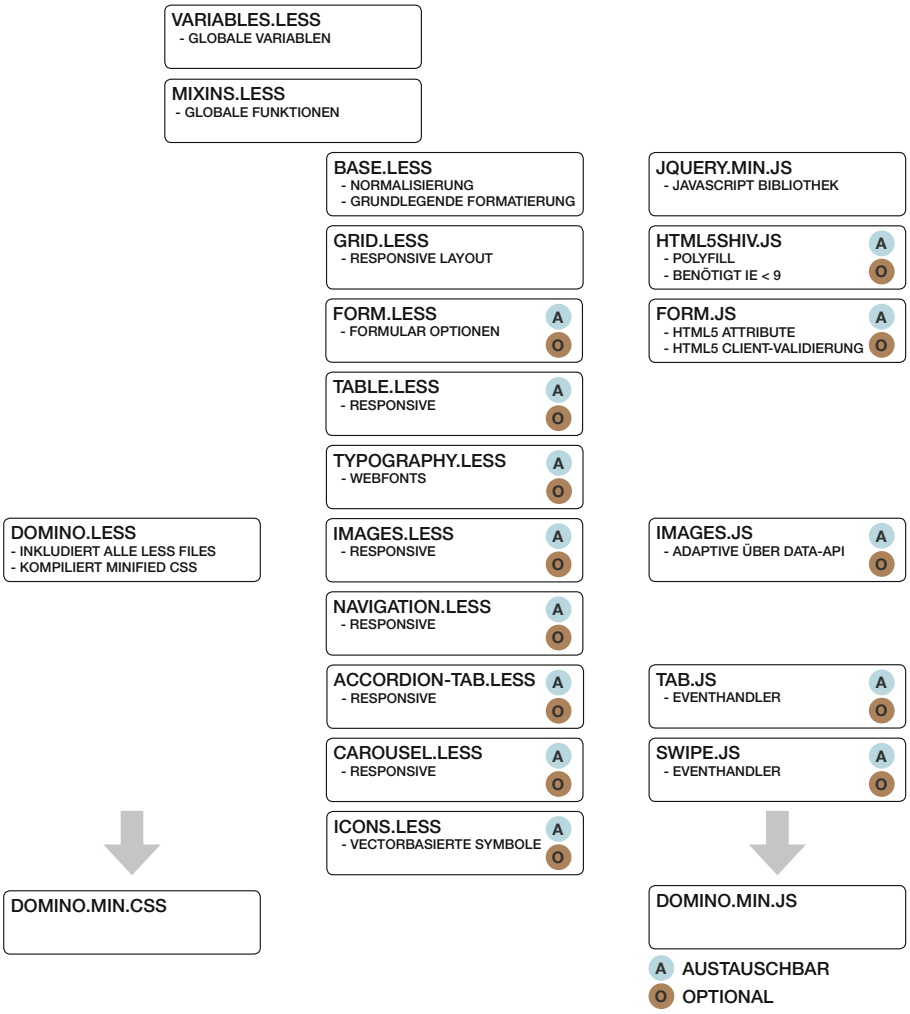


Abbildung 5.1: Aufbau des entwickelten *CSS-Frameworks DOMINO*. Es werden alle Module und Komponenten schematisch dargestellt.

nur mit dem Einsatz von mehreren Modulen funktioniert. Dies wird bei der Beschreibung der einzelnen Komponenten noch genauer erklärt.

5.1.3 Kern-Module

domino.less:

Dieses *Stylesheet* bildet das zentrale Element des *Frameworks*. Wie aus Programm 5.1 ersichtlich, werden hier alle Variablen bzw. Funktionen, alle Kernfunktionalitäten sowie alle Stil-Module zusammengeführt. Dies bildet die Grundlage für ein einzelnes, kompiliertes *CSS-Stylesheet*, das in einer Produktivumgebung zum Einsatz kommen kann. Der Entwickler hat hier die

Programm 5.1: `domino.less` ist ein zentrales *Less-Stylesheet*, welches alle weiteren Stil-Module inkludiert.

```
1 // less varibles, mixins
2 @import url("./variables.less");
3 @import url("./mixins.less");
4
5 // domino core
6 @import url("./base.less");
7 @import url("./grid.less");
8
9 // components
10 @import url("./typography.less");
11 @import url("./navigation.less");
12 @import url("./accordion-tab.less");
13 @import url("./carousel.less");
14 @import url("./icons.less");
15 @import url("./form.less");
16 @import url("./table.less");
```

Möglichkeit eigene Module hinzuzufügen bzw. alle Module die nicht benötigt werden zu löschen, um diese nicht als unnötigen Ballast mit in das Stylesheet für die Produktivumgebung zu kompilieren. Diese Maßnahme bringt einerseits weniger *HTTP-Requests* und andererseits schnellere Ladezeit durch eine Komprimierung der Dateigröße. Das schont gleichzeitig die Bandbreite des Anwenders.

variables.less:

Dieses *Stylesheet* verwaltet alle globalen Variablen, die der Entwickler individuell für einzelne Projekte anpassen kann. Die definierten Variablen kann der Entwickler in jedem weiteren *Stylesheet* verwenden und er hat somit die Möglichkeit an einer zentralen Stelle Änderungen für das gesamte Projekt vorzunehmen. Standardmäßig sind hier alle Farben sowie Schriften definiert.

mixins.less:

In diesem *Stylesheet* werden dem Entwickler allgemeine Funktionen zur Verfügung gestellt. Diese Funktionen sind Vorlagen die wiederverwendet werden können. Bei den weiteren Modulen kommen diese Funktionen zum Einsatz. Der Entwickler hat die Möglichkeit diese Funktionen anzupassen oder zu erweitern und kann diese auch für seine eigenen Erweiterungen verwenden. Anhand eines einfachen Beispiels soll verdeutlicht werden, wie flexibel der Einsatz einer Funktion aussehen kann.

Im folgenden Beispiel wird mittels *LESS* eine Funktion definiert, die in weiterer Folge eingebunden werden kann.

```
1 .border-radius (@radius: .4em) {
2   -webkit-border-radius: @radius; /* Chrome, Safari */
3   -moz-border-radius: @radius; /* Mozilla */
4   border-radius: @radius; /* W3C CSS3 Deklaration */
5 }
```

Diese Funktion verleiht einem Block-Element abgerundete Ecken mit einer Größe von 0,4em. `@radius: .4em` ist ein optionaler Parameter, welcher der Funktion mitgegeben werden kann. Der Parameter wird innerhalb der Funktion mit dem definierten Wert belegt. Der Entwickler kann nun diese Funktion in seinen eigenen *Stylesheets* mittels `.border-radius(.6em);` einbinden. Beim Kompilieren wird überall wo diese Funktion eingebunden wird, diese als valides *CSS* aufgelöst, wie im folgenden Beispiel ersichtlich ist.

```
1 .box {
2   -webkit-border-radius: .6em; /* Chrome, Safari */
3   -moz-border-radius: .6em; /* Mozilla */
4   border-radius: .6em; /* W3C CSS3 Deklaration */
5 }
```

base.less:

Mittels dem Basis-*Stylesheet* wird eine Normalisierung üblicher Browser-*Bugs*, sowie eine grundlegende Stildefinition der *HTML*-Elemente durchgeführt. Für Anpassungen oder individuelle Stil-Definitionen sollte der Webentwickler ein eigenes *LESS-Stylesheet* und die gewünschten Stil-Definitionen überschreiben. Somit gehen bei Updates des *Frameworks* keine essentiellen Konfigurationen verloren.

grid.less

Das Layout-Modul ist als vierspaltiges Rastersystem umgesetzt. Die Standarddefinition bezieht sich auf Endgeräte mit kleinem Bildschirm, welche eine Maximalbreite von 480 Pixel haben. Bei diesen sind alle Definitionen relativ zu der *Viewport*-Breite gesetzt. Als Container für die verschiedenen Spaltenbereiche dient ein eigenes Block-Element, welches auch das *Float*-Handling übernimmt. Innerhalb des Containers kann das Raster in sechs verschiedene Bereiche unterteilt werden. Diese Bereiche sind Block-Elemente die links *floaten*. Der Entwickler hat auch die Möglichkeit das Layout bis auf die zweite Ebene zu verschachteln. Zu achten ist hier nur auf die Hilfsklassen, welche die unnötigen *paddings* entfernen. Die zweite Ebene wird auch bei kleinen Anzeigegeräten als eigene Spalten gerendert. Somit lassen sich individuelle Layouts für mobile Endgeräte umsetzen.

Tabelle 5.1: Beschreibung der Klassen die dem Entwickler zum Einbinden des bereitgestellten Rastersystems zur Verfügung stehen.

Klasse	*Beschreibung*
`.row`	Die Klasse `.row` ist ein Block, der den gesamten Bereich der jeweiligen *media query* flexibel von 320 px bis 1200 px ausnützt. Sie dient als Container der mittig im Browserfenster positioniert wird.
`.one-quarter`, `.one-third`, `.one-half`, `.two-third`, `.three-quarter`, `.total`	Diese Klassen unterteilen verschiedene Bereiche in definierte Spaltenbreiten. Hiermit können sehr flexibel verschiedene Designs umgesetzt werden.
`.alpha`, `.omega`	Diese Klassen werden benötigt, um mehrere Klassen ineinander zu verschachteln. Alpha löscht den linken Standardabstand und Omega den rechten Standardabstand.

5.2 Komponenten

Im folgenden Abschnitt wird die Implementierung der einzelnen Komponenten und der zur Verfügung stehenden Klassen beschrieben. Es steht für jede Komponente jeweils ein eigenes *Stylesheet* zur Verfügung, in welchem auch die *media queries* für das Anpassen an die Bildschirmgröße im Sinne vom *Responsive Design* umgesetzt sind. Ab welcher Bildschirmbreite von einem in das andere Layout gewechselt wird, ist global in dem Variablen-Modul definiert und wird für jede Komponente automatisch übernommen.

5.2.1 Navigation

Die Navigationskomponente stellt ein sich veränderndes Block-Element dar. Dieses passt sich aufgrund der Anzeigefläche des jeweiligen Benutzers automatisch an. Der Ausgangszustand des Navigationselements ist bei kleineren Ausgabegeräten ausgeblendet. Der Benutzer kann dieses über einen kleinen *Button* aktivieren. Dabei fließt das Block-Element von links in die Anzeigefläche und schiebt den Inhaltsbereich nach rechts aus dem Bild. Bei einer größeren Anzeigefläche ist der Navigationsbereich immer sichtbar.

Die Navigationskomponente ist mittels verschiedenen *CSS*-Kassen sowie -Selektoren umgesetzt. Für den Ausgangszustand wird das Navigationselement mit negativem `margin`-Wert aus dem Anzeigebereich verschoben. Dies ist für eine korrekte Darstellung einer *CSS-Animation* mit *Transition* not-

Tabelle 5.2: Beschreibung der Klassen und Rollen, die dem Entwickler zum Einbinden der bereitgestellten Navigationskomponente zur Verfügung stehen.

	Beschreibung
`.off-canvas-toggle`	Dieses Element stellt die Interaktionsmöglichkeit für den Benutzer bei kleinen Anzeigeflächen bereit. Bei Desktops ist es ausgeblendet. Diese Klasse sollte für ein `<nav>`-Element verwendet werden.
`role=''navigation''`	Definiert das im Markup verwendete Block-Element als Navigation. Mittels *CSS-Attribut-Selektor* wird die Anzeige des Elements definiert. Innerhalb dieses Elements muss eine *unordered list*, welche die Links beinhaltet, angelegt werden.
`role=''main''`	Definiert den im Markup verwendeten Container als Inhaltsbereich. Diese Zuordnung ermöglicht das Verschieben der Elemente.
`.active-nav`	Diese Klasse wird für eine logische Zuordnung verwendet und definiert, ob die Navigation eingeblendet ist.
`.vertical`	Bei größeren Anzeigeflächen wird standardmäßig die Navigation als horizontale Menüleiste angezeigt. Mit dieser Klasse hat der Web-Entwickler die Möglichkeit, die Navigation links neben dem Inhaltsbereich als eigene Spalte anzeigen zu lassen.

wendig. Um das Navigationselement einzublenden, wird bei kleinen Anzeigeflächen ein Menüsymbol in Form eines *Buttons* angezeigt. Das Handling der Benutzerinteraktion zur Laufzeit wird mittels *jQuery* durch die Zuordnung der *CSS-Klassen* durchgeführt. Die Animation ist mittels *CSS3-Transitions* umgesetzt. Die Kennzeichnung der Navigationskomponente im Markup erfolgt über das `role`-Attribut[1]. Tabelle 5.2 zeigt die verfügbaren Klassen und Rollen, welche im Markup zu verwenden sind.

5.2.2 Accordion zu Tabs

Oft ist es für einen Webentwickler notwendig, Inhalte zu unterteilen. Die Komponente *Accordion zu Tabs* stellt dafür ein passendes Element zur Verfügung. Mittels *media queries* wird die Anzeige der *HTML*-Elemente für das jeweilige Ausgabegerät optimiert. Der Aufbau dieser Komponente ist über

[1] http://www.w3.org/TR/role-attribute/

Tabelle 5.3: Beschreibung der Klassen welche dem Entwickler zum Einbinden der Komponente *Accordion zu Tabs* zur Verfügung stehen.

Klasse	*Beschreibung*
`.acc-tab`	Zuweisung der Komponente *Accordion zu Tabs* für die *unordered list* im Markup.
`.active`	Deklariert den aktiven Link. Diese Klasse wird für die logische Zuordnung verwendet, welcher Inhaltsbereich angezeigt werden soll.
`.is-open`	Diese Klasse blendet den aktiven Inhaltsbereich ein.

eine *unordered list* realisiert. Innerhalb der einzelnen Listen-Elemente wird zuerst ein Link angelegt und dann ein Bereich, welcher den Inhalt einschließt. Der Link dient als Interaktionselement, welcher die verschiedenen Inhaltsbereiche einblendet. Das Handling der Benutzerinteraktion zur Laufzeit wird mittels *jQuery* durch die Zuordnung der *CSS*-Klassen durchgeführt. Tabelle 5.4 zeigt die verfügbaren Klassen, welche im Markup zu verwenden sind.

5.2.3 Carousel

Ähnlich wie bei der Komponente *Accordion zu Tabs* können mit der Komponente *Carousel* Inhalte unterteilt werden. Hierbei werden die Inhaltsbereiche horizontal unterteilt und mit einer flüssigen Animation hin und her geschaltet. Auf *Touchscreen*-Displays erfolgt die Benutzerinteraktion durch wischen in die jeweilige Richtung, wobei bei herkömmlichen Monitoren der Wechsel der Inhaltsbereiche mit einfachem Mausklick erfolgt.

Ein Rahmen definiert die zur Verfügung stehende Fläche für die Anzeige der Inhaltsbereiche. Dieser Rahmen ist bei kleineren Anzeigegeräten auf die gesamte Bildschirmbreite skaliert und scrollbar. Auf größeren Monitoren ist die Größe abhängig davon, wo im Markup dieses Element eingesetzt wird. Ein Container innerhalb des Rahmens stellt den gesamten Platz für die Inhaltsbereiche nebeneinander zur Verfügung. Alles was über den Rahmen hinaus ragt, ist nicht sichtbar. Der Container wird mittels einer *unordered list* definiert und beinhaltet die einzelnen Inhaltsbereiche in Form von Listen-Elementen. Ein Beispiel für die Anwendung wäre eine *Slideshow*, wobei mehrere Bilder als Galerie zusammengefasst werden. Eine zusätzliche Klasse, die ein `<span>`-Element definiert, kann für die Legende der einzelnen Bilder zur Anzeige gebracht werden. Für den Desktop werden zwei Navigationselemente zum hin- und herschalten der Inhalte bereitgestellt.

Tabelle 5.4: Beschreibung der Klassen, welche dem Entwickler zum Einbinden der Komponente *Carousel* zur Verfügung stehen.

Klasse	*Beschreibung*
`.content-slider`	Zuweisung der Komponente *Carousel* zu der *unordered list* im Markup.
`.slides`	Definiert den Container für die Inhaltsbereiche.
`.is-open`	Diese Klasse blendet den aktiven Inhaltsbereich ein.
`.active`	Deklariert die Navigationselemente und welcher Inhaltsbereich zuvor bzw. danach angezeigt wird.
`.info`	Diese Klasse bietet einen Bereich für die Beschreibung des Inhalts an.

Programm 5.2: Klassendefinition für das Symbol `icon-home`.

```
1 [class^="icon-"]:before,
2 [class*=" icon-"]:before {
3   font-family: 'domino';
4   font-style: normal;
5   font-weight: normal;
6   display: inline-block;
7   text-decoration: inherit;
8   width: 1em;
9   margin-right: 0.2em;
10  text-align: center;
11  opacity: .9;
12 }
13
14 .icon-home:before { content: '\2302'; }
```

5.2.4 Symbole

Symbole sind ein dienliches Instrument, um den Inhalt von Informationen zu unterstreichen. Wie in Abschnitt 4.3.4 erklärt, werden die Symbole in Form eines Schriftstils mittels `@font-face` eingebunden. Zum Einbinden eines Symbols in die Webapplikation wird das `<i>`-Element in Verbindung mit der Klasse des gewünschten Symbols verwendet. Die Klassen sind immer gleich aufgebaut. Der Klassenname beginnt mit `icon-`. Mit der *CSS-Pseudoklasse* `:before` wird der jeweilige „Buchstabe“ hinzugefügt. Symbole können im Text oder vor Navigationselementen und *Buttons* verwendet werden. Das Programm 5.2 zeigt die generelle Definition aller Symbole und als Beispiel die Klassendefinition eines Symbols.

Tabelle 5.5: Beschreibung der Klassen, die dem Entwickler zum Einbinden von Formularen zur Verfügung stehen.

Klasse	*Beschreibung*
`.form-sbs`	Diese Klasse wird nur bei Anzeige auf größeren Monitoren verwendet. Hierbei wird das Formularfeld und das zugehörige *Label* in einer Zeile dargestellt.
`.form-inline`	Ermöglicht die Anzeige von Formularfeldern nebeneinander.

5.2.5 Formular

Die Formular-Komponente dient hauptsächlich der Browserkompatibilität und in zweiter Linie der Layoutierung auf großen Bildschirmen. Mit einem *JavaScript-Polyfill* werden die clientseitige Validierung sowie die neuen Attribute von *HTML5* nachgebildet. Auf kleinen Bildschirmen werden die Formularfelder untereinander angeordnet dargestellt. Das zugehörige *Label* wird dabei direkt oberhalb des jeweiligen Feldes angezeigt. Es wurden eigene Klassen, für eine abweichende Anzeige von größeren Bildschirmen erstellt. Dabei hat der Webentwickler die Möglichkeit, das *Label* links neben dem Formularfeld *floaten* zu lassen. Zusätzlich wurde eine Klasse für die Verwendung von Formularfeldern innerhalb von Texten definiert. Für die Beschreibung der Klassen siehe Tabelle 5.5.

5.2.6 Tabelle

Um die Darstellung von Tabellen auf kleine Anzeigegeräte zu optimieren, wurden die Tabellen-Elemente mittels *CSS* verändert. Die Darstellung der Tabellenelemente `<table>`, `<thead>`, `<tbody>`, `<th>`, `<tr>` und `<td>` wird auf `display: block` geändert. Somit verhalten sich die Elemente einer Tabelle wie normale Block-Elemente. Der Kopf der Tabelle wird ausgeblendet. Der Inhalt des Kopfes wird jedem Tabellen-Element mittels der *CSS-Pseudoklasse* `:before` als Inhalt hinzugefügt. Dies erlaubt die Veränderung einer Tabelle ohne großer Veränderung des Markups. Mittels *jQuery* wird jedem Tabellen-Element als `title`-Attribut die jeweilige Überschrift der Tabellenspalte zugeordnet. Programm 5.3 zeigt die optimierte Definition von Tabellen für kleine Bildschirme.

5.3 Dokumentation

Die Dokumentation bietet eine Demonstration jedes Elements, welches Bestandteil des *Frameworks* ist und erläutert dabei die Einbindung und die

Programm 5.3: Definition einer Tabelle optimiert für kleinere Bildschirme.

```
table, thead, tbody, th, td, tr {
  display: block;
}
thead tr {
  position: absolute;
  top: -9999px;
  left: -9999px;
}
td {
  border: none;
  position: relative;
  padding-left: 50%;
}
td:before {
  position: absolute;
  left: 1%;
  width: 45%;
  padding-right: 1em;
  content: attr(title);
}
```

Funktionen. Die Dokumentation selbst wurde als *Responsive Design* mit dem *Framework* umgesetzt. Dies ist gleichzeitig ein guter Test für den praktischen Einsatz des *DOMINO-Frameworks*.

5.4 Verwendung

Das *Framework* stellt zwei Versionen für Entwickler bereit: eine vorkompilierte Version die auch komprimiert ist und die Version mit dem Quellcode. Beide beinhalten die gleichen Features. Mit der vorkompilierten Version ist das System sofort einsatzbereit, dafür aber nicht so flexibel. Mit der Version des Quellcodes kann der Entwickler individuelle Anpassungen vornehmen, muss aber die Quelldateien selber kompilieren und komprimieren. Ein Beispiel für die Einbindung des *CSS-Frameworks* ist in Programm 5.4 ersichtlich.

Programm 5.4: Einbindung des Frameworks in ein neues Projekt.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>DOMINO Responsive CSS Framework</title>
5
6   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
7   <meta name="description" content="" />
8   <meta name="keywords" content="" />
9   <meta name="author" content="" />
10  <meta name="viewport" content="width=device-width, initial-scale=1,
      maximum-scale=1">
11
12  <link href="./css/domino.min.css" media="screen" rel="stylesheet" type
      ="text/css" />
13
14  <!--[if lt IE 9]>
15  <script src="//html5shim.googlecode.com/svn/trunk/html5.js"></script>
16  <![endif]-->
17  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.
      js"></script>
18  <script src="./js/domino.min.js"></script>
19 </head>
20
21 <body>
22
23 </body>
24 </html>
```

Kapitel 6

Evaluierung und Vergleich

Dieses Kapitel dient der Evaluierung der eigenen Arbeit sowie dem Vergleich mit, den in Kapitel 3 vorgestellten, fremden Ansätzen. Die Evaluierung soll einen Einblick in die Einsetzbarkeit des Systems geben. Der Vergleich mit den fremden Systemen dient der Abgrenzung sowie den klaren Aussagen über die Leistungsfähigkeit der einzelnen Systeme. Um die in Kapitel 4 definierten Kriterien in der Umsetzung zu überprüfen, werden folgende Schritte unternommen:

- Umsetzung einer Testanwendung auf Basis eines Layout-Mockups mit allen *Frameworks*,
- visueller Vergleich in verschiedenen Geräten und Browsern,
- Leistungsvergleich mittels zwei verschiedenen Tools,
- Auswertung der Ergebnisse.

6.1 Vergleichsprozess

Es ist schwierig die Qualität von fremden Systemen zu überprüfen. Einerseits muss damit gearbeitet werden, um den Workflow und die Einfachheit des Systems herauszufinden und andererseits müssen die Quelltexte unter die Lupe genommen werden, um verschiedene Lösungswege unterscheiden zu können. Da ein *CSS-Framework* hauptsächlich aus *CSS*-Definitionen besteht und zusätzliche Module zum Einbinden von Steuerungselementen zur Verfügung stellt, ist ein Vergleich verschiedener Lösungsansätze auf Basis des Quelltextes allein nicht aussagekräftig. Daher wird bei dieser Evaluierung neben den Standardkriterien, die ein *CSS-Framework* auf alle Fälle erfüllen sollte, zusätzlich eine Beispielanwendung mit dem jeweiligen Systemen umgesetzt. Diese dient dem Vergleich der unterschiedlichen Workflows und gleichzeitig der Überprüfung der Browserkompatibilität und dem Leistungsvergleich. Folgende Kriterien müssen von jedem System erfüllt werden:

- Layout Funktionalität,

- grundlegende Stildefinition,
- Browserkompatibilität.

6.1.1 Umsetzung eines Responsive Designs

Webentwickler arbeiten mit einem *CSS-Framework*, um eine bestimmte Webanwendung in Form und teilweise Funktion zu entwickeln. Diese Hauptaufgabe eines *Frameworks* wird in einer kleinen Testanwendung mit allen zu überprüfenden *Frameworks* umgesetzt und die Ergebnisse in Abschnitt 6.2 erläutert. Eine valide *HTML*-Datei wird als Grundlage für die Umsetzung mittels den verschiedenen Systemen erstellt. Es wird hierbei besonderer Wert auf die Umsetzung des *Responsive Designs* und der korrekten Darstellung auf verschiedenen Endgeräten gelegt.

Identifikation der Elemente

Einzig durch das Einbinden des Systems und den zur Verfügung stehenden Methoden soll das Layout der Vorlage umgesetzt werden. Folgende Liste enthält eine Beschreibung der verwendeten Elemente wie sie in Abbildung 6.1 mit Nummern gekennzeichnet sind.

- `100:` Kopfbereich, welcher ein Bild (101) und einen Navigationsbereich (102) beinhaltet.
- `102:` Navigationsbereich, welcher Navigationselemente (103) beinhaltet – blendet sich bei mobilen Endgeräten aus.
- `200:` Featured-Bereich, welcher ein großes Bild (201), einen Text (202), sowie ein Zitat (203) beinhaltet.
- `300:` Inhaltstrennung, mittels horizontaler Linie.
- `400:` Inhaltsbereich, aufgeteilt in drei Bereiche (401-403).
- `500:` Inhaltsbereich, welcher einen *Tabs*-Bereich (501) und ein Bild (504) beinhaltet.
- `501:` *Tabs*-Bereich, welcher einen Navigationsbereich (502) und einen Inhaltsbereich (503) beinhaltet – Inhalt kann durch Benutzerinteraktion gewechselt werden.
- `600:` Fußbereich, welcher einen Text (601) und ein Formular (602) beinhaltet.
- `602:` Formular, welches ein Textfeld, eine Checkbox und einen Button beinhaltet.

6.1.2 Validität

Bei manchen *CSS-Frameworks* ist es der Fall, eigens definiertes Markup für verschiedene Komponenten zu verwenden. Dies kann sich negativ auf die

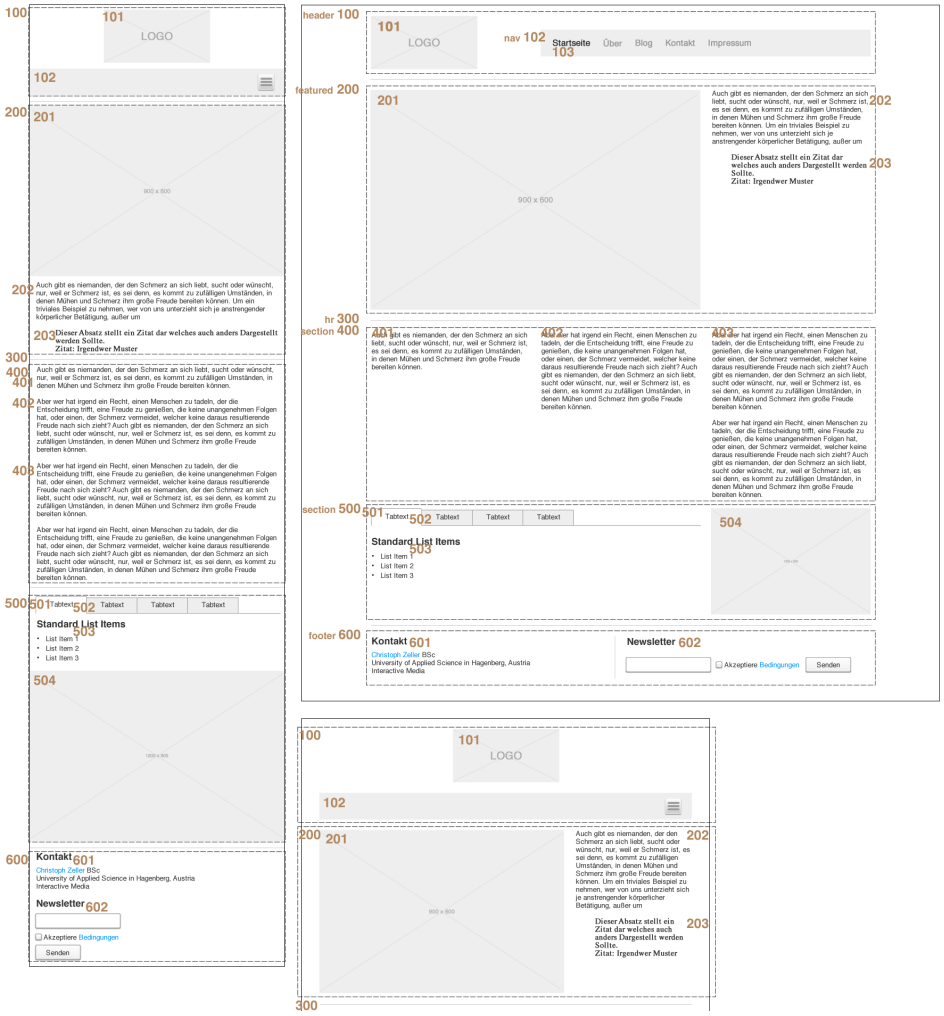


Abbildung 6.1: Mockup der Testanwendung. Nummerierung dient als Kennung für die Beschreibung der Elemente.

Standardkonformität auswirken. Um die Standardkonformität nach Erstellung der Testanwendung für jedes System zu testen, werden das jeweilige *HTML*-, sowie die eingebundene *CSS*-Datei validiert. Mittels dem von der *W3C* zur Verfügung gestelltem *Markup Validation Service*[1], sowie dem *CSS Validation Service*[2], wird die Validität der erstellten und eingebundenen Dateien überprüft.

[1] http://validator.w3.org
[2] http://jigsaw.w3.org/css-validator/

6.1.3 Visueller Vergleich

Der visuelle Vergleich dient der Kontrolle der Browserkompatibilität und dem erwarteten Verhalten des *Responsive Designs* auf den getesteten Systemen. Gleichzeitig wird eine visuelle Überprüfung der grundlegenden Stilelemente durchgeführt. Um einen ordentlichen Vergleich aller Systeme zu erreichen, wurde eine Testumgebung aufgebaut. Folgende Aussagen müssen für die positive Bewertung eindeutig zutreffen. Trifft eine der Aussagen nicht zu, ist die Testanwendung nicht mit dem jeweiligen Gerät bzw. Browser kompatibel.

- Automatische Anpassung des Layouts aufgrund der *Viewport*-Größe des Geräts.
- Bilder müssen auf die Größe des *Viewports* skaliert dargestellt werden.
- Alle Elemente werden wie erwartet angezeigt.
- Die bereitgestellten Funktionalitäten können ausgeführt werden.
- Text-Elemente werden angemessen dargestellt.

In der Tabelle 6.1 werden alle Geräte und Browser, die zur Überprüfung der visuellen Eigenschaften herangezogen wurden, aufgelistet.

6.1.4 Leistungsvergleiche

Um bei den Leistungsvergleichen die gleichen Voraussetzungen zu schaffen, wurden für alle Tests dieselbe Hardware und dieselbe Internetanbindung herangezogen. Auf dem verwendeten Webserver läuft *Apache/2.2.22* mit *Ubuntu*[3] als Betriebssystem. Dieser ist mittels einer *20Mbit/s*-Standleitung über die LinzAG mit dem Internet verbunden. Auf dem Server werden keine Maßnahmen zur Komprimierung der einzelnen Dateien getroffen. Die Testanwendung der jeweiligen Systeme wurden auf dem Server bereitgestellt und wird mit zwei frei verfügbaren Analysetools überprüft.

6.2 Ergebnisse

In diesem Abschnitt werden die, aus den Tests ermittelten, Ergebnisse angeführt und kurz erläutert.

6.2.1 Umsetzung Testanwendung

Im ersten Schritt wurde eine kleine Testanwendung für den weiteren Verlauf der Evaluierung der Systeme erstellt. Um einen aussagekräftigen Vergleich anzustellen, konnten nur zwei Komponenten eingebunden werden, weil *YAML* keine weiteren vergleichbaren Komponenten zur Verfügung stellt. Im folgenden Abschnitt findet sich eine kurze verbale Beschreibung, welche die

[3] http://www.ubuntu.com/

Tabelle 6.1: Die Tabelle enthält eine Geräteliste mit denen die Testanwendung überprüft wird.

Gerät	*Betriebssystem*	*Browser*
Apple iPhone4	iOS6	Mobile Safari
Apple iPhone4	iOS6	Chrome 23
Apple iPhone3G	iOS3.2	Mobile Safari
Samsung Galaxy S3	Android 4.1	Standardbrowser
Samsung Galaxy S3	Android 4.1	Chrome
LG P990	Android 2.3	Standard Android
BlackBerry Curve	BlackBerry OS 6.0	Standardbrowser
Motorola Xoom	Android 4.0	Standardbrowser
Motorola Xoom	Android 4.0	Chrome
Motorola Xoom	Android 4.0	Opera
Apple iPad2	iOS5	Mobile Safari
MacBook Pro	Mac OSX 10.7	Safari 6.0
MacBook Pro	Mac OSX 10.7	Firefox 18.0
MacBook Pro	Mac OSX 10.7	Google Chrome 24.0
PC	Windows 7	Internet Explorer 7
PC	Windows 7	Internet Explorer 8
PC	Windows 7	Internet Explorer 9
PC	Windows 7	Internet Explorer 10b
PC	Windows 7	Google Chrome 23.0
PC	Windows 7	Firefox 17.0
Sony Playstation 3	Firmware Version 4.31	Standardbrowser

positiven und negativen Erfahrungen beim Erstellen mittels der jeweiligen Systeme waren.

Bootstrap

Bootstrap stellt eine ausführliche Dokumentation[4] für Entwickler zur Verfügung. Wie bereits in Abschnitt 3.1.5 erwähnt, gibt es zwei Versionen, die *Bootstrap* für Entwickler zur Verfügung stellt. Zur Umsetzung der Testanwendung wurde die kompilierte Version herangezogen.

Die Einbindung von *Bootstrap* funktioniert über zwei zur Verfügung gestellte *Stylesheets* und eine *JavaScirpt*-Datei. Das *Stylesheet bootstrap.css* stellt die Kernfunktionalitäten des Systems bereit. Dieses wird mittels *bootstrap-responsive.css* um die Funktionalität der geräteunabhängigen Darstellung erweitert. Das bereitgestellte *JavaScript* erweitert das *Framework* um zusätzliche *User-Interface-Elemente*, wobei bei der Testanwendung nur die

[4] http://twitter.github.com/bootstrap/getting-started.html

Navigation, wie auch die *Tabs* zum Einsatz kommen. Für diese Elemente wird *jQuery* benötigt, welches ebenso eingebunden werden muss.

Die Layoutierung mit dem bereitgestellten Rastersystem ist mittels Dokumentation einfach nachvollziehbar. Es wurde ein fixes Raster mit jeweiligen Abstufungen für kleinere Darstellungsgeräte verwendet. Für die Aufteilung des Rasters können Elemente mit den Klassen *span1 – span12* unterteilt werden. Dieses Raster ist nach dem Prinzip *Desktop-First* aufgebaut. Das Layout wird erst für den Desktop umgesetzt. Die Anpassung auf kleinere Ausgabegeräte erfolgt automatisch indem die einzelnen Spalten-Elemente als Block-Elemente mit fixer Breite mittels *media queries* umgewandelt werden.

Für die Einbindung der zwei verwendeten Komponenten, muss der Entwickler das Markup entsprechend anpassen. Diese Vorgehensweise erleichtert dem *Framework* die logischen Operationen mittels *JavaScript* zu implementieren. Für den Entwickler bedeutet diese Tatsache, dass er die proprietäre Umsetzung erlernen muss. Ob sich dies negativ auf die Validierung bzw. Anzeige mit veralteten Browsern auswirkt, wird sich in weiterer Folge zeigen. Die eingebundenen Bilder sowie das Formular, werden ohne Zutun des Webentwicklers automatisch für das jeweilige Anzeigegerät angepasst.

YAML

YAML stellt eine ausführliche Dokumentation[5] für Entwickler zur Verfügung. Vier verschiedene Beispiellayouts erleichtern den Einstieg. Für die Umsetzung der Testanwendung wurde auf dem Beispiel „Flexible Grid“ aufgesetzt.

YAML stellt für die Darstellung auf Desktops eine Klasse zur Verfügung, welche das Layout mittig vom Bildschirm anordnet. Diese Klasse *ym-wrapper* umfasst in der Vorlage jeden Bereich extra – Kopf-, Navigations-, Inhalts und Fußbereich. Deshalb wurde dies auch bei der Testanwendung so umgesetzt, auch wenn diese Tatsache das Markup unnötig vergrößert. Mittels einem zentralen *Stylesheet*, welches in die *HTML*-Datei eingebunden wird, werden die notwendigen weiteren *Stylesheets* mit der *@import*-Anweisung nachgeladen.

Der Kopfbereich wurde mit dem bereitgestellten Raster in ein Drittel für das Logo und zwei Drittel für die Navigation unterteilt. Für das Logo sowie für alle anderen Bilder, stellt das *Framework* eine Klasse *flexible* bereit, welche die Größe des Bildes relativ zur Viewportgröße anpasst. Für das *Responsive Design* gibt es eine zweistufige Unterteilung. Automatisch werden die Elemente nicht angepasst. Der Entwickler entscheidet selbst welche Elemente, bei welcher Maximalbreite eine andere Form erhalten. Dies realisiert das *Framework* über zwei Klassen. Bei Verwendung der Klasse *linearize-level-2* werden Elemente bis zu einer Breite von 480 Pixel und bei Verwendung der Klasse *linearize-level-1* bis zu einer Breite von 760 Pixel als Block-Elemente dargestellt. Darüber hinaus *floaten* die Elemente, wie in der Beschreibung vorgesehen.

[5] http://www.yaml.de/docs/index.html

DOMINO

DOMINO stellt zwei Versionen für Entwickler zum Einbinden des *Frameworks* bereit. Für die Umsetzung der Testanwendung wurde die vorkompilierte Version verwendet. Dies geschieht durch Einbindung des zur Verfügung gestellten *Stylesheets*, sowie der *JavaScript*-Datei. Für die logischen Funktionen der Komponenten wird *jQuery* benötigt, welches ebenso eingebunden werden muss.

Die zur Verfügung stehende Dokumentation erklärt das Prinzip des Rastersystems sowie die Vorgehensweise bei der Verwendung von Komponenten. Diese werden mit *CSS-Klassen* und zusätzlichen *Element-Attributen* angewendet. Mittels den *Element-Attributen* entsteht eine logische Verknüpfung für die erweiterten Funktionen. In der Testanwendung ist die Komponente *Navigation*, sowie die Komponente *Accourdion zu Tabs* verwendet worden. Der Vorteil ist, dass das Markup an sich nicht verändert werden muss.

Mittels sprechenden Namen der *CSS-Klassen* ist es für Webentwickler einfach, das Layout anhand dem zur Verfügung stehenden Raster umzusetzen. Das Raster ist nach dem Prinzip *Mobile First* umgesetzt. Hierbei wird erst das Layout für die mobile Darstellung angewendet und erst in weiterer Folge für die Darstellung auf dem Desktop umgesetzt. Der Webentwickler sollte bei der Umsetzung mit Hilfe dem bereitgestellten Raster schon eine Vorstellung über die Darstellung am Desktop haben um die *CSS-Klassen* für die richtigen Elemente anzuwenden. Dazu finden sich in der Dokumentation Beispiele. Die grundlegenden Funktionen sowie Stile passen sich automatisch an die Größe des jeweiligen Anzeigegeräts an.

6.2.2 Validierung der Quelltexte

Die erstellten Testanwendungen der drei Systeme wurden jeweils mit dem Service vom *W3C* validiert. Hierbei wird der erstellte Quellcode auf Fehler überprüft. Die Überprüfung der drei erstellten Testanwendungen ergaben keine Fehler. Einzig eine Warnung, dass hier experimentelle Features – gemeint sind *HTML5*-Elemente – verwendet werden.

Ein anderes Resultat ergab die Überprüfung der bereitgestellten *CSS*-Dateien. Der Validierungsdienst für *CSS*-Dateien ist äußerst penibel bei der Überprüfung von *CSS*-Anweisungen. Beispielsweise werden *CSS-Hacks*, die oft für die Browserkompatibilität verwendet werden oder *Vendor-Prefixes*, die neuere Funktionen einzelnen Browsern bereitstellen, von diesem Dienst als Fehler bzw. Warnungen ausgewertet. Die Validierung von *Bootstrap* ergab 560 Fehler und 240 Warnungen in den *CSS*-Dateien. *YAML* hingegen kam auf insgesamt 68 Fehler und 45 Warnungen. Mit 16 Fehlern und 18 Warnungen ergab die Validierung des selbst entwickelten *Frameworks* das beste Ergebnis auf Basis der Standard-Konformität.

6.2.3 Visueller Vergleich

Der visuelle Vergleich der Testanwendung auf den in der Tabelle 6.1 aufgelisteten Geräten und Browsern hat ergeben, dass alle verglichenen *Frameworks* die Darstellung für moderne Browser korrekt ausführen. Einzig beim *Microsoft Internet Explorer 7* und *Microsoft Internet Explorer 8* gab es Probleme bei der Darstellung. *YAML* war das einzige *Framework*, das auch beim *Microsoft Internet Explorer 8* eine korrekte Darstellung lieferte. Die Funktionalitäten waren bei allen drei *Frameworks* auf allen getesteten Geräten und Browsern erhalten. *DOMINIO* erreichte durch die angewandte Methode von *Progressive Enhancement* auch am *IE7* und *IE8* eine annehmbare Darstellung. Die Informationen werden klar strukturiert untereinander angezeigt. Die Testanwendung hingegen, die mit *Bootstrap* umgesetzt wurde, verschob die Inhalte willkürlich. Somit hat der Benutzer keine klare Struktur, welcher er folgen kann. Webentwickler müssen sich bei der Verwendung von *Bootstrap* im Klaren sein, dass ältere Browser es nicht unterstützen. *YAML* bietet im Grundsystem keine Funktionalität, um die Navigation in der mobilen Variante auszublenden. Diese wird untereinander dargestellt und kann bei längeren Strukturen unübersichtlich werden.

6.2.4 Leistungsvergleich

Für die Leistungstests wurden zwei frei verfügbare *Online-Tools* herangezogen. Zum einen *PageSpeed Tools*[6] von Google. Es bietet zahlreiche Empfehlungen übersichtlich zusammengefasst. Diese basieren auf Googles *Web Performance Best Practices*[7]. Zum anderen *Web Page Analyzer*[8], welches Aufschluss über die Größe und Anzahl der übertragenen Daten liefert. Ladezeiten für mehrere Verbindungstypen und übertragene Datengrößen werden ebenso gemessen.

Die Abbildung 6.2 zeigt die Ladedauer der einzelnen Testanwendungen vom Server. Vergleicht man dazu die Abbildung 6.3, welche die Datengröße der jeweiligen Testanwendungen darstellt, ist der Zusammenhang zwischen Datengröße und Ladezeit ersichtlich. Umso mehr Daten geladen werden müssen, umso länger ist die Dauer bis die Webanwendung angezeigt wird. Die Abbildung 6.4 zeigt die Anzahl der *HTTP-Requests* der jeweiligen Testanwendung. Wie stark sich diese auf die Ladezeit der Webanwendung auswirkt, ist auf den ersten Blick nicht sichtbar. Eine Schlussrechnung verdeutlicht diesen Umstand.

$$\frac{\text{Datenmenge } \textit{YAML} * \text{Ladezeit } \textit{Bootstrap}}{\text{Datenmenge } \textit{Bootstrap}} = \text{Ladezeit } \textit{YAML}$$

[6] https://developers.google.com/speed/pagespeed/
[7] https://developers.google.com/speed/docs/best-practices/rules_intro?hl=de
[8] http://www.websiteoptimization.com/services/analyze/

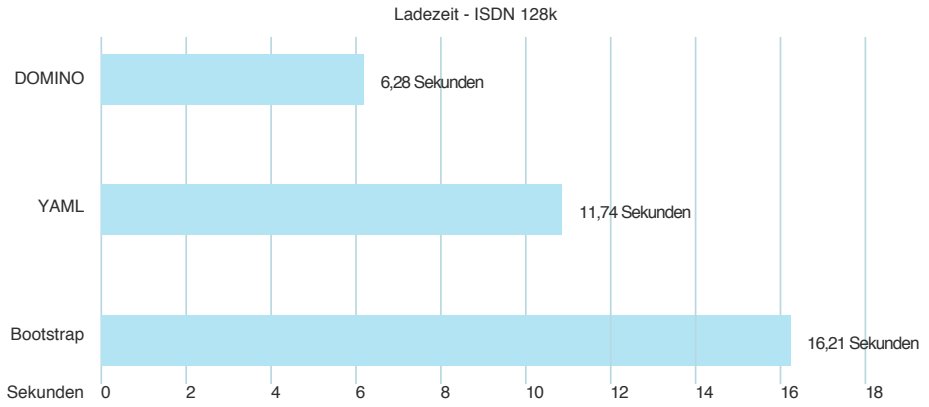


Abbildung 6.2: Ladedauer der einzelnen Testanwendungen.

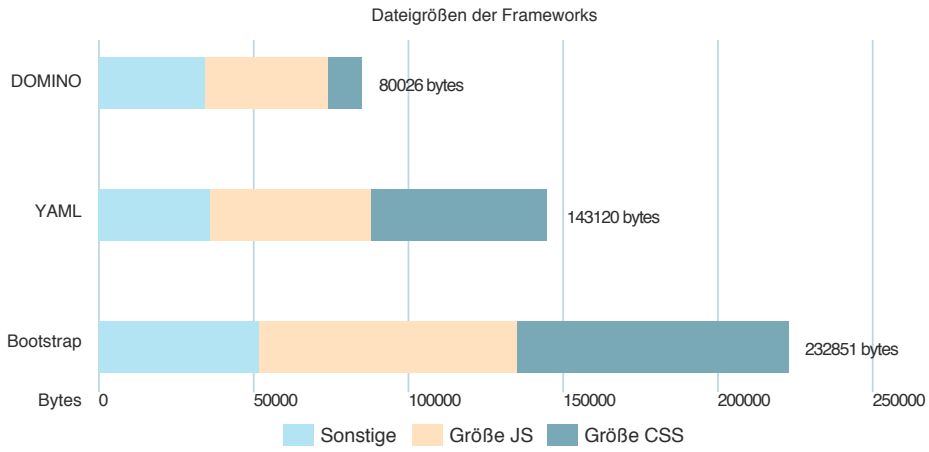


Abbildung 6.3: Verteilung der Dateigrößen des jeweiligen Frameworks.

$$\frac{143.120b * 16,21s}{232.851b} = 9,96s$$

Hierbei wird ersichtlich, dass die höhere Anzahl an *HTTP-Requests* eine um fast zwei Sekunden längere Ladezeit als im Vergleich mit *Bootstrap* verursacht.

Die generelle Empfehlung von den verwendeten Analyse-Tools ist es, die Anzahl der nachzuladenden Dateien zu reduzieren. Bei diesen sollte nach Möglichkeit zusätzlich die Datenmenge minimiert werden. Es soll ebenso die Dateikomprimierung auf dem Server aktiviert sowie die verwendeten Bilder optimiert zur Verfügung gestellt werden. Dies liegt in der Verantwortung der Webentwickler, weil die *Frameworks* darauf keinen Einfluss haben.

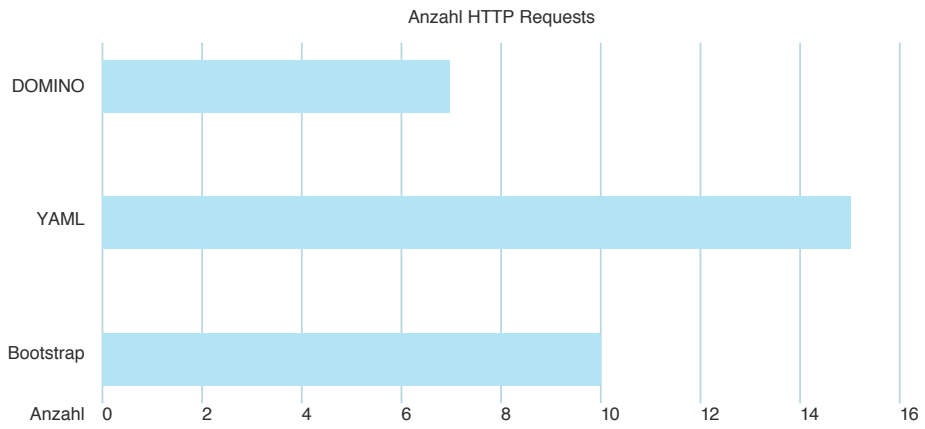


Abbildung 6.4: Anzahl der HTTP-Requests, welche die einzelnen Testanwendungen ausführten.

Kapitel 7

Fazit und Ausblick

Die Auswertung hat gezeigt, dass es mit allen drei vorgestellten Systemen möglich ist, bildschirmauflösungs- und geräteunabhängige Webanwendungen zu erstellen. Einzig die Funktion sowie die Herangehensweise unterscheiden sich. Es kommt also darauf an, welches Ziel der Webentwickler bei der Erstellung einer neuen Webanwendung verfolgt.

Bootstrap stellt mittels den Kernfunktionalitäten für alle Elemente eine grundlegende Formatierung zur Verfügung. Diese fügen sich optimal auf die zur Verfügung stehenden Darstellungsflächen ein. Das Konzept ist gut durchdacht und es geschieht sehr viel automatisch. Dies ist von Vorteil, wenn der Webentwickler schnell Ergebnisse erzielen möchte. Die Flexibilität ist aufgrund der Umsetzung mittels *LESS* auch gegeben. Webentwickler finden sich schnell zurecht, weil der Aufbau klar strukturiert ist. Durch die hohe Anzahl der zur Verfügung stehenden *User-Interface*-Elemente, erhöht sich die Datenmenge, der einzubindenden Dateien. Dies führt beim Einsatz von wenigen Elementen zu einer großen Datenmenge die vermeidbar wäre. Hierbei empfiehlt es sich diese Elemente individuell für den jeweiligen Einsatz anzupassen.

YAML ist flexibel einsetzbar und stellt für die Layoutierung verschiedene Varianten bereit. Der Entwickler hat viel Entscheidungsfreiheit, was aber auch bedeutet, dass dieser vieles noch selbst implementieren muss. Für die zur Verfügung stehenden Funktionen werden keine sprechenden Namen verwendet. Dies kann dazu führen, dass Entwickler länger brauchen sich in das System einzuarbeiten. Die in der Dokumentation empfohlenen Methoden zum Nachladen weiterer *Stylesheet*-, sowie *JavaScript*-Dateien, schont die Ressourcen der Benutzer nicht. Weiters steht nur eine geringe Anzahl an fertigen *User-Interface*-Elementen zur Verfügung. Der große Vorteil liegt einerseits an der hohen Anzahl an fertigen Implementierungen für verschiedenste *Content-Management-Systeme* und andererseits der flexiblen Einsetzbarkeit.

DOMINO ist flexibel und anpassungsfähig. Es stellt eine Reihe von Er-

weiterungen zur Verfügung, die je nach Bedarf verwendet werden können. Der Entwickler hat durch einfaches Weglassen oder Austauschen die Möglichkeit, das *Framework* seinen eigenen Bedürfnissen anzupassen bzw. zu erweitern. Hauptsächlich unterscheidet es sich von den beiden anderen Systemen durch die Unterstützung des Prinzips *Mobile First* bei der Erstellung einer Webanwendung. Alle Module und Komponenten wurden nach diesem Prinzip aufgebaut und stehen für die weitere Verwendung zur Verfügung. Die kompilierte Version bietet Webentwickler die Möglichkeit, rasch einen Prototypen zu realisieren. Die Umsetzung mittels *LESS* bringt die nötige Flexibilität, die Webentwickler für die Umsetzung spezieller Anforderungen brauchen.

Die Erstellung einer Weboberfläche zur individuellen Konfiguration des Systems ist geplant. Dies erleichtert Webentwickler die Anpassung an ihre eigenen Bedürfnisse. Um die Weiterentwicklung von *DOMINO* voran zu treiben, wird das System in naher Zukunft auf *github*[1] veröffentlicht. Dies soll anderen Entwicklern ermöglichen, Feedback oder Fehler zu melden und eigene Entwicklungen bereit zu stellen.

Ist Flexibilität eines der Hauptkriterien, ist der Webentwickler mit *DOMINO* oder *YAML* meist besser bedient. Auch für Prototypen bietet *DOMINO* eine gute Voraussetzung, wobei *Bootstrap* eine höhere Anzahl an fertigen *User-Interface*-Elementen zur Verfügung stellt. Um möglichst schnell eine Webanwendung in Form und Funktion umzusetzen, ist *Bootstrap* dank der großen Entwicklergemeinde die beste Wahl.

[1] https://github.com

Anhang A

Inhalt der CD-ROM/DVD

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

A.1 Masterarbeit

Pfad: /

Zeller_Christoph_2013.pdf Masterarbeit (Gesamtdokument)

A.2 Online-Literatur

Pfad: /web-pdfs

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

A.3 Systemvergleich

Pfad: /vergleich

/testanwendung.zip . . Testanwendung für Systemvergleich
/visuell/*.png . . . . . Screenshots der visuellen Überprüfung
/leistung/*.pdf . . . . . Ergebnisse der Leistungstests

A.4 Projektdateien

Pfad: /project

/domino . . . . . Kompilierte Version
/demo . . . . . Dokumentation und Demo
* . . . . . Quelltexte

Quellenverzeichnis

Literatur

[1] Dan Cederholm. *CSS3 for Web Designers*. New York: A Book Apart, 2010.

[2] Aaron Gustafson. *Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement*. Chattanooga, Tennessee, USA: Easy Readers, LLC, 2011.

[3] Bruce Lawson und Remy Sharp. *Introducing HTML 5*. New Riders, 2011.

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

[5] Josef Müller-Brockmann. *Grid systems in graphic design*. Niggli, 1981.

[6] Todd Parker u. a. *Designing with Progressive Enhancement: Building the Web that Works for Everyone*. Thousand Oaks, CA, USA: New Riders Publishing, 2010.

[7] Luke Wroblewski. *Mobile First*. New York: A Book Apart, 2011.

Online-Quellen

[8] URL: http://de.wikipedia.org/wiki/HTML5 (besucht am 29.07.2012).

[9] URL: http://de.wikipedia.org/wiki/Cascading_Style_Sheets (besucht am 26.08.2012).

[10] *Absatz von Smartphones weltweit vom 1. Quartal 2009 bis zum 2. Quartal 2012 nach Betriebssystem*. URL: http://de.statista.com/statistik/daten/studie/74592/umfrage/absatz-von-smartphones-weltweit-nach-betriebssystem/ (besucht am 19.08.2012).

[11] Chloe Albanesius. *Google's New Rule: Mobile First*. URL: http://www.pcmag.com/article2/0,2817,2359752,00.asp (besucht am 18.12.2012).

[12] David Baron. *CSS Conditional Rules Module Level 3*. URL: http://www.w3.org/TR/2012/WD-css3-conditional-20121213/ (besucht am 13.11.2012).

[13] Mark Boulton. *Structure First. Content Always.* URL: http://www.markboulton.co.uk/journal/structure-first-content-always (besucht am 23. 12.2012).

[14] Rik Cabanier u. a. *HTML Canvas 2D Context.* URL: http://www.w3.org/TR/2012/CR-2dcontext-20121217/ (besucht am 20.12.2012).

[15] Jacob Gube. *Module Tabs in Web Design: Best Practices and Solutions.* URL: http://www.smashingmagazine.com/2009/06/24/module-tabs-in-web-design-best-practices-and-solutions/ (besucht am 22.03.2012).

[16] Peter Kröner. URL: http://upload.wikimedia.org/wikipedia/commons/1/19/HTML5-Spezifikations-Übersicht.svg (besucht am 26.11.2012).

[17] Ethan Marcotte. *Fluid Grids.* URL: http://www.alistapart.com/articles/fluidgrids/ (besucht am 03.12.2012).

[18] Monika Steinberg. *HTML5: W3C und WHATWG gehen getrennte Wege.* URL: http://t3n.de/news/html5-w3c-whatwg-gehen-getrennte-403496/ (besucht am 12.12.2012).

[19] *Warm Gun conference.* URL: http://www.ustream.tv/recorded/10072913 (besucht am 19.12.2012).

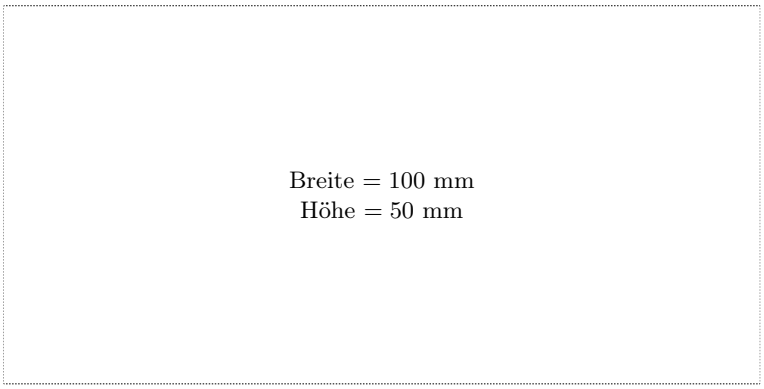
[20] *Weltweit ausgelieferte Smartphones und PCs im Jahr 2012 und 2011.* URL: http://de.statista.com/statistik/daten/studie/217329/umfrage/weltweite-auslieferung-von-smartphones-und-pcs/ (besucht am 19.08. 2012).

[21] *What Users Want from Mobile.* URL: http://www.compuware.com/d/release/592528/new-study-reveals-the-mobile-web-disappoints-global-consumers (besucht am 28.12.2012).

[22] Luke Wroblewski. *Multi-Device Layout Patterns.* URL: http://www.lukew.com/ff/entry.asp?1514 (besucht am 12.12.2012).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —