Ein komponentenbasiertes Framework zur serverseitigen Optimierung von Responsive Webdesign

Christoph Pömer

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2015

Erklärung

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

Hagenberg, am 29. Juni 2015

Christoph Pömer

Inhaltsverzeichnis

Kurzfassung

Die Internetnutzung hat sich in den letzten Jahren stark verändert. Besucher rufen eine Website heute neben einem Desktop PC vermehrt auch von mobilen Endgeräten auf und erwarten, dass ihnen derselbe Inhalt auf all ihren Geräten funktional aufbereitet zur Verfügung steht. Dementsprechend steigt der Bedarf an anpassungsfähigen und flexiblen Umsetzungsstrategien. Responsive Webdesign, das sich in den letzten Jahren schnell weiterentwickelt hat, repräsentiert eine zukunftsfähige Universallösung. Es ermöglicht die Erstellung einer einzigen Website, die ihr Layout dynamisch an das jeweilige Endgerät anpasst.

Allerdings bietet Responsive Webdesign nicht nur Vorteile, sondern ist auch mit diversen Nachteilen verbunden. Wird eine Website mit Responsive Webdesign erstellt, werden dieselben Daten an jedes Gerät gesendet. Das bedeutet, dass Bilder meist nur skaliert und nicht reduziert werden. Auch ein Austausch und eine Optimierung von einzelnen Websitekomponenten für verschiedene Geräteklassen ist mit Responsive Webdesign nicht umsetzbar. Dementsprechend weisen responsive Websites gewöhnlich auch auf einem mobilen Gerät ein hohe Dateigröße auf, was wiederum zu langen Ladezeiten und einer schlechten Performance führen kann.

Um eine optimale Performance und ein damit verbundenes gutes Nutzungserlebnis zu bieten, müssen bestimmte Bestandteile einer Website, wie die Navigation oder die eingebundenen Bilddateien, für verschiedene Geräteklassen und Displaygrößen optimiert werden. Solche Verbesserungsmaßnahmen können allerdings nur bedingt mit clientseitigem Responsive Webdesign realisiert werden. Aus diesem Grund erweitert der erstellte Framework-Prototyp ein responsives Layout um eine serverseitige Komponente.

Die Integration einer serverseitigen Komponente ermöglicht eine auflösungsabhängige Optimierung und Auslieferung von Bildern sowie einen gerätespezifischen Austausch von einzelnen Seitenbestandteilen. Durch diese Erweiterung können signifikante Performanceverbesserungen erreicht und die Flexibilität einer responsiven Website erhöht werden.

Abstract

The way we use the Internet has changed over the last few years. More and more mobile devices are conquering the Web, which leads to entirely new challenges for Webdesigners and developers. The mobile world has many constraints and it is no longer solely about the evolving capabilities of the devices but rather about consistency and a pleasant cross-device user experience. Due to this, there is a huge demand for adaptable and flexible implementation strategies. Responsive Webdesign that has evolved rapidly over the last few years, represents an important solution to develop mobile-optimized websites. It enables the delivery of a single website that dynamically adjusts its layout to the respective device.

But there are also many drawbacks and challenges, Responsive Webdesign has to deal with. Using Responsive Webdesign, the same content is sent to all accessing devices. It means that the same website components (e.g. images) that are sent to a desktop monitor are sent to devices with low-resolution screens, even if they can not show them in their native resolution. In addition, swapping and adjusting specific website components, based on defined device classes, is not possible with client-side Responsive Webdesign. This leads to larger file sizes, longer page loading times and a loss of performance on mobile devices.

In order to ensure an optimal performance and a good user experience, certain components of a website, such as the navigation pattern or images, need to be optimized for different device classes and screen sizes. However, such improvements can not be implemented properly with client-side Responsive Webdesign. Therefore, the implemented Framework-Prototype extends and combines a client-side responsive layout with server-side component optimization.

The purpose of this approach is to make the whole concept more efficient. It allows to deliver optimized images when they are requested and to swap entire website components such as navigation patterns. This means that partial pieces of a website can be inserted and removed, where appropriate, in a larger full responsive layout that is given to all browsers. With these enhancements, significant performance improvements are possible and a website converts from a fixed size to a highly varying one.

Kapitel 1

Einleitung

In den vergangenen Jahren haben immer mehr mobile Geräte das Web erobert. Somit reicht eine rein auf den Desktop ausgerichtete Website heute nicht mehr aus. Diese Tatsache führt zu völlig neuen Herausforderungen für Webdesigner und Webentwickler. Mobile Geräte, wie Smartphones und Tablets, haben verschiedene Eigenschaften und Interaktionsmöglichkeiten, was die Darstellung und Bedienung einer Website betrifft. Diese müssen von Entwicklern aufgegriffen und genutzt werden. Da Besucher dieselbe Website sowohl von mobilen Geräten als auch von Desktop PCs aufrufen, geht es bei der Erstellung eines Webauftritts heute nicht mehr nur um die sich ständig entwickelnden Fähigkeiten und Ressourcen dieser Geräte, sondern auch um Konsistenz und eine geräteübergreifende User Experience[1]. Nutzer erwarten, dass ihnen der gleiche Inhalt auf all ihren Geräten funktional aufbereitet zur Verfügung steht [18, S. 6].

Responsive Webdesign ermöglicht die Erstellung einer einzigen Website, die ihr Layout automatisch an die Displaygröße des jeweiligen Geräts anpasst und stellt somit eine Universallösung dar. Dies betrifft vor allem die Anordnung und Darstellung von verschiedenen Websiteelementen. Für viele Websites erreicht Responsive Webdesign ein ausgewogenes Verhältnis zwischen Nutzbarkeit und dem Aufwand der Implementierung und entwickelt sich so zur bevorzugten Methode für die Erstellung von mobilen Websites. Allerdings bietet Responsive Webdesign nicht nur Vorteile sondern führt auch zu einer Reihe von Einschränkungen. Durch die Erstellung einer einzelnen anpassungsfähigen Website, die an alle verfügbaren Geräte ausgeliefert wird, wird auch dieselbe Datenmenge an all diese Geräte gesendet, was wiederum zu, im mobilen Bereich problematischen, unnötigen Dateigrößen und Ladezeiten führt.

[1] User Experience umschreibt alle Aspekte der Erfahrungen eines Nutzers bei der Interaktion mit einem Produkt, Dienst, einer Umgebung oder Einrichtung [65].

1.1 Problemstellung und Relevanz

Webdesignern bietet Responsive Webdesign eine gute Möglichkeit Websites, die für verschiedene Endgeräte optimiert sind, zu entwickeln. Allerdings hat Responsive Webdesign auch eine Reihe von Nachteilen und Einschränkungen. Bei der Entwicklung einer Website müssen neben der Displaygröße auch andere Faktoren, wie die verfügbare Netzwerkverbindung, eine Unterstützung von bestimmten Seitenbestandteilen, wie Videos, oder die HTML und CSS Kompatibilität eines Geräts berücksichtigt werden. Das bedeutet, dass verschiedene Websitekomponenten ausgetauscht und an das jeweilige Endgerät angepasst werden müssen. Solche Erweiterungen sind mit Responsive Webdesign nicht möglich und können nur durch serverseitige Logik erreicht werden.

Der theoretische RESS Ansatz[2] von Luke Wroblewski versucht einige dieser Nachteile aufzuheben (siehe Abschnitt 3.3). Es ist ein Konzept das Responsive Webdesign mit serverseitiger Logik kombiniert. Serverseitige Anpassungen können diverse Optimierungen sowie signifikante Performanceverbesserungen für mobile Geräte und Desktop PCs ermöglichen. Für die Realisierung eines solchen RESS-Systems gibt es allerdings keine konkreten Vorgaben. Dementsprechend ist die Umsetzung des Konzeptes mit einem erheblichen serverseitigen Programmieraufwand verbunden. Die große Herausforderung dieses kombinierten Lösungsansatzes liegt daher in einer erforderlichen Zusammenarbeit zwischen Webentwickler und Webdesigner im gesamten Layout- und Design-Prozess. Allerdings wäre es wünschenswert, diese beiden Aufgabengebiete getrennt zu halten. Ein System, welches Webdesignern die Umsetzung einer entsprechenden Website ermöglicht und dabei keinen serverseitigen Programmieraufwand erfordert, würde somit eine ideale Lösung darstellen.

1.2 Zielsetzung

Im Zuge dieser Arbeit wird ein Framework-Prototyp erstellt, mit dem Ziel einige Einschränkungen und Nachteile von Responsive Webdesign durch die Integration einer serverseitigen Komponente aufzuheben. Um eine bessere Anpassung von Websitekomponenten an verschiedene definierte Geräteklassen zu erreichen, werden gängige serverseitige Geräteerkennungsbibliotheken integriert und die ermittelten Daten mit gerätebezogenen clientseitigen Abfragen kombiniert. Basierend auf den analysierten Gerätedaten wird eine serverseitige Bildoptimierungsfunktion implementiert, die eingebundene Bilddateien automatisch reduziert. Darüber hinaus wird eine Template-Engine integriert, die es Webdesignern ermöglicht, verschiedene Seitenbestandteile für definierte Geräteklassen zu spezifizieren. Der primäre Fokus des Frame-

[2] RESS: Responsive Design + Server Side Components

works liegt dementsprechend auf der Auslieferung von auflösungsoptimierten Bildern und der Anpassung von Websiteelementen, basierend auf definierten Geräteklassen. Der finale Framework-Prototyp soll ein hohes Maß an Flexibilität aufweisen und die Erstellung eines Website-Frontends, unabhängig der verwendeten Backend-Lösung, ermöglichen.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist in einen theoretischen und einen praktischen Teil gegliedert. Die theoretischen Kapitel der Arbeit stellen das fachliche Fundament für den eigenen Ansatz dar und dienen auch zur Erläuterung der verwendeten Begriffe.

In *Kapitel 2* werden die gängigen Verfahren zum Aufbau mobiler Websites definiert und kurz beschrieben. Ebenso wird auf die Vor- und Nachteile der einzelnen Lösungsansätze eingegangen. Der primäre Fokus liegt hier auf Responsive Webdesign, da der eigene Ansatz auf diesem Verfahren aufbaut.

Kapitel 3 widmet sich verwandten Ansätzen, die bereits eine Verbesserung und Erweiterung von Responsive Webdesign ermöglichen.

Kapitel 4 beschäftigt sich mit dem Design des eigenen Lösungsansatzes. Dieses Kapitel gibt einen Einblick in die Anforderungen und grundlegenden eigenen Überlegungen zum Aufbau des Framework-Prototyps. Der Fokus liegt hier auf einer Optimierung von Bildern und einer Möglichkeit, einzelne Seitenelemente für definierte Geräteklassen zu spezifizieren.

In *Kapitel 5* wird die praktische Umsetzung des im vorigen Kapitel theoretisch beschriebenen Lösungsansatzes erläutert. Dabei wird auch auf die Verwendung des Frameworks eingegangen.

Kapitel 6 beschäftigt sich mit der Evaluierung des Framework-Prototyps. Dabei erfolgt eine Reflexion der gestellten Anforderungen und eine Eigenanalyse bzw. Argumentation zur Flexibilität und dem Nutzwert des Systems. Um eventuelle Performanceverbesserungen durch die Nutzung des eigenen Ansatzes aufzuzeigen, wird ein Leistungsvergleich mit anderen Lösungen durchgeführt.

Im abschließenden *Kapitel 7* wird ein Fazit über die gesamte Arbeit und die aktuellen Entwicklungen im Bereich Responsive Webdesign gezogen.

1.4 Gendering

In der vorliegenden Arbeit wird auf eine geschlechtsgerechte Schreibweise verzichtet, um einen stringenten Aufbau sowie angenehmen Lesefluss bzw. eine angenehme Lesbarkeit und Verständlichkeit zu verwirklichen. Es wird darauf hingewiesen, dass das generische Maskulinum verwendet wird, obwohl gleichzeitig die weibliche als auch die männliche Person angesprochen ist.

Kapitel 2

Technologische Grundlagen

Die Internetnutzung verlagert sich immer mehr zum mobilen Web. Die Vorteile des mobilen Internets sind klar ersichtlich. Nutzer haben heute zu jeder Zeit und in jeder Lebenssituation, Zugriff auf benötigte Informationen. Dieser Komfortgewinn stellt allerdings auch viele neue Anforderungen an die Gestaltung und Aufbereitung von Websites. Webdesign muss flexibler werden. Ein rein auf den Desktop PC ausgerichtetes Layout reicht heute nicht mehr aus. Dieses kann auf einem mobilen Gerät, mit einem kleineren Display und einer niedrigeren Auflösung, nicht benutzerfreundlich dargestellt werden. Lange Ladezeiten und eine umständliche Bedienung, die nur durch permanentes Scrollen und Zoomen ermöglicht wird, sind meist die Folge. Hinzu kommt eine eventuelle Bedienung der Geräte per Finger und Touchdisplay, die weitere Anforderungen an die Usability einer Website stellt [13].

Basierend auf diesen Tatsachen haben sich verschiedene Verfahren entwickelt, die eine Erstellung von Websites, welche für mobile Geräte optimiert sind, ermöglichen. Bei der Erstellung einer mobilen Website wird zwischen folgenden Konzepten unterschieden. Die Umsetzung einer eigenständigen und unabhängigen mobilen Website stellt die klassische Lösung dar. Auf diese Weise kann der Inhalt vollständig auf den mobilen Kontext ausgerichtet werden. Ein weiteres Konzept ist Adaptive Webdesign, welches die Anpassung des Layouts einer Webpräsenz an verschiedene fix definierte Displaygrößen und Geräte ermöglicht. Ein Entwicklungsverfahren, welches in den letzten Jahren stark an Bedeutung gewonnen hat, ist Responsive Webdesign [12, S. 4]. Es ermöglicht die Erstellung einer einzigen Website, die ihr Layout dynamisch an das jeweilige Endgerät anpasst. Eine weitere Möglichkeit zur Darstellung von Inhalten auf mobilen Geräten stellen plattformspezifische Applikationen (Apps) dar. Da diese jedoch nicht im Fokus dieser Arbeit liegen, wird nicht näher auf diese Methode eingegangen.

2.1 Eigenständige mobile Website

Der erste Ansatz ist, eine weitere, für mobile Geräte optimierte Version einer Website. Mobile Webauftritte sind eigenständige Websites, welche separat konzipiert und erstellt werden und oftmals eine Ergänzung zu einer Desktop Website darstellen. Sie können sich inhaltlich von der Desktop Version unterscheiden, sind meist auf Geschwindigkeit, Übersichtlichkeit und eine mobile Navigation optimiert und sollen dabei dem Nutzungserlebnis einer nativen Applikation nahe kommen. So können beispielsweise Inhalte in der mobilen Version gekürzt oder in der Navigation nur die wichtigsten Punkte angezeigt werden. Meist ist auch eine Option implementiert, die es ermöglicht, von einem mobilen Gerät auf die Standard-Version der Website zu wechseln.

2.1.1 Aufbau und Funktion

Das Prinzip einer separaten mobilen Website beruht darauf, dass es verschiedene Ressourcen gibt, die je nach Gerät und Kontext ausgeliefert werden. Dies kann sich auf verschiedene CSS Stylesheets, Bilder und JavaScript Dateien oder auch auf völlig unterschiedliche HTML-Dokumente mit anderem Inhalt beziehen. Grundsätzlich findet die Auswahl der passenden Ressourcen automatisch statt. Die Auslieferung dieser Inhalte erfolgt, wie in [50] beschrieben, entweder über eine eigene URL (siehe Abb. 2.1), meist eine Subdomain wie *m.example.com*, oder die Inhalte sind unter derselben URL erreichbar. Die Verwendung einer Subdomain bietet den Vorteil, dass diese durch die eigenständige URL separat publik gemacht und beworben werden kann. Somit ist die mobile Website immer und geräteunabhängig über die eigene spezifische URL erreichbar.

Da eine separate mobile Website gewöhnlich eine völlig eigenständige Lösung darstellt, basiert diese auch auf einer eigenen Codebasis und ermöglicht somit die Kürzung und Umstrukturierung von Inhalten. Darüber hinaus können Bilder für kleinere Geräte entsprechend reduziert und die Navigation für eine Bedienung per Finger optimiert werden. Zweitrangige Informationen und Seitenelemente können nur auf der Desktop Website angezeigt und auf der mobilen Webpräsenz entfernt oder durch, für den mobilen Kontext, wichtigere Inhalte ersetzt werden. Das Design und die Funktionalität kann sich somit vollständig auf die spezifizierte Geräteklasse konzentrieren. Um Zugriff auf den vollen Funktionsumfang eines Webauftritts zu erhalten, sollte, wie bereits eingangs erwähnt, ein Link auf der Website platziert werden, der einen Wechsel zwischen der Desktop und der mobilen Webpräsenz ermöglicht [50].

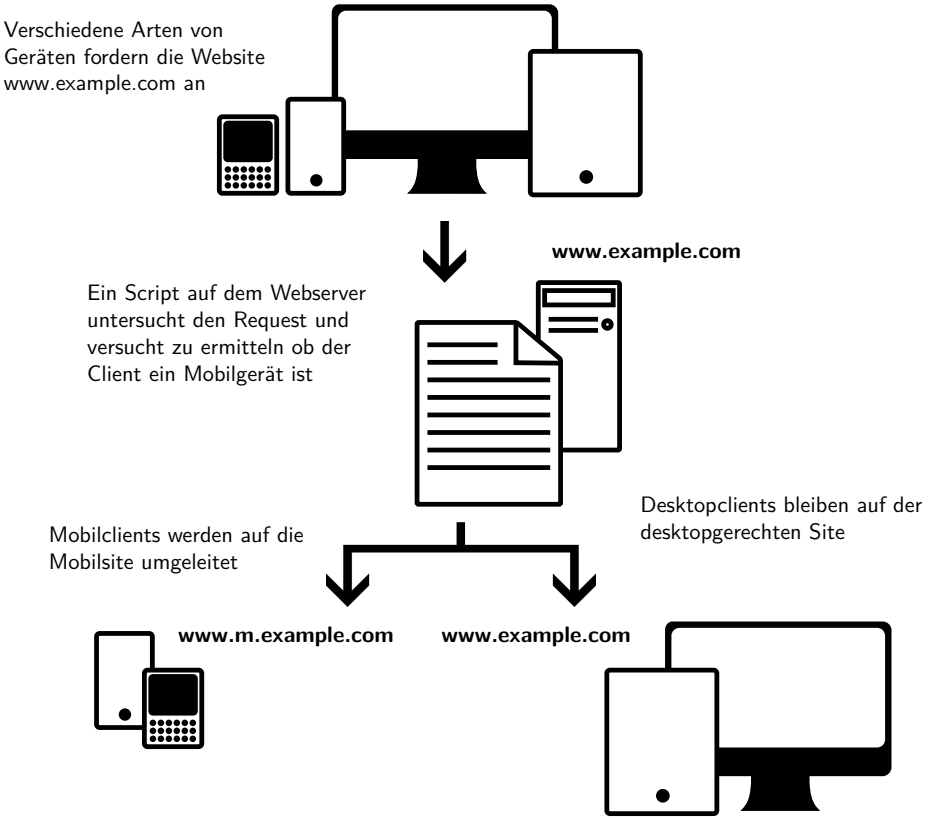


Abbildung 2.1: Beispiel für einen Workflow bei der Geräteerkennung und anschließende Weiterleitung zur entsprechenden URL und Website-Version. Bildquelle: [2, S. 96].

Serverseitige Geräteerkennung

Separate mobile Websites sind gewöhnlich über eine eigenständige Subdomain erreichbar. Diese URL kann auf der Desktop Präsenz entsprechend verlinkt werden und dem Website-Besucher somit einen Wechsel auf die mobile Version ermöglichen. Soll allerdings eine automatische Weiterleitung eines mobilen Geräts auf die entsprechend optimierte Version einer Website erfolgen, wird dies oftmals durch eine serverseitige Geräteerkennung realisiert. Diese *User-Agent-Detection* wird auch als *User-Agent-Sniffing* bezeichnet [2, S. 96–97]. Jeder Browser verfügt über eine eigene *User-Agent-Kennung*, mit der er sich gegenüber einem Webserver vorstellt. Ein *User-Agent-String* (siehe Abb. 2.2) ist ein Text der aus verschiedenen Konventionen wie Version,

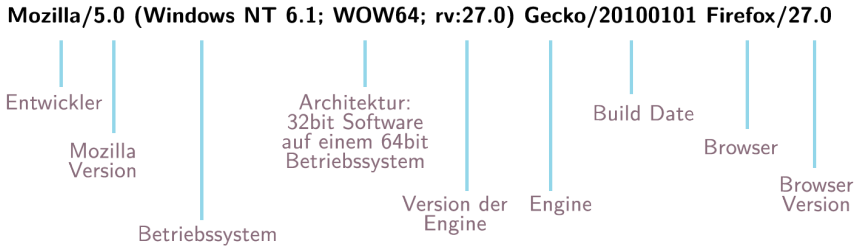


Abbildung 2.2: Grundlegender Aufbau eines *User-Agent-Strings*. Bildquelle: [59].

Plattform, Gerät und Hersteller aufgebaut ist. Ein Beispiel für einen *User-Agent-String* von einem iPhone 6 mit iOS 6 und dem Standard-Webbrowser Safari:

```
Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac OS X)
AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile
/10A5376e Safari/8536.25
```

Bei einem Chrome Desktop Browser sieht er folgendermaßen aus:

```
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/41.0.2228.0 Safari/537.36
```

Diese Beispiele zeigen, dass ein direktes und eindeutiges Interpretieren dieser Geräte- und Browserkennung, ohne eine längere Einarbeitungszeit nur schwer möglich ist. *User-Agent-Strings* bestehen nicht immer aus den gleichen Bestandteilen, da die verschiedenen Konventionen nie wirklich vereinheitlicht wurden. Wie in [50] kritisiert, geben aus historischen Gründen alle Browser Mozilla als Bezeichnung und *like Gecko* als Render-Engine an, obwohl in keinem dieser Browser eine Gecko-Rendering-Engine arbeitet. Diese falschen Angaben erschweren eine korrekte Analyse zusätzlich. Nutzer haben auch die Möglichkeit ihren Browser so zu konfigurieren, dass dieser einen anderen *User-Agent-String* sendet und sich somit als anderes Gerät ausgibt. Diese Vorgehensweise wird auch als *User-Agent-Spoofing* bezeichnet [2, S. 101]. Zur serverseitigen Geräteerkennung gibt es im Web bereits eine Reihe von Lösungen und Bibliotheken, die durch das Einbinden am eigenen Webserver eine gewünschte Weiterleitung automatisch durchführen. Die Website www.detectmobilebrowser.com bietet beispielsweise ein freies Script in verschiedenen Programmiersprachen zum Download an. Es handelt sich hierbei um eine einfache und schnelle Lösung die den *User-Agent-String* ausliest und darin nach eindeutigen Hinweisen für einen mobilen Browser sucht. Allerdings bietet diese Lösung keine Möglichkeit, genauere Informationen über ein Gerät zu ermitteln [4, S. 311]. Um mehr Details über das

anfragende Gerät zu erfahren, bieten sich *Device Description Repositories* wie WURFL[1], DeviceAtlas[2] oder 51Degrees[3] an. Diese Bibliotheken stehen ebenfalls in verschiedenen Sprachen zur Verfügung und bieten Zugriff auf mehr Details und Informationen, wie Displayauflösung, Geräteklasse, HTML Kompatibilität und weitere Hardware- und Softwareeigenschaften. Es handelt sich meist um offline Datenbanken oder cloudbasierte Dienste die den *User-Agent-String* als Kriterium für die Datenabfrage nutzen. Somit ist es auch möglich Smartphones, Tablets und auch ältere Geräte zu unterscheiden und dafür optimierte Website-Versionen zu erstellen (siehe Abb. 2.3) [50]. Allerdings können *Device Description Repositories* immer nur bekannte *User-Agents* berücksichtigen. Offline Datenbanken, die am eigenen Webserver eingebunden werden, müssen deshalb laufend auf die neueste Version aktualisiert werden. Wird ein cloudbasierter Dienst genutzt, werden die Daten auf dem Webserver des Anbieters selbstständig aktualisiert. Der Nachteil dieser Lösungen ist, dass für jeden Request eine Anfrage an den Anbieter-Server gesendet werden muss, was zu einer längeren Ladezeit für die Website führt. Allerdings arbeiten viele Cloud-Lösungen bereits mit intelligenten Cashing-Strategien, um unnötige Wartezeiten zu verkürzen [4, S. 313–315]. Sollte ein *User-Agent-String* nicht erkannt werden, muss der Besucher zu einer der verfügbaren Versionen, wie der Desktop Website, umgeleitet werden. Erfolgt einmal eine falsche Zuordnung der Geräteklasse und damit verbundene Weiterleitung zu einer falschen Version, sollte es dem Benutzer immer möglich sein, auf eine andere Version zu wechseln [50].

2.1.2 Vorteile einer mobilen Website

Bei der Erstellung einer mobilen Website-Version kann sich der Entwickler vollständig auf diese Geräteklasse und die damit verbundenen Besonderheiten und Einschränkungen konzentrieren. Da jede mobile Version gewöhnlich auf einer eigenen Codebasis basiert, müssen keine Kompromisslösungen entwickelt werden. Inhalte können sich vollständig auf den mobilen Kontext konzentrieren und Probleme, wie zu große Bilder oder zu lange Inhalte treten nicht auf. Da der inhaltliche Umfang meist wesentlich kleiner ist als bei der Desktop Version, kann die Website für mobile Netzwerkverbindungen und damit verbundene kurze Ladezeiten optimiert werden. Die Navigation kann für eine Bedienung per Finger und Touchdisplay erweitert und die Planung und Anordnung der Elemente für unterschiedliche Anzeigegeräte optimiert werden [50]. Durch die Verwendung einer Subdomain kann die Website separat publik gemacht werden. Dabei sollte es, durch die Platzierung eines Links auf der Website, einem Website-Besucher immer möglich sein, auf die jeweils andere Version der Webpräsenz zu wechseln.

[1] WURFL Website: http://wurfl.sourceforge.net
[2] DeviceAtlas Website: https://deviceatlas.com
[3] 51Degrees Website: https://51degrees.com/products/device-detection

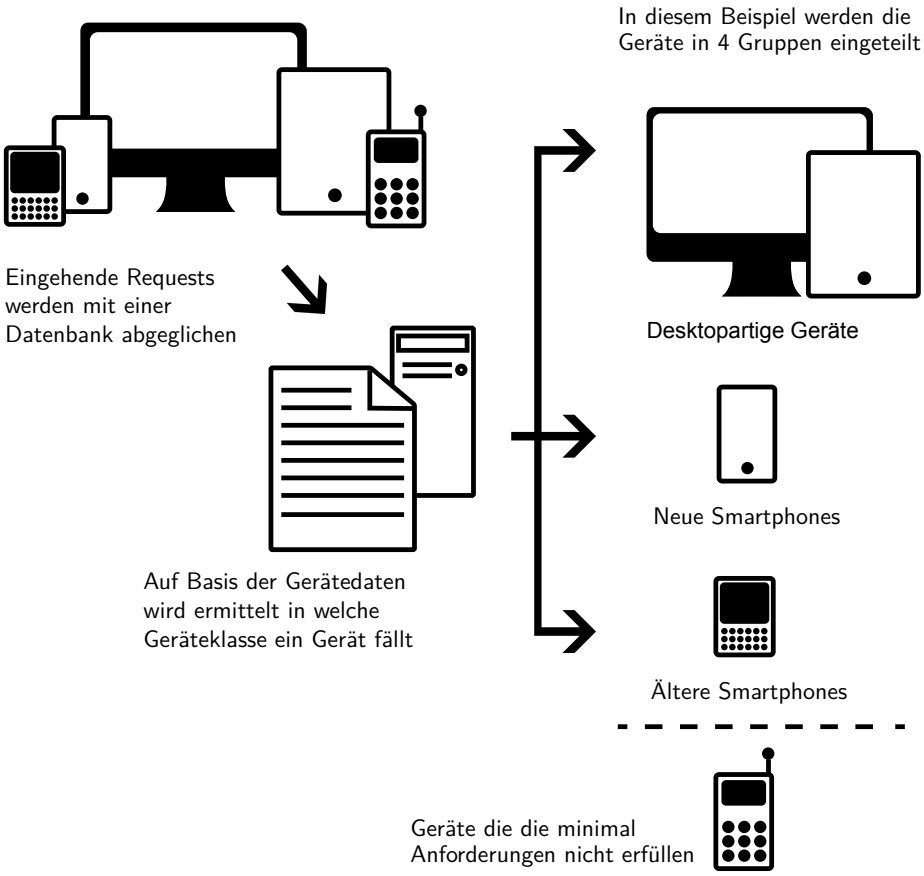


Abbildung 2.3: Hypothetisches Beispiel zur Definition von verschiedenen Website-Versionen auf Basis von festgelegten Geräteklassen. Bildquelle: [2, S. 181].

Wie in [2, S. 181] beschrieben, ist es durch die Verwendung einer serverseitigen Geräteerkennung und einer damit verbundenen Einteilung der Geräte in Geräteklassen (siehe Abb. 2.3) möglich, verschiedene optimierte Website-Versionen an unterschiedliche Endgeräte wie Tablets, Smartphones und natürlich Desktop PCs auszuliefern. Auch eine Unterstützung von älteren Geräten ist durch die Erstellung einer simplen Basisversion möglich. Dies kann vor allem für Unternehmen wie Nachrichtenagenturen oder Transportfirmen sinnvoll sein, da hier dem Informationsgehalt mehr Bedeutung als dem Design der Website zugemessen wird und die verfügbaren Inhalte so allen Zielgruppen zugänglich gemacht werden können.

Besteht bereits eine Desktop Website und es soll kein vollkommen neuer

Webauftritt gestaltet und erstellt werden, ist die Umsetzung einer separaten mobilen Website eine sehr gute Lösung. Diese ist in der Entwicklung grundsätzlich nicht so zeitaufwändig und kann völlig neu und entsprechend optimiert konzipiert werden. Soll eine mobile Website einen völlig anderen Informationsgehalt als die entsprechende Desktop Präsenz beinhalten, ist die Erstellung einer eigenen mobilen Version meist unumgänglich [50].

2.1.3 Nachteile einer mobilen Website

Ein wesentlicher Nachteil einer separaten mobilen Website ist der zusätzliche Zeitaufwand für die Entwicklung der verschiedenen Webauftritte. Sollten sich die unterschiedlichen Website-Versionen inhaltlich unterscheiden, ist auch die Wartung und Weiterentwicklung mit einem höheren Aufwand verbunden, da diese immer parallel erfolgen muss. Zudem passiert es schnell, dass Aktualisierungen nicht bei jeder Version gleichermaßen durchgeführt werden. Durch die Reduzierung und Optimierung von Inhalten auf einer mobilen Website kann die Performance erhöht und Ladezeiten verkürzt werden. Da Besucher allerdings vermehrt dieselbe Website sowohl von mobilen Geräten als auch von Desktop PCs aufrufen, erwarten sie, dass ihnen derselbe Inhalt auf all ihren Geräten vollständig zur Verfügung steht. Eine besser Lösung wäre es, sekundäre Inhalte für Nutzer mobiler Geräte primär nicht anzuzeigen, dafür aber über einen zusätzlichen Link verfügbar zu machen. Da mobile Websites oft über eine eigene URL oder eine Subdomain erreichbar sind, kann es auch bei der Weitergabe von Links per E-Mail oder Socialmedia-Plattformen zu Problemen kommen. Wird ein Link einer mobilen Version versendet und der Empfänger öffnet diesen auf einem Desktop PC, kann es zu einer falschen Darstellung der Website in der mobile Version kommen [50].

Auch die Verwendung einer serverseitigen Geräteerkennung, durch die Analyse des *User-Agent-Strings*, birgt Probleme in sich. Zum einen können *User-Agent-Strings*, wie in [2, S. 101] beschrieben, gefälscht werden und zum anderen kann eine offline Vergleichsdatenbank immer nur bekannte *User-Agents* berücksichtigen und muss somit immer aktuell gehalten werden.

2.1.4 Zusammenfassung

Der Vorteil einer mobilen Website besteht darin, dass der Inhalt vollständig auf den mobilen Kontext ausgerichtet werden kann. Das bedeutet, dass die Anordnung und Platzierung einzelner Elemente optimiert, Bilder komprimiert und Texte entsprechend kurz und prägnant gehalten werden können. Die Erstellung und Wartung von mehreren unterschiedlichen Website-Versionen stellt allerdings auch einen erheblich höheren Wartungsaufwand dar. Durch eine serverseitige Geräteerkennung ist eine Einteilung der Geräte in Geräteklassen und die Erstellung von entsprechend angepassten Website-

Versionen möglich. Die Analyse des *User-Agent-Strings* ermöglicht auch die Unterstützung von älteren Geräten. Allerdings ist die oft verwendete *User-Agent-Detection* fehleranfällig und kann durch *User-Agent-Spoofing* manipuliert werden. Soll die mobile Version einer Website einen eigenständigen Inhalt enthalten oder eine bereits bestehende, für Desktop PCs optimierte, Webpräsenz nicht neu erstellt werden, stellt eine separate mobile Website eine ideale Lösung dar.

2.2 Adaptive Webdesign

Der Begriff Adaptive Webdesign wurde von Aaron Gustafson, der ein Buch mit dem gleichnamigen Titel verfasst hat, geprägt [6]. Der Autor meint damit, dass sich eine Website an die Fähigkeiten des Nutzergeräts anpassen kann. Grundsätzlich wird zwischen den Begriffen *responsive* (siehe Abschnitt 2.3), der reaktionsfähig bedeutet und *adaptive*, also anpassungsfähig, unterschieden. Einer anpassungsfähigen Website fehlt dabei ein flexibles Grid, denn es wird an fix definierten Umbruchpunkten ein, an die jeweilige Displaygröße angepasstes, Layout ausgegeben. Basierend auf dieser Tatsache sind anpassungsfähige Layouts starrer und können sich nicht stufenlos an jede Größenänderung des Darstellungsfensters anpassen [21, S. 14]. Wird ein adaptives Layout noch um zusätzliche serverseitige Abfragen, wie sie von separaten mobilen Websites verwendet werden, erweitert, spricht man von Adaptive Webdesign. Aufgrund dieser Tatsache kann Adaptive Webdesign als ein umfangreicheres Konzept als Responsive Webdesign, das keine serverseitige Logik verwendet, gesehen werden (siehe Abb. 2.4). Eine Kombination von Responsive Webdesign und den serverseitigen Komponenten aus Adaptive Webdesign wird auch als RESS (Responsive Design + Server Side Components, siehe Abschnitt 3.3) bezeichnet [54].

2.2.1 Aufbau und Funktion

Ist von einem adaptiven, anpassungsfähigen Layout die Rede, handelt es sich um ein für verschiedene fix definierte Displaygrößen optimiertes Web-Layout. Diese Fixed Layouts orientieren sich grundsätzlich an sehr häufig auftretenden Displaygrößen (z.B. iPhone für Smartphone und iPad für Tablet) [54]. Der Kern dieses Layouts ist ein starres Raster in Kombination mit *Media Queries*. Wie in [2, S. 12–14] beschrieben, handelt es sich bei einer *Media Query* um eine CSS Technik, mit der innerhalb eines CSS Stylesheets aktuelle Werte der Medieneigenschaften des verwendeten Browsers, wie beispielsweise die Auflösung des Browserfensters, abgefragt werden können. Trifft eine *Media Query* zu, werden die entsprechend definierten CSS Regeln angewandt. Eine *Media Query* wird in einer CSS Datei folgendermaßen definiert:

```
@media only screen {/* CSS Regel für Bildschirm */}
```

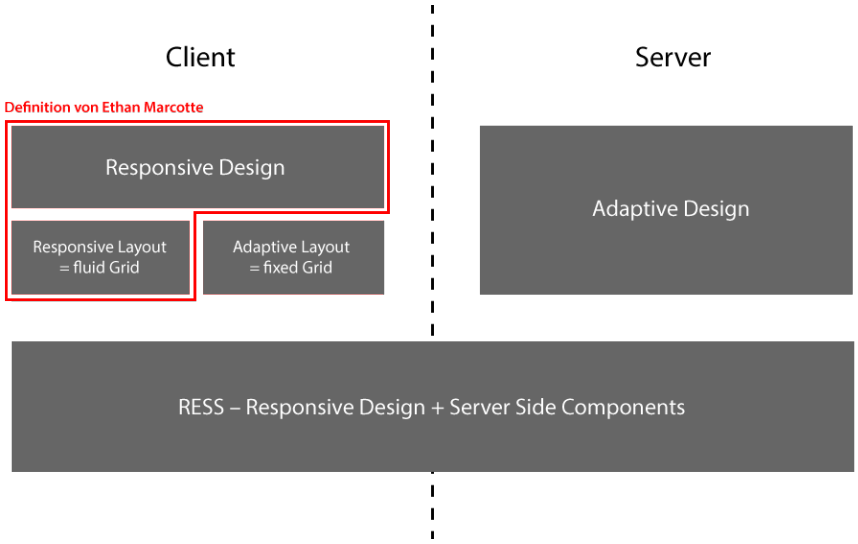


Abbildung 2.4: Eine Gegenüberstellung von Responsive Webdesign, das auf clientseitigen Techniken basiert und Adaptive Webdesign, das zusätzlich zu einem adaptiven Layout auf serverseitige Logik setzt. Eine separate mobile Website würde hier auf der Serverseite angeführt werden. Bildquelle: [54].

Dieses simple Beispiel zeigt eine Regel, die nur angewandt wird wenn der Inhalt auf einem Display (*screen*) ausgegeben wird. Das bedeutet, dass die Regel nicht ausgeführt wird, wenn eine Website beispielsweise gedruckt wird. Ein weiteres Beispiel ist eine *Media Query* die das Abfragen von bestimmten Browser-Informationen ermöglicht. *Media Queries* sind eine Weiterentwicklung der Medientypen der CSS2-Spezifikation. Medientypen können allerdings nur Geräte, wie Display oder Drucker, ansprechen und keine Geräteeigenschaften [56]. So genannte Medieneigenschaften sind beispielsweise die Breite des Browserfensters (*width*) oder die Ausrichtung des mobilen Geräts (*orientation*). Die folgende *Media Query* setzt sich aus einem Medientyp und einer Medieneigenschaft zusammen:

```
@media only screen and (max-width: 480px) { /* Hier werden die Regeln
    definiert */ }
```

Bei diesem Beispiel wird die CSS Regel nur angewandt wenn der Inhalt auf einem Display angezeigt wird und das Browserfenster eine maximale Breite von 480 Pixel aufweist. Durch die Definition von unterschiedlichen Regeln für verschiedene Auflösungen können so genannte *Breakpoints* festgelegt werden. Dies ermöglicht eine Änderung des Layouts an diesen definierten Punkten, was durch die Gültigkeit von jeweils anderen CSS Eigenschaften erreicht wird [2, S. 87]. Eine andere Möglichkeit ist die Verwendung von Medientypen in einem *<link>* Element:

```
<link rel="stylesheet" type="text/css" href="print.css" media="print">
```

Durch einen solchen Verweis ist es möglich, ein Druck-Stylesheet (*print*) einzubinden. Das bedeutet, dass dieses Stylesheet nur genutzt wird, wenn der Inhalt ausgedruckt wird.

Bei einem Adaptive Layout werden verschiedene Website-Layouts für mehrere exakte *Viewports* entwickelt. Der Begriff *Viewport* beschreibt den sichtbaren Bereich in einem Browserfenster. Grundsätzlich werden Layouts für Desktop, Tablet und Smartphone erstellt. Wie bereits erwähnt orientieren sich die dafür definierten Abmessungen an bestimmten, weit verbreiteten, Geräten. *Media Queries* ermöglichen im Anschluss die passende Ansicht für die Displaygröße des Seitenbesuchers zu laden. Die Auflösungen der *Media Queries* orientieren sich dabei ebenfalls an den festgelegten Layout-*Breakpoints*. Dies führt zu einem adaptiven Layout das sich an ausgewählte Displaygrößen anpasst [54].

Adaptive Webdesign erweitert ein Adaptive Layout noch zusätzlich um serverseitige Abfragen. Beim Adaptive Design steht weniger das Website-Design im Vordergrund, sondern vielmehr das Ausgabegerät. Es werden die Eigenheiten und die Auflösung des spezifischen Geräts adressiert. Der Server passt in diesem Fall die Website und den damit verbundenen Inhalt an das anfragende Gerät an. Dies ermöglicht es, an jede Geräteklasse (z.B. Mobile, Tablet, Desktop), ein vollständig eigenes Template auszuliefern, das bestmöglich auf die Bedürfnisse des Website-Besuchers zugeschnitten ist. Bilder und andere Medienelemente können an die Auflösung des Endgeräts angepasst und dafür optimiert werden [71]. Somit erhält ein Gerät mit einem kleineren Display auch eine kleinere Bilddatei und ein hochauflösender Desktop Monitor eine hochauflösende Grafik. Die Erstellung von verschiedenen Templates, die je nach Auflösung ausgeliefert werden, bedeutet allerdings auch einen deutlichen Mehraufwand, der dem von separaten mobilen Websites (siehe Abschnitt 2.1) nahe kommt. So muss ebenfalls mit der, bei mobilen Websites vermehrt eingesetzten, serverseitigen Geräteerkennung (siehe Abschnitt 2.1.1) gearbeitet werden, um die Templates an die entsprechenden Geräte auszuliefern. Aufgrund der serverseitigen Optimierung, für die jeweilige Geräteklasse, führen Lösungen dieser Art aber zu einer sehr guten Performance [54].

Wie in [6] beschrieben, spielt bei der Erstellung einer Website mit Adaptive Webdesign, auch *Progressive Enhancement* eine wichtige Rolle (siehe Abb. 2.5). Der primäre Fokus liegt hier auf dem Nutzer und dem Ausgabegerät und nicht auf dem Design der Website. Somit können grundlegende Informationen, die eine Website ausmachen, allen Nutzern und Geräten zur Verfügung gestellt werden. Zusätzliche Funktionalitäten und Designelemente werden erst eingebunden wenn ein Gerät diese auch unterstützt.

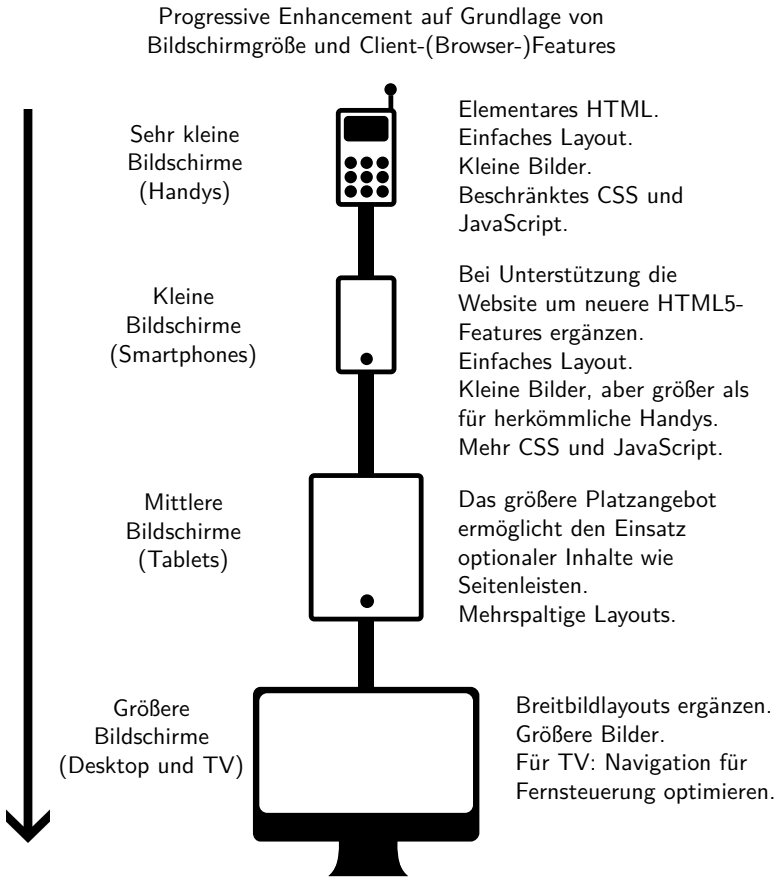


Abbildung 2.5: Adaptive Webdesign ermöglicht die Erstellung von mehreren auflösungsabhängigen Website-Layouts die je nach Endgerät ausgeliefert werden. Bildquelle: [2, S. 56].

2.2.2 Vorteile von Adaptive Webdesign

Ein wesentlicher Vorteil von Adaptive Webdesign ist, dass es viel gestalterischen Freiraum bietet. Durch die Verwendung eines adaptiven Layouts und damit verbundenen starren Rasters kann das Design für verschiedene fixe Auflösungen angepasst und optimiert werden. Folglich kann im Entwicklungsprozess auch mit Mockups und Skizzen gearbeitet werden, da feste Abmessungen für die einzelnen Website-Versionen existieren. Dies ermöglicht eine zeitsparende und technisch unkomplizierte Umsetzung [54].

Durch die serverseitigen Erweiterungen in Adaptive Webdesign, können einzelne Website-Versionen noch besser an die jeweilige Geräteklasse angepasst werden. Diese zusätzliche Komponente ermöglicht die Erstellung und

Optimierung eines eigenen Templates, für jede Geräteklasse (z.B. Mobile, Tablet, Desktop). Dementsprechend kann jedes Template bestmöglich auf die Bedürfnisse des Website-Besuchers und der Geräteklasse zugeschnitten werden. Durch eine Optimierung von Bildern und anderen Medienelementen, sind auch wesentliche Performanceverbesserungen möglich [71].

Soll eine bereits bestehende Website im Nachhinein für mobile Endgeräte optimiert werden, ist Adaptive Webdesign eine gute und zeitsparende Möglichkeit. Durch die Verwendung von verschiedenen optimierten Templates, muss die bestehende Desktop Website nicht völlig neu konzipiert werden.

2.2.3 Nachteile von Adaptive Webdesign

Da eine Website mit einem adaptiven Layout nur an bestimmte exakte *Viewports* angepasst wird, kommt es häufig zu Fehldarstellungen auf abweichenden Endgeräten. Das bedeutet, dass eine Website, sobald die Breite des Browserfensters einen definierten *Breakpoint* unter- oder überschreitet, auf ein anderes Layout umspringt und in diesem einrastet. Dementsprechend bleibt auf kleineren Displays oft verfügbarer Platz ungenutzt, da die Website nur für fixe Auflösungen konzipiert wird und sich nicht dynamisch auf unterschiedliche Fensterbreiten anpassen kann. Um dies zu vermeiden sollte, um die relevanten Geräte und Abmessungen zu bestimmen, vor der Umsetzung eine aufwändige Zielgruppenanalyse durchgeführt werden. So können die wichtigsten Geräte und Auflösungen ermittelt und die Website-Versionen für diese angepasst und optimiert werden [54].

Auch bei Adaptive Webdesign kann die Verwendung einer serverseitigen Geräteerkennung zu Problemen führen. Wird der *User-Agent-String* eines Geräts nicht erkannt, kann ein falsches Template eingebunden und angezeigt werden. Daher sollte es einem Besucher auch bei einer adaptiven Webpräsenz immer möglich sein, auf eine andere optimierte Version der Website zu wechseln [50].

2.2.4 Zusammenfassung

Adaptive Webdesign ermöglicht durch ein adaptives Layout die Anpassung einer einzigen Website an verschiedene fix definierte Auflösungen und Geräte. So wird an fix definierten Umbruchpunkten ein, an die jeweilige Displaygröße angepasstes, Layout ausgegeben. Durch serverseitige Erweiterungen können Bilder und andere Medienelemente für jedes Endgerät optimiert und ausgetauscht werden. Ein adaptives Layout erfordert allerdings die Definition von fixen *Breakpoints* und kann sich somit nicht stufenlos an die Breite des Darstellungsfensters anpassen, sondern nur auf bestimmte fix definierte Auflösungen reagieren. Dies führt oftmals zu falschen Darstellungen auf Geräten deren Abmessungen und Auflösungen außerhalb der definierten Umbruchpunkte liegen.

Wird eine bestehende Website, ohne einer kompletten Neugestaltung, für mobile Geräte optimiert, bietet Adaptive Webdesign eine gute Lösung, die eine zeitsparende und technisch unkomplizierte Umsetzung ermöglicht.

2.3 Responsive Webdesign

Responsive Webdesign ist ein Verfahren, welches die Entwicklung einer einzigen Website, die auf allen gängigen Endgeräten gleichermaßen funktioniert, ermöglicht. Der Begriff Responsive Webdesign erschien erstmals im Jahre 2010 in einem Artikel von Ethan Marcotte [63]. In diesem innovativen Artikel beschreibt Marcotte Responsive Webdesign, als einen Webdesign Ansatz, der sich auf die Entwicklung von Websites, die ein optimales Betrachtungserlebnis ermöglichen, fokussiert. Das bedeutet, dass eine Website, die auf dem Responsive Webdesign Konzept basiert, eine gute Lesbarkeit und Navigation über eine Reihe von verschiedenen Geräten und Displaygrößen hinweg, bieten soll. Es basiert dabei auf Techniken wie einem flexiblen (fluid) Grid, *Media Queries* und flexiblen Bildern. *Media Queries* (siehe Abschnitt 2.2.1) ermöglichen die Gültigkeit und Anwendung von definierten CSS Regeln und Formatierungen, in Abhängigkeit von bestimmten Browserbedingungen, wie der aktuellen Fenstergröße oder der Geräteausrichtung [2, S. 10–11]. Basierend auf dieser Technik können mehrere *Breakpoints* definiert werden und eine Website auf einem kleinen Display einspaltig und auf einem größeren Monitor, mit viel Platz, mehrspaltig angezeigt werden (siehe Abb. 2.6). Ein responsives Layout kann somit als eine Weiterentwicklung eines adaptiven fixen Layouts betrachtet werden. Die von Marcotte definierten Techniken sind alle clientseitig. Das bedeutet, dass Responsive Webdesign im Gegensatz zu Adaptive Webdesign (siehe Abschnitt 2.2) auf keiner serverseitigen Logik basiert. Adaptive Webdesign kann somit als umfangreicheres Konzept gesehen werden, da es sowohl auf ein clientseitiges adaptives Layout als auch auf eine serverseitige Logik setzt (siehe Abb. 2.4). Wird Responsive Webdesign mit serverseitigen Komponenten kombiniert, wird dies meist als RESS (Responsive Design + Server Side Components, siehe Abschnitt 3.3) bezeichnet [54].

2.3.1 Aufbau und Funktion

Mobile Browser stellen beim Versuch eine möglichst gute Erfahrung zu gewährleisten, Websites meist in der Breite eines Desktop-Bildschirms dar. Diese ist in der Regel 980 Pixel, wobei sich die Auflösung von Gerät zu Gerät unterscheiden kann. Im Anschluss skaliert ein Browser die Schriftgröße und andere Inhalte um den Bildschirm des mobilen Geräts zu füllen. Dies führt meist zu einer inkonsistenten Darstellung und umständlichen Interaktion. Websites die für verschiedene Geräte optimiert werden, müssen daher das *viewport-Meta-Tag* im Kopfbereich (Header) der HTML-Datei einbin-

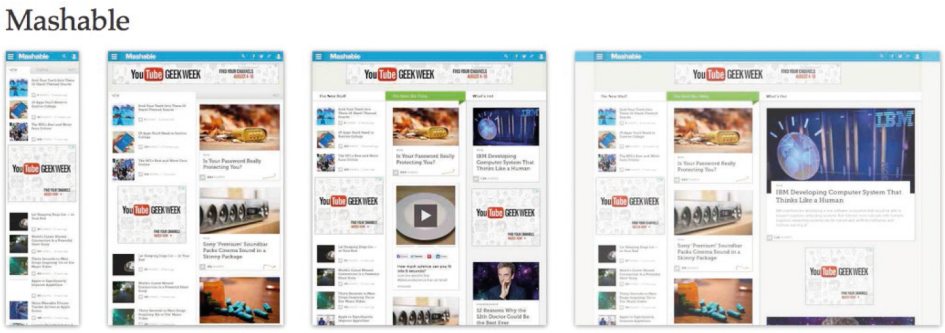


Abbildung 2.6: Wie die Website www.mashable.com zeigt, ermöglicht ein flexibles Grid eine dynamische Anpassung an die verfügbare Fensterbreite. Bildquelle: [22].

den. Das *viewport-Meta-Tag* gibt dem Browser Anweisungen dazu, wie er Abmessungen und Skalierung der Website steuern soll. Da mobile Geräte mit verschiedenen Auflösungen und Abmessungen existieren, empfiehlt es sich dem Browser diese Entscheidung zu überlassen:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Wie in [61] beschrieben, wird durch das Setzen des entsprechenden Meta-Wertes *width=device-width*, der Website die Anweisung gegeben, die Breite des Displays in geräteunabhängigen Pixel zu nutzen. Dementsprechend können Inhalte neu angeordnet werden und sich an die jeweilige Displaygröße anpassen. Der Wert *initial-scale=1.0* entspricht immer einer Vollbild-Darstellung von 100 % [13]. Dies verhindert ein Zoomen und eine damit verbundene falsche Breite der Website, die beim Drehen eines mobilen Geräts in das Querformat auftreten kann. Somit ist es der Website möglich die volle Breite des Displays im Querformat zu nutzen [61]. Über das Attribut *maximum-scale=1.0* kann zusätzlich das Zoomen auf der Website deaktiviert werden. Dieser Wert kann allerdings zu einer fehlerhaften Darstellung bei der Nutzung eines Geräts im Querformat führen und sollte deshalb, wie in [2, S. 72–73] empfohlen, nur in speziellen Fällen gesetzt werden.

Responsive Webdesign setzt, anders als Adaptive Webdesign (siehe Abschnitt 2.2), auf ein flexibles Grid. Ein flexibles Layout nutzt für Größenangaben proportionale Einheiten anstatt Pixel und ermöglicht so eine dynamische Anpassung an die verfügbare Fensterbreite. Soll ein fixes pixelbasiertes Layout durch ein proportionales flexibles Layout ersetzt werden, muss dabei folgendermaßen vorgegangen werden (siehe Abb. 2.7). Zuerst wird die gesamte Breite der Website (960 Pixel) durch 100 % ersetzt. Anschließend können die einzelnen Spalten der Website auf diese 100 % aufgeteilt werden. Ändert sich die Breite des Browserfensters, nehmen die Spalten immer den gesamten verfügbaren Platz ein und der Inhalt wird weder beschnitten, noch

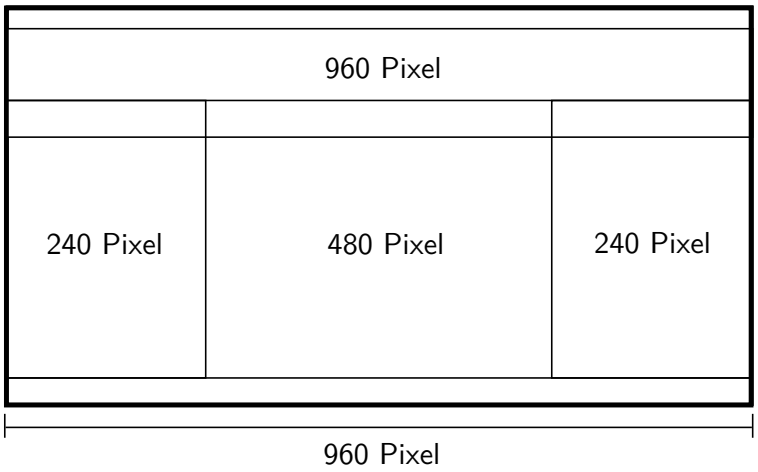


Abbildung 2.7: Hypothetisches Beispiel für ein Website-Layout mit fixen Breitenangaben in Pixel. Bildquelle: [2, S. 28].

bleibt verfügbarer Platz ungenutzt. Wie in [62] beschrieben, kann mit der Gleichung

$$\frac{Ziel}{Kontext} = Ergebnis \tag{2.1}$$

von einem pixelbasierten Layout auf ein flexibles Layout mit prozentualen Werten umgerechnet werden. Für eine Spalte mit 240 Pixel (siehe Abb. 2.7) würde die Berechnung 240 Pixel / 960 Pixel = 25 % lauten. Dabei wird das umschließende Element (960 Pixel) als *Kontext* und das zu berechnende Element (240 Pixel) als *Ziel* in die Gleichung 2.1 eingesetzt. Basierend auf dieser Berechnung nehmen die äußeren beiden Spalten jeweils 25 % der Gesamtbreite ein. Der Content-Bereich in der Mitte nutzt die verbleibenden 50 %. Header- und Footer-Bereich nehmen in diesem Beispiel die gesamte Breite der Website, sprich 100 %, ein. Um eine lückenlose Anpassung des Layouts, über alle Bildschirmgrößen hinweg zu erreichen, sollten auch Abstände (*Margins*[4] und *Paddings*[5]) von Pixel auf Prozent umgerechnet werden.

Proportionale Angaben beziehen sich allerdings nicht nur auf das Layout einer Website. Auch Schriftgrößen können mit der genannten Gleichung 2.1 umgerechnet werden. Das Ergebnis wird im Anschluss in *em* oder ebenfalls als prozentualer Wert angegeben. Im *<body>* Element wird die Basisschriftgröße für eine Website definiert. Wird diese auf 100 % gesetzt, werden alle Elemente auf der Website in einem relativen Verhältnis zu der Standardschriftgröße des Browsers skaliert. Diese ist in den meisten Fällen 16 Pixel.

[4] Außenrand oder -abstand: ein erzwungener Leerraum zwischen dem aktuellen Element und seinen Eltern- und Nachbarelementen [23].

[5] Innenabstand: ein erzwungener Leerraum zwischen dem Inhalt eines Elements und seinem eigenen Elementrand [24].

Somit würde das Ergebnis 1 *em* = 100 % = 12 pt = 16 Pixel lauten. Soll die Schriftgröße einer Überschrift (*<h1>*) 24 Pixel betragen, kann dies ebenfalls durch die bereits genannte Gleichung 2.1 berechnet werden. Für dieses Beispiel würde die Berechnung 24 Pixel / 16 Pixel = 1,5 *em* lauten. Das bedeutet, dass eine Überschrift mit 24 Pixel 1,5 mal so groß wie die Basisschriftart ist [62]. Dies kann im Stylesheet folgendermaßen angegeben werden:

```
h1 {
  font-size: 1.5em;    /* 24px / 16px = 1.5em */
}
```

Der Einsatz von Prozent und *em* ist eine für Schriften ideale Kombination die auf allen Browsern und Plattformen eine optimale Darstellung ermöglicht [2, S. 35].

Auch Bilder und andere eingebettete Medien müssen für ein flexibles Grid angepasst und entsprechend eingebunden werden. Für eine optimale Darstellung müssen diese Medienelemente proportional mit dem Layout mitskalieren. Allerdings gibt es keine Möglichkeit, die Größe in proportionalen Einheiten wie Prozentangaben anzugeben [2, S. 32–35]. Um dennoch zu verhindern, dass ein Bild oder anderes eingebettetes Medienobjekt breiter wird als das umschließende Element, kann folgende CSS Regel angewendet werden:

```
img, object { max-width: 100%; }
```

Mit dieser Regel wird das Bild auf 100 % der Breite des umschließenden Elements begrenzt. Dementsprechend sollen die Attribute *height* und *width* nicht mehr gesetzt werden, da diese einem Element eine fixe Größe zuordnen und somit die dynamische Skalierung eines Medienelements verhindern würden. Da die oben genannte CSS Regel nur das *width*-Attribut überschreibt, wird bei einer vorhandenen definierten Höhe in Pixel nur die Breite dynamisch angepasst. Dies kann zu Verzerrungen führen, da die Höhe des Elements immer identisch bleibt. Da Responsive Webdesign auf keiner serverseitigen Komponente basiert, können Bilder und andere Elemente, anders als bei Adaptive Webdesign (siehe Abschnitt 2.2), nicht ausgetauscht werden. Dementsprechend wird ein Bild auf einem kleineren Display nur skaliert und nicht optimiert und weist somit immer noch dieselbe Dateigröße auf. Zusätzlich wird bei der Erstellung einer responsiven Website oft mit der Konzeption der Desktop Präsenz begonnen. Wird eine solche Website im Anschluss auf einem mobilen Gerät aufgerufen, kann dies zu Performance-Problemen führen, da die mobile Website im Kern immer noch eine große und meist überladene Desktop Site ist [2, S. 56–57]. Wird ein Webauftritt mit Responsive Webdesign von Grund auf neu gestaltet, sollte deshalb der *Mobile First* Ansatz gewählt werden. Mobilzentriertes Webdesign bedeutet, dass bei der Konzeption der Website, mit der mobilen Version begonnen wird. Im Anschluss wird mit *Progressive Enhancement* auf diese Basisversion aufgebaut (siehe auch Abschnitt 2.2). Im Designprozess kann die Web-

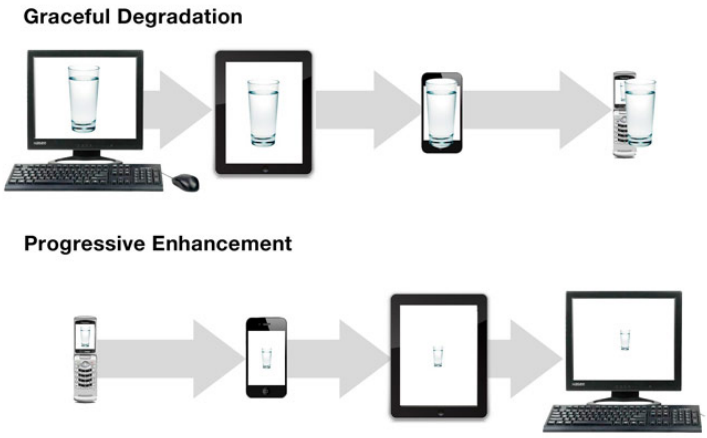


Abbildung 2.8: Die Degradierung einer Desktop Website ist meist ein komplexer und schwieriger Prozess. Durch *Mobile First* und *Progressive Enhancement* beschränkt sich der Inhalt einer Website auf das Wesentliche und wirkt auch auf einem Desktop Monitor nicht überladen. Bildquelle: [51].

site somit auf den wesentlichen Informationsgehalt reduziert und optimiert werden. Auf dieser Grundlage wird im Anschluss bis zur Desktop Version aufgebaut. Somit stellt der Informationsgehalt der mobilen Version das Kernelement der Website dar. Durch *Progressive Enhancement* erfolgt im Anschluss der schichtweise Aufbau. Zusätzliche Funktionalitäten und Designelemente werden nur eingebunden und angezeigt, wenn ein Browser diese auch unterstützt. Die wesentlichen Inhalte stehen somit jedem Benutzer zur Verfügung. Diese Vorgehensweise erlaubt es, dass eine Website auf fast allen existierenden Browsern ein entsprechendes Nutzungserlebnis bietet. Da bei der Erstellung der Website mit der mobilen Version begonnen wird, entfällt die Rationalisierung und Degradierung der Desktop Website. Basierend auf dieser Tatsache muss im Anschluss nicht entschieden werden, welche Inhalte von der Desktop Version auf die mobile Website übernommen und welche gestrichen werden (siehe Abb. 2.8). Diese Vorgehensweise verhindert eine überfüllte und viel zu lange mobile Website, die durch eine Umstrukturierung eines überladenen Desktop-Layouts entstehen würde [51]. Soll eine Website mit *Mobile First* und *Progressive Enhancement* konzipiert werden, wird die elementare Ausgangsbasis der Website, ohne *Media Queries* erstellt, da diese von älteren Browsern nicht unterstützt werden. Der Schichtweise Aufbau der Website und die Umleitung auf die jeweils höhere Version wird dann über *Media Queries* realisiert [2, S. 61].

Da der Fokus dieser Arbeit auf der Verbesserung und Erweiterung von Responsive Webdesign liegt, werden die Vor- und Nachteile dieses Konzeptes in den folgenden Abschnitten ausführlicher behandelt und auch auf die

Unterschiede zu den beiden zuvor vorgestellten Verfahren eingegangen.

2.3.2 Vorteile von Responsive Webdesign

Marcotte prägte den Term Responsive Webdesign im Jahre 2010. Seit dem hat sich dieses Konzept rasch weiterentwickelt. Responsive Design bedeutet heute nicht mehr nur das sich das Layout einer Website automatisch an die verfügbare Displaygröße anpasst. Es basiert darauf ein responsives Nutzungserlebnis zu bieten. Das bedeutet, die vorhandenen Geräteinformationen bestmöglich zu nutzen um die Website und das Nutzungserlebnis an die Umstände des Website-Besuchers optimal anzupassen. So kann, zum Beispiel die Hintergrund- und Textfarbe einer Website angepasst werden, um die Lesbarkeit des Inhalts, je nach Tageszeit, zu verbessern [12, S. 8–11]. Betrachtet man den raschen Entwicklungsprozess auf diesem Gebiet, gibt es eine Reihe von Argumenten die, bei der Neugestaltung einer Website, für die Nutzung von Responsive Webdesign sprechen.

Inhalt

Es existiert der Mythos, dass der Inhalt und der Funktionsumfang einer Website, für mobile Geräte und kleine Displays gekürzt und beschränkt werden sollte. Diese Aussage stammt aus einer Zeit, bevor Smartphones und Tablets das Web eroberten. Für ältere Geräte mit kleineren Bildschirmen und alten limitierten Browsern, ist diese Aussage auch durchaus nachvollziehbar, da diese nur für das Browsen auf elementaren Websites ausgelegt sind. Allerdings kann ihr durch die rasche Verbreitung des Smartphones heute keine wesentliche Bedeutung mehr zugemessen werden. Besitzer von mobilen Geräten nutzen das mobile Web heutzutage an jedem Ort und in jeder Lebenssituation und erwarten eine voll funktionsfähige Website auf ihrem Gerät. Das bedeutet, dass Besucher von mobilen Geräten denselben Informationsgehalt und dieselben Funktionalitäten, die ihnen auf der Desktop Website zur Verfügung stehen, auch von einer mobilen Version erwarten. Durch serverseitige Erweiterungen und eine damit verbundene häufige Reduzierung des Inhalts und Funktionsumfangs, trifft dies bei Adaptive Webdesign und auch separaten mobilen Websites oftmals nicht zu [49].

Wie bereits erwähnt, ermöglicht Responsive Webdesign die Erstellung einer einzigen Website, die sich automatisch an die jeweilige Displaygröße anpasst. Durch eine Umstrukturierung des Layouts steht derselbe Informationsgehalt allen Website-Besuchern zur Verfügung.

URL Struktur

Eine konsistente URL Struktur ist heutzutage wichtiger denn je. Eine Website bzw. ihr Inhalt wird auf verschiedene Arten, von verschiedenen Orten

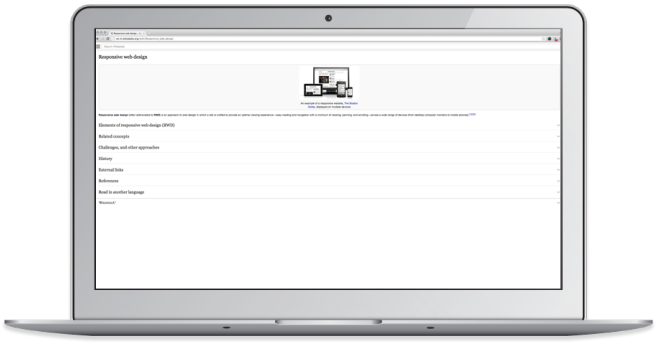


Abbildung 2.9: Die mobile Version der Website Wikipedia (www.wikipedia.org) auf einem großen Monitor. Bildquelle: [49].

und über unterschiedliche Geräte hinweg geteilt. Wird die URL einer separaten mobilen Website auf einem mobilen Gerät, beispielsweise per E-Mail oder einer Socialmedia-Plattform, an einen anderen Nutzer gesendet und der Empfänger öffnet diese auf einem Desktop PC, kann es zu einer falschen Darstellung der Website kommen. Sollte der Empfänger nicht automatisch auf die Desktop Version der Website umgeleitet werden, kommt es meist zu einer Wiedergabe eines einspaltigen, verzerrten mobilen Layouts (siehe Abb. 2.9) [49]. Die Nutzung eines solchen mobilen Layouts auf einem größeren Monitor ist sehr unkomfortabel und sollte vermieden werden. Responsive Webdesign ist eine innovative Lösung zur Realisierung des so genannten *One Web*. Dieses basiert auf der Tatsache, dass zur Auslieferung des gleichen Inhalts an alle gängigen Geräte dieselbe URL verwendet wird [9, S. 6]. Durch Responsive Webdesign und oftmals auch Adaptive Webdesign werden Probleme, die beim Teilen einer Website auftreten können vermieden, da sich das Layout automatisch an die neuen Gegebenheiten anpasst. Die URL Struktur ist auch für Suchmaschinen von besonderer Bedeutung.

Suchmaschinenoptimierung (SEO)

Separate mobile Websites nutzen Redirects und unterschiedliche URL's, wie *m.example.com*, für die verschiedenen Website-Versionen. Darüber hinaus ist es durchaus möglich, dass eine mobile Website eine völlig andere Seitenstruktur als die Desktop Präsenz aufweist. Dies führt meist zu Problemen, da Suchmaschinen nicht erkennen können, dass zwei URL's denselben Inhalt ausliefern und somit als die gleiche Seite indexiert und gewertet werden müssen. Die Suchmaschine Google favorisiert beispielsweise für die Suchmaschinenoptimierung und eine dementsprechend gute Bewertung, Responsive

Webdesign. Responsive Webdesign ermöglicht die Auslieferung einer einzigen Website und aller Inhalte über eine identische URL. Trotzdem werden die Informationen entsprechend für mobile Geräte umstrukturiert und aufbereitet, was in Suchmaschinen zu einer Markierung als, für mobile Geräte geeignete, Website führt [49].

Weiterentwicklung und Wartung

Wie in [49] beschrieben, werden aktuelle Inhalte, sowie eine permanente Weiterentwicklung und damit verbundene agile Webauftritte immer wichtiger. Websites sollen heute auf Trends und neue Technologien reagieren und dementsprechend laufend adaptiert werden. Im Gegensatz zu separaten mobilen Websites oder Adaptive Webdesign mit multiplen Templates, muss die Weiterentwicklung einer Website, mit Responsive Webdesign, nur an einer Code-Ausgangsbasis durchgeführt werden. Dies ermöglicht es einem Entwickler sich auf eine optimale und zeitsparende Implementierung und Optimierung für die entsprechenden *Breakpoints* zu fokussieren.

Des Weiteren ist bei der Erstellung einer Website auch der darauffolgende Wartungsaufwand von großer Bedeutung. Eine eigenständige mobile Website, sowie verschiedene Website-Templates, müssen immer separat zur Desktop Präsenz gewartet werden. Dies ist mit einem hohen Aufwand verbunden, da Änderungen überall durchgeführt werden sollten. Unterscheiden sich die verschiedenen Website-Versionen vom Aufbau und Informationsgehalt, ist eine regelmäßige Wartung mit einem noch höheren Zeitaufwand und damit einhergehenden hohen Kosten verbunden. Bei Responsive Webdesign entfällt dieser Aufwand, da eine einzige flexible Website für alle Geräte erstellt wird. Der Informationsgehalt und Inhalt bleibt, unabhängig von dem verwendeten Gerät, identisch. Das bedeutet, dass die Wartung und Aktualisierung von Inhalten nur einmal, auf einer einzigen Webpräsenz, durchgeführt werden muss.

Zukunftsorientiert

Separate mobile Websites und Webauftritte die auf Adaptive Webdesign basieren, können immer nur für bereits bestehende Geräte entwickelt werden. Diese Lösungen setzen auf fixe Layouts und werden meist für aktuell gängige Auflösungen und Geräteklassen konzipiert und optimiert. Somit ist eine Anpassung für neue Geräte permanent erforderlich und mit einem großen Entwicklungs- und Testaufwand verbunden.

Im Gegensatz dazu ist Responsive Webdesign, da es auf einem flexiblen Layout basiert, eine zukunftsorientierte Lösung, die sich dynamisch an die Gegebenheiten eines Endgeräts anpassen kann. Wird eine Website mit diesem Ansatz konzipiert und erstellt, bietet sie auf allen gängigen und auch zukünftigen Displaygrößen eine entsprechend optimierte Darstellung und

führt somit auch zu einem wesentlich geringeren Änderungs- und Testaufwand [10, S. 25].

Allerdings bietet Responsive Webdesign nicht nur Vorteile sondern ist auch mit diversen Nachteilen und Einschränkungen verbunden.

2.3.3 Nachteile von Responsive Webdesign

Bei der Entwicklung einer modernen responsiven Website sollten neben der Displaygröße auch andere Faktoren, wie die verfügbare Netzwerkverbindung, eine Unterstützung von bestimmten Seitenbestandteilen oder auch die HTML und CSS Kompatibilität des Browsers berücksichtigt werden. Um eine optimale Performance und ein damit verbundenes gutes Nutzungserlebnis zu bieten, müssen Websitekomponenten, wie Navigation oder Medienelemente, ausgetauscht und an das jeweilige Endgerät angepasst werden. Dies führt zu einer Reihe von Problemen und Einschränkungen und stellt Webdesigner und das Responsive Webdesign Konzept vor große Herausforderungen.

Inhalt und Datenmenge

Separate mobile Websites und Adaptive Webdesign erlauben eine serverseitige gerätespezifische Optimierung des Website-Inhalts und eine damit verbundene Reduzierung der Datenmenge, die an ein Gerät gesendet wird. Dies kann durch eine Limitierung der Inhalte und einer Komprimierung von Medienelementen, wie Bildern, erreicht werden.

Responsive Webdesign nutzt keine serverseitigen Optimierungsmaßnahmen und sendet dementsprechend dieselbe Menge an Daten an alle anfragenden Geräte. Dies führt zu einer unnötigen Dateigröße und einer damit verbundenen langen Ladezeit [9, S. 7–9]. Folglich werden dieselben Websiteelemente, die an einen Desktop Monitor gesendet werden, genauso an ein mobiles Endgerät, mit einem niedriger auflösenden Bildschirm, gesendet, auch wenn dieses Gerät den Inhalt nicht in seiner nativen Auflösung darstellen kann [1, S. 96].

Darüber hinaus kann es, durch die Umstrukturierung des Layouts, zu sehr langen Seiten kommen, bei denen ein Nutzer schnell den Überblick verliert. Die meisten Desktop Websites beinhalten sehr viele Informationen. Sollen diese auf einem mobilen Gerät entsprechend untergebracht werden, wirkt die mobile Version meist überladen und unübersichtlich. Durch die automatische Umstrukturierung des Layouts sind für Nutzer von mobilen Geräten Inhalte oft nicht in der optimalen Reihenfolge angeordnet. Des Weiteren kann es erforderlich sein, dass für die mobile Version einer Website ein anderer Inhalt benötigt wird. Dieses Problem lässt sich mit Responsive Webdesign nicht lösen, da immer derselbe Inhalt an alle Geräte gesendet wird [50].

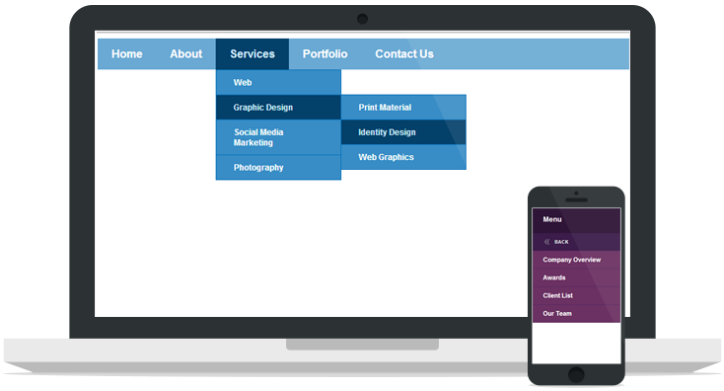


Abbildung 2.10: Die Navigation muss für eine Bedienung per Maus und Tastatur auf einem Desktop Montior, sowie für eine Bedienung per Finger auf einem Smartphone optimiert und implementiert werden. Bildquelle: [25].

Medienelemente

Ein weiteres großes Problem von Responsive Webdesign ist die Verwendung von Medienelementen wie Bildern. Diese Medieninhalte passen ihre Größe meist nur an das umschließenden Element an und können nicht, wie bei separaten mobilen Websites oder Adaptive Webdesign, ausgetauscht werden. Wird eine responsive Website auf einem mobilen Gerät geöffnet, werden sie nur auf die verfügbare Breite skaliert und nicht komprimiert. Dementsprechend ändert sich auch die Dateigröße dieser Elemente nicht und dies führt zu einer schlechten Performance und längeren Ladezeiten auf mobilen Geräten. In der heutigen Zeit weisen viele mobile Websites bereits dieselbe Dateigröße wie ihre entsprechenden Desktop Versionen auf [1, S. 96–97]. Dies sollte durch *Mobile First* und *Progressive Enhancement* vermieden werden, denn gerade die Performance und Bandbreite spielen auf mobilen Geräten, mit einem begrenzten Datenvolumen, eine entscheidende Rolle.

Gerätespezifische Inhalte

Verschiedene Geräteklassen benötigten unterschiedliche gerätespezifische Inhalte. Ein Beispiel wäre die Navigation die für Desktop Nutzer eine Bedienung per Maus und Tastatur und auf einem mobilen Gerät eine Nutzung per Finger ermöglichen muss. Ein Austausch und eine Optimierung von Websitekomponenten ist mit Responsive Webdesign nicht umsetzbar. Die einzige Möglichkeit Inhalte zu wechseln oder zu verbergen, ist es, diese wechselweise über die CSS Eigenschaft *display:none* auszublenden. So können zwei Navigation-Pattern implementiert und je nach Geräteklasse ausgetauscht, beziehungsweise konvertiert werden (siehe Abb. 2.10). Allerdings werden die ausgeblendeten Websiteelemente nur versteckt und im Hintergrund trotz-

dem geladen, was wiederum zu einer schlechteren Performance führen kann. Dieser Vorgang kann nur durch serverseitige Optimierungsmaßnahmen effizienter gestaltet werden. Dementsprechend würde eine serverseitige Komponente, wie bei Adaptive Webdesign, die Auslieferung von, für die jeweilige Geräteklasse optimierten, Websiteelementen ermöglichen [9, S. 6–7].

User Experience

Ein weiterer Kritikpunkt an Responsive Webdesign ist die Gefahr, dass bei der Entwicklung der mobile Kontext nicht ausreichend berücksichtigt wird. Wird derselbe Code für die Desktop und die mobile Webpräsenz verwendet, kann dies zu einer Vernachlässigung der User Experience[6] für mobile Nutzer führen. Durch die Umstrukturierung des Inhalts entstehen lange unübersichtliche Seiten und die Verwendung von vielen Bildern kann sich negativ auf die Performance auswirken. Darüber hinaus weisen responsive Websites oft eine zu hohe Dateigröße und Datenmenge auf. Dies ist vor allem für mobile Nutzer mit einer langsameren Internetverbindung und einem begrenzten Datenvolumen ein wesentlicher Nachteil [70]. Da es sich bei Responsive Webdesign um eine Kompromisslösung aus Desktop und mobiler Website handelt, werden teilweise auch spezielle Möglichkeiten des mobilen Kontextes, über die zusätzliche Sensorik, nicht voll ausgeschöpft [3].

2.3.4 Zusammenfassung

Responsive Webdesign ist ein aktuelles Verfahren, dass die Erstellung einer einzigen anpassungsfähigen Website, die an alle verfügbaren Geräte ausgeliefert wird, ermöglicht. Das bedeutet, dass sich das Layout an die verfügbare Bildschirmauflösung anpasst und der Inhalt entsprechend umstrukturiert wird. Die Website reagiert somit automatisch auf die Gegebenheiten des Endgeräts. So kann ein Layout auf einem größeren Bildschirm mehrspaltig und auf einem mobilen Gerät einspaltig angezeigt werden. Die wichtigsten Bestandteile einer responsiven Website sind *Media Queries*, flexible Bilder und ein flexibles Grid. Bilder und andere eingebettete Medienelemente werden proportional zum definierten Layout skaliert. Durch *Mobile First* und *Progressive Enhancement* beschränkt sich der Inhalt einer Website auf das Wesentliche und wird auch auf einem Desktop Monitor nicht unübersichtlich. Eine einheitliche URL Struktur ermöglicht ein konsistentes Nutzungserlebnis über mehrere Geräte hinweg und ist auch für eine gute Suchmaschinen-Bewertung von Vorteil. Die Weiterentwicklung und Wartung einer responsiven Website ist mit einem geringeren Aufwand verbunden, da diese nur einmal, auf einer einzigen Webpräsenz, durchgeführt werden muss. Im Gegensatz zu separaten mobilen Websites und Adaptive Webdesign, ist eine

[6] User Experience umschreibt alle Aspekte der Erfahrungen eines Nutzers bei der Interaktion mit einem Produkt, Dienst, einer Umgebung oder Einrichtung [65].

responsive Website durch ihr flexibles Layout, zukunftsorientiert und bietet eine entsprechend optimierte Darstellung auf allen gängigen und zukünftigen Displaygrößen.

Allerdings bietet Responsive Webdesign nicht nur Vorteile, sondern ist auch mit einer Reihe von Nachteilen verbunden. So ist es nicht möglich, auf der mobilen Website einen anderen Inhalt anzubieten, da diese Version auf der Desktop Präsenz basiert. Dies kann zu langen unübersichtlichen Seiten und verbunden mit skalierten Bildern, zu einer schlechteren Performance führen. Unterschiedliche Geräteklassen benötigen verschiedene gerätespezifische Websitekomponenten, die für das jeweilige Bedienungskonzept optimiert und ausgetauscht werden sollten. Eine solche Optimierung ist mit Responsive Webdesign, ohne die Integration einer serverseitigen Komponente, nicht möglich.

Wir eine Website von Grund auf neu erstellt, ist Responsive Webdesign mit *Mobile First* und *Progressive Enhancement* ein sehr guter Ansatz, der es ermöglicht, sich bei der Entwicklung der Webpräsenz auf die wichtigsten und elementarsten Elemente einer Website zu fokussieren und die mobile Version von Anfang an im Blick zu haben [50].

Kapitel 3

Stand der Technik

Responsive Webdesign hat sich in den letzten Jahren schnell weiterentwickelt und ist zu einem der bekanntesten Lösungsansätze im mobilen Web geworden. Betrachtet man die stetig steigende Anzahl an mobilen Geräten die heute das Web nutzen, steigt auch der Bedarf für flexible, anpassungsfähige Websites und entsprechende Umsetzungsverfahren. Jedoch bleiben die größten Probleme des Responsive Webdesign Konzeptes bestehen. Bilder können meist nicht geräte- und auflösungsabhängig optimiert werden und sorgen somit für eine schlechte Performance auf mobilen Geräten. Auch die Optimierung von anderen Inhalten und der Austausch von Websiteelementen für Geräteklassen ist mit clientseitigem Responsive Webdesign nicht möglich. Seit Marcotte den Begriff Responsive Webdesign geprägt hat, haben sich allerdings eine Reihe von neuen Technologien und Techniken etabliert, die eine Verbesserung dieser Einschränkungen anstreben [12, S. 4].

3.1 Responsive Web Frameworks

Die Erstellung einer responsiven Website erfordert einen erheblichen Mehraufwand, da diese auf allen Geräten und Displaygrößen ein entsprechend gutes Ergebnis liefern soll. Dementsprechend müssen responsive Websites theoretisch an alle möglichen *Breakpoints* angepasst werden. Dies erfordert einen erheblichen Konzeptions-, Umsetzungs- und Testaufwand. Basierend auf dieser Tatsache haben sich eine Reihe von Frameworks entwickelt, die die Umsetzung einer responsiven Website enorm erleichtern.

Zwei der wohl bekanntesten Responsive Web Frameworks sind *Twitter Bootstrap* und *Foundation*. *Bootstrap* [26] wurde von *Twitter* entwickelt und ist mittlerweile in der Version 3 verfügbar. Es basiert auf einem flexiblen 12-spaltigen CSS Grid mit einer Breite von 940 Pixel und verwendet LESS-CSS, eine effektive Stylesheet-Sprache. Es ist ein sehr umfangreiches Framework und enthält bereits eine Reihe von CSS Styles, Icons und Design-Pattern. Darüber hinaus lässt sich *Bootstrap* durch eine Reihe von jQuery Plugins er-

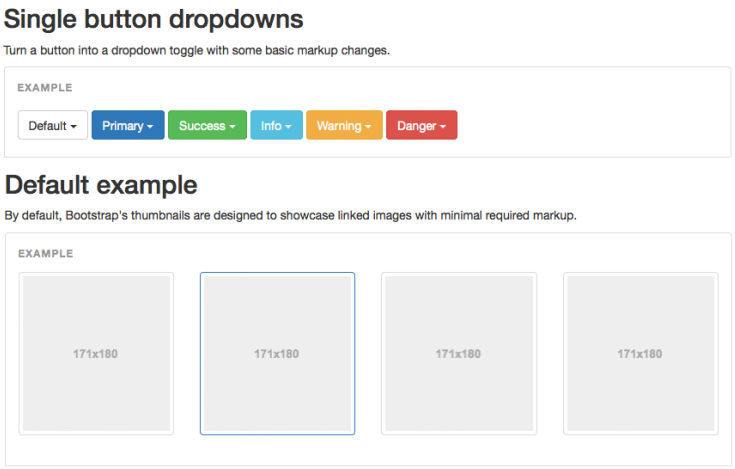


Abbildung 3.1: Responsive Web Frameworks stellen bereits eine Reihe von vordefinierten Komponenten zur Verfügung. Bildquelle: [26].

weitern. *Foundation* [27] zählt neben *Twitter Bootstrap* zu den am weitesten entwickelten und verbreiteten Frameworks. Es ist ebenfalls ein Open-Source-Projekt und wird von der digitalen Agentur ZURB entwickelt und gepflegt. Anders als *Bootstrap* nutzt *Foundation* SASS-CSS und Zepto-JavaScript, ein äquivalent zur jQuery.

Die Nutzung eines Frameworks bietet eine Reihe von Vorteilen, ist allerdings auch mit Nachteilen verbunden. Wird bei der Umsetzung einer responsiven Website auf ein Framework gesetzt, stehen von Beginn an eine Reihe von Templates und CSS Styles zur Verfügung, die die Umsetzungszeit signifikant verkürzen. Darüber hinaus wird der Code des Frameworks von den Entwicklern permanent gewartet, Fehler beseitigt und der Funktionsumfang durch Updates erweitert. Da man bei der Verwendung eines Frameworks an die entsprechenden Komponenten und CSS Regeln gebunden ist, schränkt dies aber auch die Anpassungsfähigkeit und Flexibilität einer Website ein. Dementsprechend müssen meist eine Reihe von CSS Styles überschrieben werden, was zu doppeltem CSS Code und einem unnötigen Overhead führt [52]. Wird der Framework CSS Code nicht überschrieben oder ausgetauscht, sehen sich Websites, die auf dem gleichen Framework basieren, oftmals sehr ähnlich. Die Nutzung eines Frameworks sollte somit für jedes Projekt individuell entschieden werden.

3.2 Optimierung von Medien

Bilder und andere Medienelemente stellen seit jeher eines der größten Probleme des Responsive Webdesign Konzeptes dar. Der Fokus dieses Abschnittes

liegt auf der Optimierung von Bildern, da diese die wohl größte Herausforderung für Responsive Webdesign repräsentieren. Der Vollständigkeit halber wird auch auf die Optimierung von eingebetteten Medieninhalten, wie Videos, eingegangen.

Durch die Limitierung des *<img>* HTML-Elements ist es nicht möglich responsive flexible Bilder zu definieren. Dementsprechend werden Bilder nur skaliert und nicht komprimiert, was zu einer schlechteren Performance und langen Ladezeiten auf mobilen Geräten führt [5, S. 2]. Um diese Probleme mit reinem HTML zu beheben, hat sich die W3C Responsive Images Community Group [28] gegründet. Im Rahmen der Diskussion wurde ein weiteres HTML-Element, *<picture>*, sowie die neuen Attribute namens *srcset* und *sizes* definiert [57].

3.2.1 <picture>, srcset und sizes

Mit dem *srcset* Attribut ist es möglich ein Set von Bildern zu definieren. Dabei wird das gewöhnliche *<img>* HTML-Element verwendet und um zusätzliche Bildquellen erweitert. Das folgende Beispiel demonstriert die Verwendung des neuen Attributes:

```
<img alt="dog" src="dog.jpg" srcset="dog-HD.jpg 2x, dog-narrow.jpg 300w,
     dog-narrow-HD.jpg 600w 2x" />
```

Das *src* Attribut dient hierbei als Fallback-Lösung. Das bedeutet, dass das spezifizierte Bild geladen wird, wenn ein Browser die neue *srcset* Erweiterung nicht unterstützt. Das *srcset* Attribut erlaubt die Verwendung einer Microsyntax [20, S. 4–6]. Das erste Argument stellt den Pfad zu der Bilddatei dar. Standardmäßig wird auf einem großen normalen Display das Bild *dog.jpg* geladen. Das zweite Argument, der w-Deskriptor, beschreibt eine physikalische Breite des Bildes in Pixel. Dies führt dazu, dass der Browser bei einem *Viewport* mit einer Breite von unter 300 Pixel ($300\,w$) und einem normalen Display die Grafik *dog-narrow.jpg* lädt. Das letzte Argument, der x-Deskriptor, bestimmt bei welchem DPR (Device Pixel Ratio) ein Bild geladen wird. $1\,x$ entspricht einem normalen Display und $2\,x$ einem hochauflösenden Bildschirm mit doppelter Pixeldichte (z.B. Retina-Display von Apple). Somit wird die Grafik *dog-HD.jpg* auf einem größeren, hochauflösenden Display und das Bild *dog-narrow-HD.jpg*, ebenfalls auf einem hochauflösenden Display, bei einem entsprechend kleineren *Viewport* ($300\,w \cdot 2 = 600\,w$) ausgegeben [57].

Durch diese Vorgehensweise werden Bilder je nach *Viewport*-Breite ausgetauscht. Soll allerdings ein flexibles mehrspaltiges Layout verwendet werden funktioniert dieser Ansatz nicht mehr, da der Browser die Breite der Bilder nicht kennt. Durch die dynamische Umstrukturierung des Layouts kann es vorkommen, dass Bilder in größeren *Viewports* kleiner dargestellt werden [57]. Dieses Problem basiert auf der Tatsache, dass Browser Bilddateien aus Performance-Gründen immer als erstes laden. Das bedeutet, dass

ein Browser beim Aufruf einer Website das HTML nach Bild-URL's durchsucht und diese im Anschluss lädt, bevor er mit dem Aufbau des DOM-Baumes und dem Laden externer CSS und JavaScript Ressourcen fortfährt. Allerdings kennt der Browser immer die Auflösung und Breite des aktuellen *Viewports*. Diese Informationen nutzen auch *Media Queries* (siehe Abschnitt 2.2) um ein Layout auf einem kleineren Display entsprechend umzustrukturieren. Um diese Angaben auch für Bilddateien zu ermöglichen, wurde das *sizes* Attribut eingeführt. Über dieses Attribut kann festgelegt werden, wie viel Platz (Pixel) von einer Grafik benötigt wird und wie breit das Bild dargestellt werden soll. Durch diese zusätzliche Angabe lädt der Browser immer die kleinstmögliche Bilddatei, die ein entsprechend scharfes Ergebnis in ihrem Container liefert. Für das folgende Beispiel sollen drei verschiedene Bildversionen, je nach Displaygröße, in einem flexiblen Grid ausgegeben werden [14, S. 15–18]:

```
<img srcset="large.jpg 1024w, medium.jpg 640w, small.jpg 320w" sizes="(
    min-width: 36em) 33.3vw, 100vw" src="small.jpg" alt="A rad wolf" />
```

Dabei werden über das *srcset* Attribut die drei Bilddateien mit entsprechender Breitenangabe (*w*-Deskriptor) definiert. Beispielsweise hat die Bilddatei *large.jpg* eine physikalische Auflösung von 1024×768 Pixel. Zusätzlich ist in diesem Beispiel noch das Attribut *sizes* angeführt. Innerhalb dieses Attributes können nun verschiedene Rahmenbedingungen beschrieben werden. Die Angaben bestehen dabei aus einer optionalen *Media Query*, die die *Viewport*-Breite angibt und der Breite der Bilddatei. Diese wird meist in der Einheit *vw* (*Viewport*-Width) angegeben, kann allerdings auch in absoluten Werten wie Pixel sowie *em* definiert oder über die *calc()*-Funktion berechnet werden [57]. Wird die Breite des Bildes in *vw*, also relativ zum *Viewport*, angegeben ermöglicht dies eine flexible Skalierung der Grafik. In diesem Beispiel soll ein Bild in ein dreispaltiges Layout eingegliedert werden (siehe Abb. 3.2). Basierend auf dieser Tatsache muss die Bilddatei ein Drittel des *Viewports* (33.3 *vw*) einnehmen. Allerdings basiert unser Layout in diesem Beispiel auch auf einem *Breakpoint* bei 36 *em*. Ist ein Display oder Fenster kleiner als dieser Umbruchpunkt, wird das Layout entsprechend umstrukturiert. Dabei nehmen die Bilder dann die volle Breite, also 100 %, des *Viewports* ein. Diese neue Information muss ebenfalls im *sizes* Attribut angegeben werden. Dabei wird nun die *Media Query* und die zusätzliche *Viewport*-Width (100 *vw*) definiert:

```
sizes="(min-width: 36em) 33.3vw, 100vw"
```

Durch diese Angaben nimmt ein Bild ein Drittel des *Viewports* ein, solange dieser eine Mindestbreite von 36 *em* aufweist. Ist der *Viewport* kleiner, wird die Grafik über die gesamte Breite (100 %) dargestellt (siehe Abb. 3.2). Der Browser liest dabei das *sizes* Attribut der Bilddatei aus und prüft jede angegebene *Media Query* bis eine zutreffende Abfrage gefunden wird. Sollte

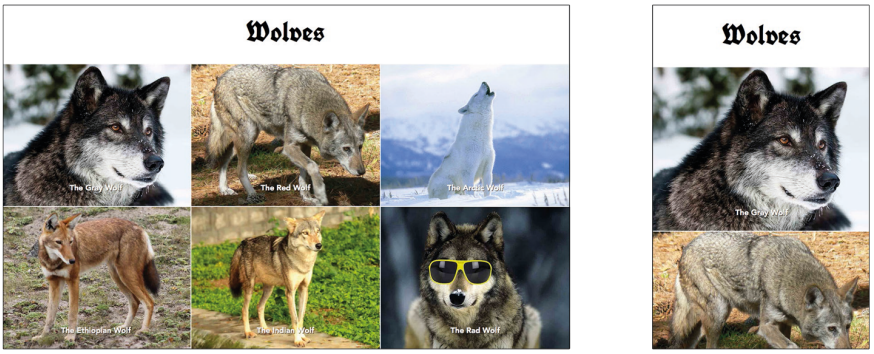


Abbildung 3.2: Beispiel für eine Website mit einem dreispaltigen Layout auf einem großen Monitor und umstrukturiert auf einem kleinen Display. Bildquelle: [14, S. 18].

kein *Query* zutreffen, wird die *default length* (100 *vw*), also die angegebene Standardbreite der Grafik, verwendet. Die Angaben im *sizes* Attribut sind dabei immer folgendermaßen aufgebaut:

```
sizes="[media query] [length],
[media query] [length],
...
[default length]"
```

Diese zusätzlichen Angaben ermöglichen einem Browser Bilder effizient zu laden und in einem responsiven fluid Layout optimal darzustellen [14, S. 17–19]. Allerdings kann ein Entwickler derzeit noch nicht bestimmen, welche Grafik von einem Browser verwendet werden soll. Diese Entscheidung liegt bei dem jeweiligen Browser selbst. Somit eignen sich *srcset* und *sizes* aktuell nur für unterschiedliche Größen und Qualitätsstufen desselben Motivs [57].

Eine Veränderung des Bildausschnittes oder der Austausch des kompletten Bildes ist durch das neue *<picture>* HTML-Element möglich und wird als *Art Direction* bezeichnet. Das *<picture>* Element dient als Container für verschiedene Bildquellen. Die einzelnen Bilder werden dabei über ein *<source>* HTML-Element definiert. Bei dem folgenden Beispiel wird eine Grafik auf einem schmaleren *Viewport* im Hochformat und auf einem breiteren *Viewport* im Querformat angezeigt:

```
<picture>
  <source media="(min-width: 38em)" srcset="art-direction-horiz.jpg" />
  <source srcset="art-direction-vertical.jpg" />
  <img src="art-direction-vertical.jpg" alt="" />
</picture>
```

Mit diesem Code wird die Grafik *art-direction-vertical.jpg* durch die Grafik *art-direction-horiz.jpg* ersetzt, wenn der *Viewport* eine Breite von 38 *em*

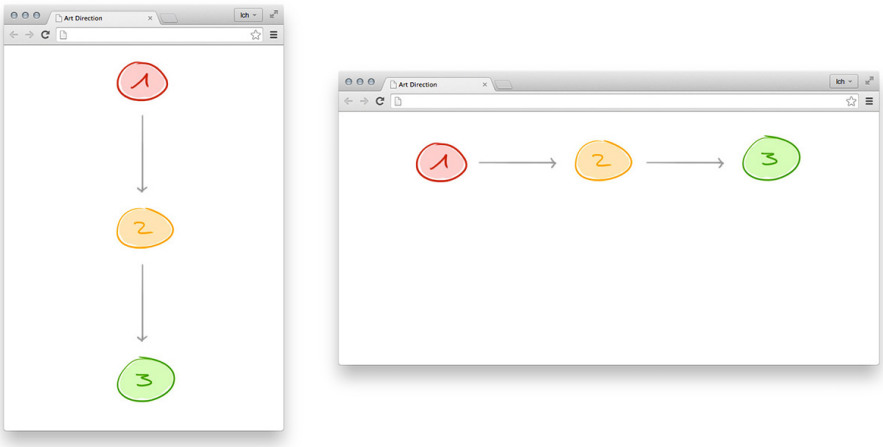


Abbildung 3.3: Wird das *<picture>* HTML-Element von einem Browser unterstützt, kann eine Bilddatei, je nach *Viewport*, ausgetauscht werden. Bildquelle: [57].

überschreitet. Diese Vorgehensweise ermöglicht es, dass bestimmte horizontal ausgerichtete Grafiken auch auf einem kleineren Display erkennbar bleiben (siehe Abb. 3.3) [57]. Dabei werden innerhalb des *<picture>* Elements die verschiedenen Bildquellen vom Browser von oben nach unten gelesen und wenn eine *Query* zutrifft, die entsprechende Grafik geladen. Der letzte Bestandteil ist das bekannte *<img>* Element. Dieses dient als Fallback-Lösung. Sollte keine *Media Query* zutreffen, wird diese Bilddatei geladen [14, S. 21–22]. Zusätzlich ist auch die Einbindung von verschiedenen Bildquellen mit unterschiedlichen Qualitätsstufen (Device Pixel Ratios) möglich:

```
<picture>
  <source media="(min-width: 56.25em)" srcset="large.jpg 1x, large@2x.
    jpg 2x" />
  <source media="(min-width: 37.5em)" srcset="medium.jpg 1x, medium@2x.
    jpg 2x" />
  <source srcset="small.jpg 1x, small@2x.jpg 2x" />
  <img src="fallback.jpg" alt="" />
</picture>
```

Bei diesem Beispiel basiert das Design auf zwei Breakpoints (37.5 *em* und 56.25 *em*). Auf einem kleinen normalen Display, mit einem *Viewport* unter 37.5 *em*, wird die Bilddatei *small.jpg* geladen. Handelt es sich dabei um ein hochauflösendes Display, wird die Bilddatei *small@2x.jpg* verwendet. Werden diese HTML-Elemente von einem Browser nicht unterstützt, wird die über das *<img>* Element spezifizierte Grafik als Fallback-Lösung geladen [57].

Das *<picture>* Element ermöglicht auch die Definition von verschiedenen Dateiformaten. Dies kann über das zusätzliche *type* Attribut realisiert werden. Über dieses Attribut wird der MIME-Type der entsprechenden Bild-

datei definiert:

```
<picture>
  <source type="image/svg" src="logo.svg" />
  <source type="image/png" src="logo.png" />
  <img src="logo.gif" alt="RadWolf, Inc." />
</picture>
```

Kann der Browser den MIME-Type *image/svg* nicht unterstützen, wird dieser Eintrag übersprungen und die Möglichkeit einer Einbindung des nächsten Elements geprüft. Sollte von einem Browser keiner der *<source>* Einträge unterstützt werden, wird wiederum die Bilddatei des *<img>* Elements geladen [14, S. 23–24].

Flexible Bilder sind einer der wichtigsten Bestandteile von responsiven Websites. Allerdings war es bisher ohne Scripte nicht möglich Bilder, auf allen Displaygrößen optimiert, darzustellen. Mit der Entwicklung des *<picture>* Elements und der neuen Attribute *srcset* und *sizes* mitsamt ihren Komponenten wurden neue Möglichkeiten geschaffen und in die Spezifikation aufgenommen. Diese neuen Elemente sollen in Zukunft die Einbindung von flexiblen, optimierten und anpassungsfähigen Bildern mit reinem HTML ermöglichen. Aktuell werden diese HTML-Elemente allerdings noch nicht von jedem Browser unterstützt. Die Verfügbarkeit kann über die Website *CanIUse.com* (*<picture>* [29] und *srcset* [30]) überprüft werden [57]. Aus diesem Grund muss bei der Entwicklung einer responsiven Website, die bei der Einbindung von Bildern auf diese reine HTML Lösung setzt, gegenwärtig eine weitere zusätzliche Fallback-Lösung, zur Realisierung von responsiven Bildern, implementiert werden.

3.2.2 JavaScript Plugins

Sollen das neue *<picture>* HTML-Element und die Attribute *srcset* und *sizes* bereits heute bei der Entwicklung einer Website verwendet werden, kann dies durch ein JavaScript Polyfill Plugin, wie beispielsweise das der Responsive Images Community Group, realisiert werden. Das JavaScript Plugin *Picturefill.js*[1] ermöglicht aktuell eine browserübergreifende Unterstützung der neuen Komponenten [57]. Dabei wird die aktuelle Version 2.0 des Scripts im Header der HTML Seite eingebunden:

```
<!-- Asynchronously load the polyfill. -->
<script src="picturefill.js" async></script>
```

Durch das Attribut *async* blockiert das Script eine Website nicht von einem Browser geladen zu werden. Wird nun ein Bild unter Verwendung des *<img>* Elements und des *srcset* Attributes eingebunden, kann dies, wie in der Spezifikation definiert, durchgeführt werden:

```
<img sizes="100vw, (min-width: 40em) 80vw" srcset="pic-small.png 400w,
    pic-medium.png 800w, pic-large.png 1200w" alt="Obama" />
```

[1] RICG Picturefill Website: http://scottjehl.github.io/picturefill

Allerdings sollte man dabei auf das *src* Attribut verzichten, um einen zusätzlichen Download des Fallback-Bildes auf älteren Browsern vorzubeugen.

Soll auch das *<picture>* Element verwendet und von älteren Browsern unterstützt werden, muss dieses zuerst mit folgendem Script erstellt werden:

```
<script>
  document.createElement( "picture" );
</script>
```

Die Einbindung erfolgt im Anschluss ebenfalls nahe an der Spezifikation. Der Unterschied besteht in der Nutzung des *<img>* Elements. Dabei verzichtet dieses auf ein *src* Attribut als Fallback und basiert ebenfalls auf dem *srcset* Attribut. Diese Vorgehensweise soll wiederum einen doppelten Download der Bilddatei vorbeugen [20, S. 7–10]:

```
<picture>
  <source srcset="extralarge.jpg, extralarge.jpg 2x"
  media="(min-width: 1000px)" />
  <source srcset="large.jpg, large.jpg 2x" media="(min-width: 800px)" />
  <source srcset="medium.jpg" />
  <img srcset="medium.jpg" alt="Cambodia Map" />
</picture>
```

Allerdings besteht auch bei dieser Lösung ein Problem, das alle JavaScript Polyfill Plugins betrifft. Sollte ein Browser das *<picture>* HTML-Element oder das *srcset* Attribut nicht unterstützen und der Website-Besucher zusätzlich JavaScript deaktiviert haben, kann das Polyfill Script nicht ausgeführt werden. Dementsprechend werden auf der Website keine Bilder geladen und nur die jeweiligen *alt* Texte angezeigt [20, S. 11].

Aufgrund der unterschiedlichen Ansätze und Herangehensweisen wird zusätzlich auf weitere populäre JavaScript Plugins eingegangen. Ein anderes Plugin setzt beispielsweise auf das klassische *<img>* Element und die zusätzliche Verwendung von *data* Attributen. *HiSRC* ist ein Script, das die Erstellung von Bildern in verschiedenen Auflösungen ermöglicht [53]. Basierend auf Daten, wie der Displayauflösung oder der verfügbaren Bandbreite, werden die Bilder entsprechend ausgetauscht. Dabei wird im *src* Attribut des *<img>* Elements der Pfad zu der kleinsten erstellten Version der Grafik angegeben. Die höher aufgelösten Versionen des Bildes werden im Anschluss in den *data* Attributen definiert:

```
<img src="http://placekitten.com/200/100" data-1x="http://placekitten.
    com/400/200" data-2x="http://placekitten.com/800/400" class="hisrc"
    alt="placekitten" />
```

Durch die CSS Klasse *hisrc* erkennt das Script das Bildelement und kann den Pfad zu der Grafik entsprechend austauschen. Allerdings wird bei dieser Lösung das im *src* Attribut spezifizierte kleinste Bild immer geladen, auch wenn der Besucher die Website von einem Desktop Monitor aufruft. Somit wird auf einem solchen Gerät mit einer hohen verfügbaren Bandbreite jede Bilddatei doppelt geladen. Ein mobiles Gerät, mit einer schlechteren

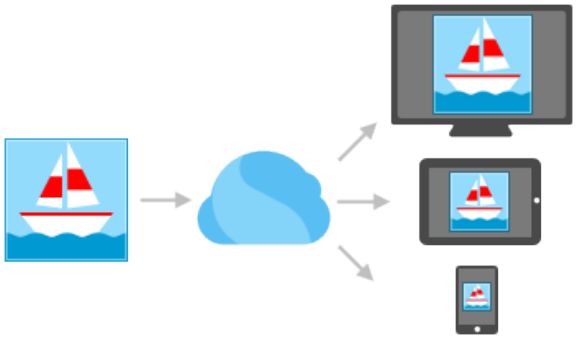


Abbildung 3.4: *Sencha.io Src* ermittelt die Auflösung eines Geräts und skaliert ein Bild in der Cloud automatisch auf die passende Größe. Bildquelle: [69].

Netzwerkverbindung, lädt andererseits nur die optimierte kleine Version. Ein weiterer Nachteil dieser Lösung ist, dass das Plugin auf jQuery basiert und somit die umfangreiche JavaScript Bibliothek voraussetzt [43].

Ein wesentliches Problem vieler JavaScript Lösungen besteht darin, dass ein Webbrowser alle Bilddateien einer HTML Seite vor dem Ausführen dieser lädt und eine im *src* Attribut des *<img>* Elements definierte Bilddatei somit immer geladen wird. Ein Eingreifen und Austauschen des Bildpfades, bevor die Grafik geladen wird, ist somit nicht möglich. Das Script *Mobify.js* arbeitet an einer Lösung den HTML Code einer Seite zu scannen, bevor diese geladen wird [31]. Dies würde den Austausch von Bildpfaden vor dem ersten Laden der Website durch den Browser ermöglichen. Allerdings handelt es sich hierbei um keine verlässliche Lösung, da diese Technik derzeit nur von aktuellen Webbrowsern unterstützt wird [43].

3.2.3 Responsive Webserver

Eine weitere Möglichkeit bieten responsive Webserver wie *ReSRC.it* [32] oder *Sencha.io Src* [43]. Diese fungieren als Proxy der zwischen der Website und der Bilddatei arbeitet [69]. *Sencha.io Src* ermittelt anhand des *User-Agent-Strings* des anfragenden Geräts die Bildschirmauflösung und ändert dementsprechend die Größe der Bilddatei (siehe Abb. 3.4). Der Service erwartet hierzu den Pfad zu einer Bilddatei, der anschließend in einer optimierten Version zurückgegeben wird [8, S. 113]. Dabei wird dem Pfad des *src* Attributes im *<img>* Element einfach die Adresse des Anbieters, wie beispielsweise *Sencha.io Src*, vorangestellt [69]:

```
<img src="http://src.sencha.io/http://www.your-domain.com/image.jpg" />
```

Die Möglichkeiten von Webservices wie *Sencha.io Src* sind allerdings auch begrenzt. Beispielsweise können Bilder nur verkleinert werden. Das bedeutet, dass immer ein Bild in einer entsprechend hohen Qualität vorhanden

sein muss. Da *Sencha.io Src* Bilder nur skaliert, kann auch der Bildausschnitt einer Grafik nicht verändert werden. Zudem handelt es sich hierbei um eine externe Lösung und die Funktionalität einer Website ist somit an die Webserver und Richtlinien eines Dritten gebunden [8, S. 114].

3.2.4 Adaptive Images

Adaptive Images ist eine serverseitige Lösung, basierend auf einer *User-Agent-Detection*, die die Bildschirmgröße eines Geräts ermittelt und im Anschluss eingebundene Bilder automatisch skaliert und im Cache ablegt [73]. Dabei wird die Auflösung eines Geräts ermittelt und in einem Cookie gespeichert. Basierend auf dieser Information wird im Anschluss nach einem Bild mit einer entsprechenden Auflösungen gesucht. Wird keine Bilddatei gefunden, wird die große Grafik entsprechend skaliert und in einem Cache Ordner abgelegt [8, S. 114–116]. Dieses Plugin eignet sich hervorragend für bestehende Websites, da der HTML Code einer Website nicht verändert werden muss. Das PHP Script interpretiert Anfragen eines Browsers für Bilddateien und passt die Grafiken im Anschluss automatisch an die Gegebenheiten des Geräts an [43]. Ein Problem dieser Lösung besteht darin, dass die URL unabhängig von der Größe des angeforderten Bildes immer gleich bleibt. Dies kann zu Schwierigkeiten mit *Content Delivery Networks* (CDNs) führen, da hier die URL zu einer Grafik im Cache abgelegt wird, um die Zugriffszeit beim nächsten Zugriff zu verbessern. Basierend auf dieser Tatsache, kann es vorkommen, dass bei einem wiederholten Zugriff ein Bild mit der falschen Größe vom Cache geladen wird [8, S. 116].

3.2.5 CSS Background Images

In manchen Fällen bietet sich auch die Einbindung und der Austausch von Bildern durch CSS an. Dabei wird eine Bilddatei in verschiedenen Größen und Auflösungen erstellt und anschließend mit der CSS Eigenschaft *background-image* eingebunden:

```
@media only screen and (max-width: 320px) {
  /* Small screen */
  .hero #cafe { background-image: url("../img/candc290.jpg"); }
}
```

Durch die Nutzung von *Media Queries* kann ein Bild, je nach Auflösung entsprechend, ausgetauscht werden. Ein wechselweises ein- und ausblenden von verschiedenen Bilddateien, über die CSS Eigenschaft *display:none*, sollte vermieden werden, da ausgeblendete Grafiken im Hintergrund dennoch vom Browser geladen werden. Wird eine Bilddatei über die CSS Eigenschaft *background-image* eingebunden, wird kein JavaScript benötigt, was wiederum zu einer besseren Performance führt. Allerdings widerspricht dieser Lösungsansatz dem grundlegenden Konzept von CSS, das nur für das Design

und nicht für den Inhalt einer Website verwendet werden sollte [17, S. 81–95].

3.2.6 SVG (Scalable Vector Graphics)

Das Dateiformat SVG ermöglicht die Darstellung von Vektor-Grafiken im Browser. Dementsprechend werden Farbwerte und Formen nicht über Pixel definiert, sonder mathematisch beschrieben. Dadurch haben SVG Grafiken eine sehr geringe Dateigröße und können verlustfrei und ohne Zuwachs dieser beliebig skaliert werden. Wird eine Grafik als responsive SVG erstellt, werden die Höhen- und Breitenangaben aus dem SVG-Code oder der Bilddatei entfernt. Dies ermöglicht es, dass sich eine Grafik später automatisch an den zur Verfügung stehenden Platz anpasst. Ein erstelltes Bild kann im Anschluss als Code in einer HTML-Datei oder als Grafik eingebunden werden. Allerdings werden SVG Grafiken nicht von jedem Browser unterstützt und es sollte somit immer eine Fallback-Lösung in einem anderen Dateiformat, wie PNG, eingebunden werden [58]. SVG Vektor-Grafiken eignen sich aufgrund ihrer verlustfreien Skalierbarkeit im Web besonders für Bitmap Grafiken wie Logos, Infografiken, Werbungen und Icons. Die Verwendung eines Image-Sprites ermöglicht die Implementierung einer responsiven SVG Grafik. Image-Sprites sind Bilddateien die mehrere Grafiken beinhalten. Dabei wird durch CSS die Position einer Grafik im Sprite bestimmt und der maskierte Bereich entsprechend ausgegeben. Somit kann ein Bild auf einer Website, je nach Auflösung, automatisch ausgetauscht werden. Für die Abfrage der einzelnen *Breakpoints* und das Austauschen der Bilder werden *Media Queries* verwendet.

Das SVG Format ermöglicht eine direkte Einbettung von CSS in den SVG Code. Dabei werden alle Grafiken in einer SVG Datei übereinander platziert und mit nummerierten CSS Klassen versehen:

```
@media screen and (min-width: 25em) {
  #home_icon_0 {
    display: none;
  }
  #home_icon_1 {
    display: block;
  }
}
```

Anschließend wird immer die gewünschte Grafik mit *display: block* innerhalb des jeweiligen *Breakpoints* angezeigt und die anderen Bilder mit *display: none* ausgeblendet. Somit muss nur die entsprechende SVG Grafik eingebunden werden und es wird keine separate CSS Datei für das Maskieren der einzelnen Grafiken benötigt [15, S. 116–124].

SVG Dateien unterstützen ebenfalls eine direkte Einbindung von Rastergrafiken. Durch *Media Queries* und die CSS Eigenschaft *background-image* werden die einzelnen Bilddateien direkt im SVG Code definiert und anschließend entsprechend ausgegeben:

```
@media screen and (max-width: 400px) {
  svg {
    background-image: url("images/small.png");
  }
}
```

Somit muss nur eine SVG Grafik im HTML Code eingebunden werden. Die Rastergrafiken werden im Anschluss, je nach Displayauflösung und *Media Query*, geladen und automatisch ausgetauscht. Der Nachteil dieser Lösung besteht darin, dass Bilder durch die CSS Eigenschaft *background-image* eingebunden und somit von einem Browser nicht direkt als Bildelemente erkannt werden [19, S. 67–77].

3.2.7 WebP

WebP ist, wie in [33] beschrieben, die Bezeichnung einer neuen Dateikompression für Grafiken. Das Dateiformat wurde von Google entwickelt und erzeugt bei JPEG-ähnlicher Qualität knapp 25–34 % kleinere Dateien. Zusätzlich unterstützt es eine verlustfreie Transparenz und ist somit, bei einer 26 % kleineren Dateigröße, auch als Ersatz für PNG Dateien geeignet. Durch die höhere Kompression bei gleichbleibender Qualität, eignet sich dieses Bildformat besonders für das mobile Web und Responsive Webdesign. Allerdings wird WebP aktuell nur von den Webbrowsern Google Chrome und Opera unterstützt und eignet sich somit derzeit nicht für eine geräte- und betriebssystemübergreifende Implementierung.

3.2.8 CMS Plugins

Auch CMS (Content Management System) Anbieter müssen auf die neuen Gegebenheiten und Spezifikationen zur Einbindung von anpassungsfähigen Bildern reagieren. Einige Anbieter, wie beispielsweise TYPO3, haben die Funktionalität bereits in das CMS integriert. Des weiteren stehen auch eine Reihe von Plugins und Extensions zur Verfügung. Wie in [72] beschrieben, bietet TYPO3, ab der Version 6.2 LTS, eine integrierte Lösung zur Realisierung von responsiven Bildern an. Dabei kann zwischen dem *<picture>* Element, dem *srcset* Attribut oder dem *data* Attribut gewählt werden. Die gewünschte Variante muss unter der Einbindung der benötigten JavaScript Bibliothek von einem Entwickler ausgewählt werden. Nachdem die entsprechende Konstante in der Konfiguration gesetzt wurde, erstellt TYPO3 für jedes eingebundene Bild eine normale und eine hochauflösende Version. Sollen Bilder in verschiedenen Auflösungen für unterschiedliche *Breakpoints* erstellt werden, müssen diese ebenfalls in der Konfiguration als sogenannte *dataKeys* definiert werden.

Für das CMS WordPress stehen ebenfalls verschiedene Plugins zur Verfügung [11, S. 99–103]. Eine automatisch Lösung und Integration der Spezifikation wurde von der RICG und dem WordPress-Core-Team entwickelt [57].

Dieses Plugin erzeugt verschiedene Auflösungen eines Bildes automatisch und bindet diese mit dem *srcset* Attribut ein. Als Fallback-Lösung nutzt es die JavaScript Polyfill Bibliothek *Picturefill.js* (siehe Abschnitt 3.2.2).

Auch Drupal [60] integriert responsive Bilder in die kommende Version 8. Dies wird hier ebenfalls durch ein entsprechendes *Breakpoint* Mapping und dem HTML5 *<picture>* Element realisiert. Des weiteren gibt es auch für Drupal eine Reihe von Modulen, die beispielsweise durch JavaScript ein entsprechendes Ergebnis liefern [9, S. 69].

Aufgrund des schnellen Erfolges von Responsive Webdesign und den damit verbundenen neuen Möglichkeiten und Herausforderungen, waren auch CMS Anbieter gefordert, ihre System entsprechend anzupassen [11, S. 97–98]. Lange Zeit waren diese nur auf Desktop Präsenzen ausgerichtet und erstellte Websites konnten nicht auf verschiedene Geräteklassen und Auflösungen reagieren. Aktuell gängige CMS Systeme bieten nun bereits integrierte responsive Layouts und entsprechende Konfigurationsmöglichkeiten. Auch die Einbindung der Bilder Spezifikation des W3C und der RICG und die damit verbundene Integration der neuen HTML-Elemente und Attribute wurde weitgehend umgesetzt.

3.2.9 Flexible Videos

Auch eingebettete Medienelemente, wie Videos, stellen eine Herausforderung für Responsive Webdesign dar. Wird ein Video in einem responsiven Layout eingebettet, passt sich das Layout zwar flexibel an die Breite des Displays an, das Video selbst behält allerdings seine feste Größe und reagiert nicht auf die CSS Angaben. Dies führt zu einer abgeschnittenen oder verzerrten Darstellung auf mobilen Geräten. Wie in [55] beschrieben, gibt es auch für die Einbindung von Videos bereits eine Reihe von JavaScript Plugins, die eine korrekte Darstellung des Inhalts ermöglichen. Ein Beispiel ist das Plugin *FitVid.js* [48]. Es umschließt das eingebundene Video automatisch mit entsprechenden CSS Containern und ermöglicht so eine flexible Skalierung über alle Displaygrößen hinweg.

Um eine flexible Skalierung auch ohne JavaScript zu lösen, sollte das CSS Markup entsprechend erweitert werden. Dabei wird das eingebettete Element mit einem, mit einer CSS Klasse (*responsive-video*) versehenen, * * Container umschlossen:

```
<div class="responsive-video">
  <!-- hier steht der Embed-Code des Videos -->
</div>
```

Mit dem folgenden CSS Code passt sich das eingebettete Video immer an das umschließende relative Container-Element an. Die CSS Regel *padding-bottom: 56.25%* bezieht sich auf das 16:9 Bildformat. Wird dabei 9 durch 16 dividiert, ergibt das 56,25 %. Das Container-Element weist somit ein spezifiziertes Formverhältnis auf, an das sich das Video flexibel anpasst. Die

weiteren CSS Eigenschaften *height: 0* und *overflow: hidden* werden für eine problemlose Cross-Browser-Kompatibilität benötigt:

```
.responsive-video {
  position: relative;
  padding-bottom: 56.25%;  /* 16:9 */
  padding-top: 30px;
  height: 0;
  overflow: hidden;
}

.responsive-video iframe,
.responsive-video object,
.responsive-video embed {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
}
```

Das eingebettete Video wird *absolut* positioniert. Durch die CSS Regeln *top: 0* und *left: 0* wird der Inhalt in der linken oberen Ecke des Containers platziert. Die einzelnen Werte können, je nach Design und Layout einer Website, angepasst werden. Eine solche Vorgehensweise ermöglicht eine problemlose Einbettung und geräteübergreifende flexible Skalierung von Videos und anderen Medienelementen [55]. Allerdings sollte auch bei Videos immer auf die Performance geachtet werden. Werden sie auf einem mobilen Gerät nicht benötigt oder wird das Format von einem Endgerät nicht unterstützt, sollten sie aus dem Layout entfernt oder durch ein anderes Element, wie ein statisches Bild, ersetzt werden. Ein Austausch von Websiteelementen ist jedoch nur über entsprechende Templates und einer damit verbundenen serverseitigen Logik realisierbar.

3.3 RESS (Responsive Design + Server Side Components)

Responsive Webdesign ermöglicht, wie bereits in Abschnitt 2.3 erwähnt, die Entwicklung einer einzigen Website die auf allen Geräten funktioniert. Allerdings können einzelne Seitenelemente, wie Bilder, nur durch Lösungen von Drittanbietern optimiert und für jede Geräteklasse angepasst werden. Solche Erweiterungen sind mit clientseitigem Responsive Webdesign nicht möglich und können nur durch serverseitige Logik erreicht werden. Serverseitige Logik, wie sie von separaten mobilen Websites und auch von Adaptive Webdesign verwendet wird (siehe Abb. 2.4). Eine zukunftsfähige Lösung wäre somit, die Vorteile von Responsive Webdesign mit den Vorteilen einer separaten mobilen Website und Adaptive Webdesign zu kombinieren. Basierend auf dieser Tatsache existiert bereits ein Konzept, das versucht, die

Nachteile und Einschränkungen von Responsive Webdesign aufzuheben und diese Umsetzungsstrategie weiterzuentwickeln. Dieser Ansatz wird RESS genannt und wurde durch einen Artikel von Luke Wroblewski bekannt [74]. Er beschreibt RESS als eine Kombination von Responsive Webdesign und serverseitigen Komponenten mit dem Ziel, die User Experience (siehe Abschnitt 2.3.3) einer Website zu verbessern.

3.3.1 Theoretischer Ansatz

Wird eine Website mit Responsive Webdesign erstellt, verfügt diese über eine konsistente URL Struktur über alle Geräte hinweg und passt ihr Layout dynamisch an die jeweiligen Gegebenheiten eines Browsers an. Das bedeutet, dass eine Website über nur eine URL ein geräteübergreifendes Nutzungserlebnis bietet. Allerdings werden bei diesem Ansatz immer alle Inhalte und Medienelemente an jedes Endgerät gesendet, was zu langen Ladezeiten und einer schlechten Performance führen kann. Des weiteren basieren responsive Websites nur auf einer Codebasis. Somit müssen auch austauschbare Inhalte, wie unterschiedliche Navigation-Pattern, in einer Datei implementiert und über CSS ausgetauscht werden.

Serverseitige Lösungen senden immer nur den Inhalt an ein Gerät, der von diesem auch benötigt wird. Somit können Inhalte und Medienelemente für jede Geräteklasse entsprechend optimiert und angepasst werden. Das Problem dieser Lösungen besteht darin, dass jede Website-Version auf ihrer eigenen Codebasis basiert. Dementsprechend müssen alle Versionen immer parallel gewartet und weiterentwickelt werden [74].

Beide Ansätze haben somit ihre Vor- und Nachteile. Hier setzt Wroblewski mit seinem RESS Konzept an. Er beschreibt RESS, in [74], als einen Ansatz, der ein responsives Layout mit serverseitigen Komponenten, zur Optimierung von einzelnen Seitenelementen, verbindet. Das bedeutet, dass eine einzige responsive Website für alle Geräte erstellt wird. Im Anschluss können einzelne Seitenelemente serverseitig gerätespezifisch ausgetauscht werden. Das Konzept ähnelt Adaptive Webdesign (siehe Abschnitt 2.2), das ebenfalls auf serverseitige Komponenten zur Optimierung einer Website setzt. Allerdings basiert Adaptive Webdesign gewöhnlich auf einem adaptiven Layout, das nur für mehrere fixe Auflösungen optimiert wird. Zusätzlich nutzt Adaptive Webdesign die serverseitige Logik oftmals auch zum Austausch des gesamten Website-Inhalts. Somit wird an jede definierte Auflösung und Geräteklasse ein vollständig eigenes Template ausgeliefert. RESS hingegen verbindet die serverseitigen Komponenten von Adaptive Webdesign mit dem flexiblen Grid und responsiven Layout aus Responsive Webdesign. Die serverseitige Logik wird in diesem Konzept nur für die Optimierung einzelner Websiteelemente verwendet. Dabei können Templates für verschiedene Geräteklassen definiert werden. Diese werden vom Server, je nach anfragendem Gerät, ausgetauscht. Soll eine Navigation für eine Bedienung per Finger und

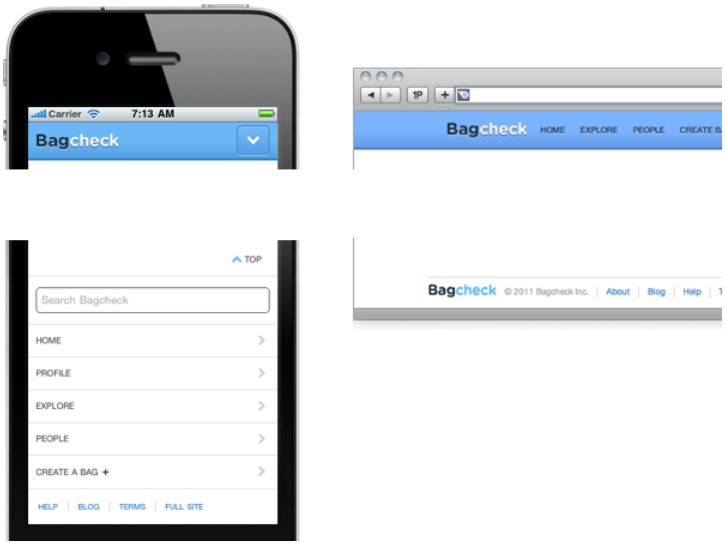


Abbildung 3.5: Erkennt der Server ein mobiles Gerät, wird statt der Desktop Navigation das Template mit der mobilen Navigation eingebunden. Bildquelle: [74].

für eine Nutzung mit Maus und Tastatur optimiert werden, stellt dies meist eine große Herausforderung für Responsive Webdesign dar. Mit RESS besteht die Möglichkeit zwei unterschiedliche Navigation-Pattern zu erstellen und für die jeweilige Geräteklasse zu optimieren. Der Server kann diese erstellten Templates, je nachdem ob es sich um ein mobiles Gerät oder einen Desktop PC handelt, austauschen (siehe Abb. 3.5). Wird von der serverseitigen Geräteerkennung ein mobiles Gerät erkannt, wird die responsive Website mit den mobilen Templates geladen. Ansonsten werden die standard Desktop Templates eingebunden. Somit können die Inhalte der Templates vollständig für die jeweilige Geräteklasse erstellt und optimiert werden. Basierend auf dieser Tatsache können nicht nur einzelne Inhalte, sondern auch Medienelemente, wie Bilder, ausgetauscht werden. Dies ermöglicht eine Reduktion der Dateigröße und eine bessere Performance auf einem mobilen Gerät, da Bilder für das jeweilige Endgerät komprimiert und optimiert werden können. Dabei bleibt das optische Erscheinungsbild der Website intakt, die Performance und das Nutzungserlebnis verbessern sich allerdings erheblich. Das Ergebnis dieser Lösung ist eine responsive Website, die über eine einzige URL erreichbar ist, dabei aber die Optimierung einzelner Seitenbestandteile für definierte Geräteklassen und eine damit verbundene signifikante Verbesserung der Performance erlaubt [74].

3.3.2 Aufbau und Funktion

Für die Umsetzung eines RESS-Systems gibt es keine konkreten Vorgaben. Der Begriff wurde, wie bereits erwähnt, von Luke Wroblewski eingeführt und beschreibt ein Zusammenspiel von Responsive Webdesign und serverseitiger Logik. Allerdings gibt er in seinem Artikel nur einen theoretischen Einblick in sein Konzept. Eine Anleitung für eine technischen Realisierung einer solchen Lösung gibt es nicht. Die Art der Implementierung dieses theoretischen Ansatzes bleibt somit jedem Entwickler selbst überlassen [9, S. 16].

Für die Bestandteile einer RESS Website gibt es keine definierten Angaben. Die Idee dahinter besteht darin, auch bei Responsive Webdesign eine Optimierung von einzelnen Seitenelementen für definierte Geräteklassen zu ermöglichen. Teilweise wird die serverseitige Logik nur für die Reduzierung und Optimierung von Bildern verwendet. Die Integration eines Template-Systems kann zusätzlich viele weitere Verbesserungen ermöglichen. So können einzelne Seitenelemente ausgetauscht oder Inhalte, wie Flash-Animationen, nur für Geräte die diese auch unterstützen, eingebunden werden. Somit wird die Ausgabe von Inhalten teilweise vom Client auf den Server verlagert, was zu signifikanten Performanceverbesserungen führen kann. Vor der Umsetzung einer RESS Website müssen dementsprechend die Anforderungen und gewünschten Adaptionen des Systems festgelegt werden [9, S. 45].

Ein RESS-System basiert meist auf einer serverseitigen Geräteerkennung (siehe auch Abschnitt 2.1.1). Mithilfe von *Device Description Repositories* wie WURFL oder DeviceAtlas ist es möglich, mehr Details über ein anfragendes Gerät zu erfahren. So können detaillierte Informationen wie die Geräteklasse oder die, für ein RESS-System relevante, Auflösung eines Endgeräts ermittelt werden. Die Bibliotheken stehen in verschiedenen Programmiersprachen zur Verfügung. Allerdings ist deren Nutzung meist an strenge Lizenzbestimmungen gebunden. Dabei handelt es sich bei einem Großteil der Systeme um offline Datenbanken. Diese können dementsprechend immer nur bekannte *User-Agent-Strings* berücksichtigen und die verwendeten Datensätze müssen somit laufend auf die neueste Version aktualisiert werden. Allerdings bieten diverse Anbieter auch cloudbasierte Lösungen für ihre Systeme an. Bei diesen Lösungen werden die Gerätedaten auf dem Webserver des Anbieters automatisch aktualisiert und folglich immer aktuell gehalten. Dementsprechend entfällt der permanente Wartungsaufwand, der aber oftmals durch höhere Lizenzgebühren kompensiert wird [4, S. 311–315].

Ein performanceorientiertes und funktionales RESS-System benötigt eine Möglichkeit Daten zwischen dem Server und dem Client auszutauschen und sich Informationen über mehrere Requests hinweg zu merken. Dies wird gewöhnlich über ein Cookie realisiert. In diesem Cookie wird die ermittelte Auflösung des Geräts gespeichert. Somit muss bei einem Reload der Website die aufwändige Analyse des *User-Agent-Strings* nicht erneut durchgeführt

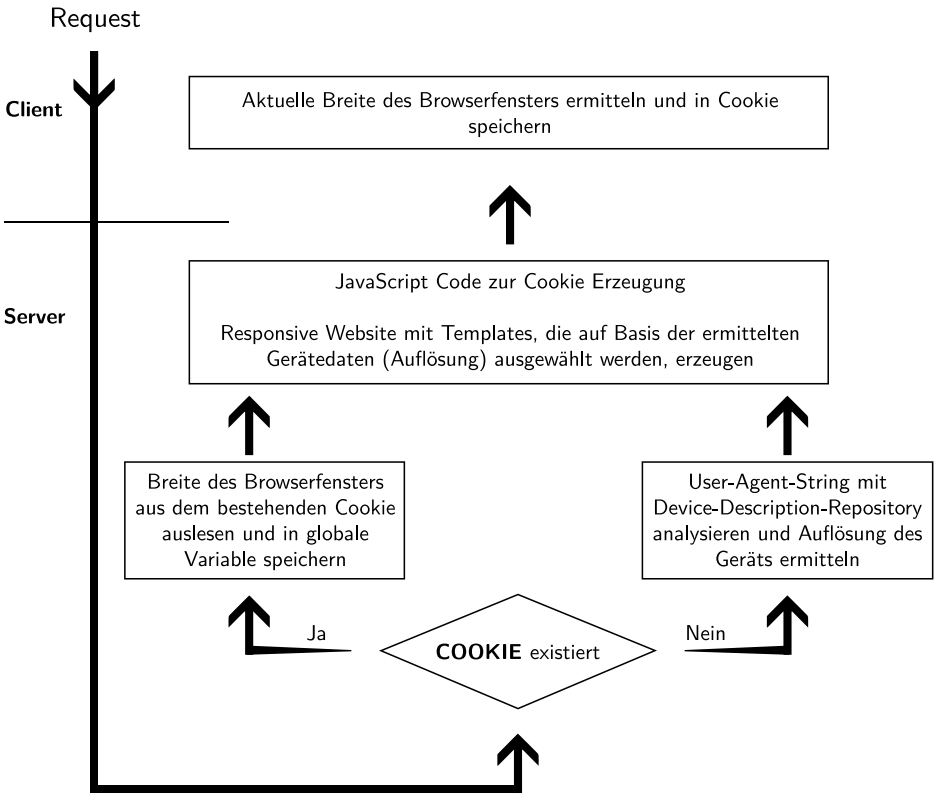


Abbildung 3.6: Hypothetisches Beispiel für den Informationsfluss in einem RESS-System. Bildquelle: [9, S. 48].

werden. Wie bereits in Abschnitt 2.1.1 erwähnt ist die *User-Agent-Detection* fehleranfällig und kann durch *User-Agent-Spoofing* manipuliert werden. Um einer fehlerhaften Analyse des *User-Agent-Strings* entgegen zu wirken, sollte die aktuelle Breite des Browserfensters auch clientseitig per JavaScript überprüft werden. Unterscheiden sich die Ergebnisse der serverseitigen und clientseitigen Überprüfung stark voneinander, sollte auf die clientseitige Analyse vertraut werden. Dieses Ergebnis wird im Cookie gespeichert und verhindert somit ebenfalls eine erneute Ausführung der *User-Agent-Detection* (siehe Abb. 3.6). Dabei kann die clientseitige Überprüfung mit JavaScript auch als Fallback-Lösung gesehen werden. Sollte die serverseitige Überprüfung nicht funktionieren oder ein falsches Ergebnis liefern, kann das System nach einem Reload auf den mit JavaScript ermittelten Wert aufbauen [9, S. 45–48].

Basierend auf der gespeicherten Auflösung können im Anschluss einzelne Seitenbestandteile in einem responsiven Layout, das an alle Browser ausge-

liefert wird, ausgetauscht werden [67]. Dies wird gewöhnlich durch eine serverseitige Programmiersprache wie PHP realisiert. Dabei werden die Inhalte einzelner Seitenbestandteile in eigenen PHP Dateien implementiert und je nach ermittelter Auflösung des Browsers eingebunden. Das folgende Beispiel zeigt die Einbindung eines Websitefragments, das nur angezeigt wird, wenn die ermittelte Breite des Browserfensters zwischen 320 und 640 Pixel liegt [44]:

```
<?php
  if ($RESS["width"] >= 320 && $RESS["width"] <= 640) {
?>

<div class="mobile-ad max-320">
  <?php include "fragments/ads/320.php"?>
</div>

<?php
  }
?>
```

Die globale Variable *$RESS* dient dabei als Speicher für die vom RESS-System ermittelte Auflösung. Auch für den Austausch von Bildern kann die gespeicherte Auflösung verwendet werden. Dabei muss nur das *src* Attribut des *<img>* Elements auf die entsprechende Bilddatei gesetzt werden [74]. Basierend auf diesen Verbesserungen konvertiert eine responsive Website von einer fixen Größe, die an alle Geräte gesendet wird, zu einem hochflexiblen Umfang, der sich an die Auflösung des anfragenden Geräts anpasst [1, S. 103–104].

3.3.3 Zusammenfassung

Responsive Webdesign ist eine innovative Lösung die es Webdesignern ermöglicht, für mobile Geräte optimierte Websites, zu erstellen. Allerdings bietet dieses Konzept nicht nur Vorteile, sondern ist auch mit einer Reihe von Nachteilen und Einschränkungen verbunden. Der theoretische RESS Ansatz von Luke Wroblewski versucht, diese Einschränkungen durch die Einbindung einer serverseitigen Logik aufzuheben. Das Konzept ähnelt Adaptive Webdesign, allerdings erweitert RESS ein responsives Layout um zusätzliche serverseitige Abfragen. Durch die Ermittlung und Nutzung von Geräteinformationen, wie der aktuellen Auflösung des Browserfensters, können einzelne Websitekomponenten für die jeweilige Geräteklasse optimiert werden. Dies ermöglicht den Austausch von einzelnen Seitenbestandteilen, wie einem Navigation-Pattern, oder auch die auflösungsabhängige Optimierung von Medienelementen. Dabei wird der nötige Rechenaufwand teilweise vom Client auf den Server verlagert. Dementsprechend wird das zu übertragende Datenvolumen deutlich reduziert und die Performance für jede definierte Geräteklasse optimiert.

Da die Realisierung einer solchen kombinierten Lösung einen erheblichen serverseitigen Programmieraufwand erfordert, liegt die entscheidende Herausforderung in der erforderlichen Zusammenarbeit von Webentwickler und Webdesigner im gesamten Layout- und Design-Prozess. Webdesigner beschäftigen sich gewöhnlich mit dem Frontend und dem Layout einer Website und sind demnach oftmals nicht mit der Programmierung von serverseitigen Lösungen vertraut. Webentwickler sind hingegen häufig für die technologische Grundlage und Funktionalitäten einer Website verantwortlich. Allerdings wäre es wünschenswert, diese beiden Aufgabengebiete getrennt zu halten [68]. Dementsprechend würde eine Möglichkeit, die es Webdesignern erlaubt, von den Vorteilen dieses neuen Konzeptes zu profitieren, eine ideale Lösung darstellen.

Kapitel 4

Entwurf

Das folgende Kapitel beschreibt die eigenen Überlegungen zu einer Verbesserung von Responsive Webdesign, durch die Einführung einer serverseitigen Komponente. Dabei wird auf das bereits beschriebene theoretische RESS Konzept (siehe Abschnitt 3.3) aufgebaut. Ziel ist es, Webdesigner und Webentwickler bei der Erstellung einer RESS Website zu unterstützen. Dabei soll die serverseitige Komponente in einen Framework-Prototyp integriert werden und somit den Programmieraufwand für einen Webdesigner minimieren. Der Fokus der angestrebten Verbesserungsmaßnahmen liegt auf einer Verbesserung der Performance durch eine automatische Auslieferung von auflösungsoptimierten Bildern und der Anpassung bzw. dem Austausch von Seitenelementen, basierend auf definierten Geräteklassen. Zusätzlich sollen die einzelnen Komponenten des Frameworks flexibel austauschbar sein und eine Wiederverwendung des Systems ermöglichen.

4.1 Anforderungen

Die Verbreitung von mobilen Geräten hat in den letzten Jahren stark zugenommen. Diese Tatsache wirkt sich wesentlich auf die Gestaltung und Aufbereitung von Websites aus. Das mobile Web stellt neue Anforderungen an Websites und Umsetzungsstrategien. Webdesign muss flexibler werden und sich individuell an die Gegebenheiten eines Endgeräts anpassen [13]. Displaygrößen und Auflösungen sind bei mobilen Geräten meist wesentlich kleiner als bei einem Desktop PC. Darüber hinaus gibt es eine enorme Vielfalt an verschiedenen Geräten. Dementsprechend müssen auch mobile Websites verschiedene Auflösungen gleichzeitig unterstützen. Auch die verfügbare Netzwerkverbindung und Bandbreite stellt für mobile Geräte ein relevantes Kriterium dar. Wird eine Website nicht entsprechend optimiert, können lange Ladezeiten entstehen und einen Website-Besucher zu einem sofortigen Verlassen der Website bewegen [3]. Obendrein rufen Besucher eine Website sowohl von mobilen Geräten als auch von Desktop PCs auf und erwarten,

dass ihnen derselbe Inhalt auf all ihren Geräten funktional aufbereitet zur Verfügung steht. Somit ist es notwendig, dass bei der Erstellung eines Webauftritts nicht mehr nur die sich ständig entwickelnden Fähigkeiten und Ressourcen dieser Geräte fokussiert werden, sondern auch Konsistenz und eine geräteübergreifende User Experience (siehe Abschnitt 2.3.3) im Zentrum der Entwicklung stehen [18, S. 6–7].

Wie bereits erwähnt, ist Responsive Webdesign (siehe Abschnitt 2.3) eine innovative und zukunftsorientierte Möglichkeit responsive, anpassungsfähige Websites zu erstellen. Es ermöglicht die Umsetzung einer einzigen konsistenten Website, die ihr Layout automatisch an die Displaygröße eines beliebigen Endgeräts anpasst. Responsive Webdesign repräsentiert somit eine Universallösung, da es einen Kompromiss aus Desktop und mobiler Website darstellt. Basierend auf dieser Tatsache bringt dieses Konzept eine Reihe von Vorteilen, jedoch auch diverse Nachteile und Einschränkungen mit sich.

Responsive Webdesign nutzt im Gegensatz zu Adaptive Webdesign (siehe Abschnitt 2.2) keine serverseitigen Optimierungsmaßnahmen. Somit wird dieselbe Datenmenge an alle Geräte gesendet, was, vor allem auf mobilen Geräten, zu einer schlechten Performance führen kann. Aus diesem Grund soll ein Framework-Prototyp entwickelt werden, der, auf Basis des RESS Konzeptes, Responsive Webdesign um eine serverseitige Komponente erweitert. Die Nutzung des Frameworks soll dabei keine serverseitigen Programmierkenntnisse voraussetzen bzw. keinen Programmieraufwand erfordern und somit einen Webdesigner bei der Erstellung einer entsprechenden Website unterstützen. Durch die serverseitige Erweiterung sollen die folgenden Nachteile von Responsive Webdesign adressiert und dementsprechend die Performance einer Website verbessert werden.

4.1.1 Optimierung von Bildern

Eines der größten Probleme des Responsive Webdesign Konzeptes sind Bilder. Diese werden auf mobilen Geräten meist nur skaliert und nicht komprimiert. Folglich wird eine hochauflösende Bilddatei, die an einen Desktop Monitor gesendet wird, ebenso an ein mobiles Gerät, mit einem niedriger auflösenden Monitor, gesendet, auch wenn dieses Endgerät das Bild nicht in seiner nativen Auflösung darstellen kann. Demzufolge kommt es bei einer langsamen Netzwerkverbindung oft zu langen Ladezeiten und einer schlechten Performance. Das Responsive Webdesign Konzept selbst bietet keine Lösung für dieses Problem. Aktuell gibt es eine Reihe von Drittanbieter-Lösungen die meist durch die Nutzung von JavaScript einen flexiblen Austausch oder eine Optimierung von Bildern für unterschiedliche Geräte und Auflösungen ermöglichen (siehe Abschnitt 3.2). Die Nutzung solcher Tools kann sich jedoch ebenso negativ auf die Performance auswirken. Sollte ein Besucher JavaScript deaktiviert haben, kann ein entsprechendes Script nicht ausgeführt werden und im Anschluss zusätzlich zu Darstellungsproblemen

führen. Mit *<picture>*, *srcset* und *sizes* sind neue HTML-Elemente und Attribute in die Spezifikation mitaufgenommen worden, die zukünftig eine Realisierung von flexiblen Bilder über reines HTML ermöglichen. Allerdings werden diese neuen Elemente und Attribute noch nicht von jedem Browser unterstützt, wodurch immer eine zusätzliche Fallback-Lösung implementiert werden muss. Die Nutzung dieser neuen HTML-Elemente führt darüber hinaus zu einem deutlichen Mehraufwand für Webdesigner, da nun für jeden *Breakpoint* eine eigene optimierte Bildversion erstellt und entsprechend eingebunden werden muss [7, S. 47].

Eine Möglichkeit die eine automatisierte Einbindung und Optimierung von flexiblen Bildern ohne JavaScript ermöglicht und dabei auf dem gängigen und von allen Browsern unterstützten *<img>* HTML-Element bzw. *src* Attribut basiert, würde somit eine ideale Lösung darstellen. Diese könnte zukünftig auch mit den neuen Elementen *<picture>* und *srcset* kombiniert und als entsprechende Fallback-Lösung verwendet werden. Dabei soll eine automatische Optimierung von Bildern am eigenen Webserver erfolgen und damit unabhängig von Drittanbieter-Lösungen funktionieren. Eine solche Lösung soll in weiterer Folge implementiert und in den Framework-Prototyp integriert werden.

Auch der Austausch und die Optimierung von Websiteelementen für Geräteklassen ist mit Responsive Webdesign nicht möglich.

4.1.2 Gerätespezifische Inhalte

Responsive Webdesign stellt, wie bereits erwähnt, eine Universallösung dar, da eine einzige Website sowohl für Desktop PCs als auch für mobile Geräte erstellt wird. Allerdings benötigen verschiedene Geräteklassen auch verschiedene gerätespezifische Inhalte, was einen wesentlichen Nachteil des Responsive Webdesign Konzeptes darstellt. Beispielsweise sollte die Navigation einer Website auf einem Desktop PC eine Bedienung mit Maus und Tastatur, jedoch auf einem Smartphone oder Tablet, mit Touchdisplay, eine Bedienung per Finger ermöglichen. Es müssen somit verschiedene Navigation-Pattern implementiert und entsprechend ausgetauscht werden. Ein weiteres Beispiel ist eine Flash-Animation oder ein eingebundenes Video. Diese Medien werden möglicherweise von einem mobilen Webbrowser nicht unterstützt und angezeigt. In diesem Fall sollten sie durch einen anderen Inhalt, wie ein statisches Bild, ersetzt werden. Ein Austausch und eine Optimierung von einzelnen Seitenelementen bzw. Medieninhalten für spezifische Geräteklassen, ist jedoch mit clientseitigem Responsive Webdesign nicht möglich. Inhalte können nur an bestimmten *Breakpoints* wechselweise über CSS ausgeblendet werden. Die ausgeblendeten Seitenbestandteile werden somit nur für den Website-Besucher versteckt, jedoch im Hintergrund trotzdem geladen. Das bedeutet, dass auch auf einem mobilen Gerät der gesamte Inhalt der Desktop Version heruntergeladen wird, was wiederum zu einer schlechten

Performance führen kann.

Eine optimale Lösungsmöglichkeit würde die Auslagerung von einzelnen Seitenbestandteilen in separate Dateien, sogenannte Partials, darstellen. Partials sind eigene Template-Dateien die eine mehrfache Einbindung und Wiederverwendung des Inhalts ermöglichen. Diese sollen im Anschluss, je nach Geräteklasse, automatisch ausgetauscht werden. Somit kann der Inhalt eines Partials vollständig für die entsprechende Geräteklasse optimiert werden und eine optimale Performance garantieren. Für diesen Zweck soll eine entsprechende Template-Engine implementiert und in den Framework-Prototyp integriert werden.

4.1.3 Individuelle Konfiguration

Bei der Erstellung einer Website wird häufig auf hilfreiche Drittanbieter-Lösungen zurückgegriffen. Beispielsweise wird zur Ermittlung von Gerätedaten ein gängiges *Device Description Repository* oder zur Umsetzung des Website-Frontends ein innovatives Web Framework verwendet. Diese Tools sind gewöhnlich an die verschiedensten Lizenzbestimmungen gebunden, die ihre Nutzung entsprechend einschränken. Das bedeutet, dass grundsätzlich für jedes Projekt eine individuelle und zulässige Auswahl solcher Drittanbieter-Lösungen erfolgen muss.

Um die Wiederverwendbarkeit des in den folgenden Abschnitten beschriebenen, Framework-Prototyps zu erhöhen, soll dieser eine hohe Flexibilität aufweisen. Deshalb ist es relevant, dass einzelne lizenzgebundene Bestandteile des Systems austauschbar sind und einem Webdesigner eine individuelle Konfiguration der Framework Komponenten ermöglicht wird.

4.2 Aufbau und Funktion

Das entwickelte Konzept basiert auf dem theoretischen RESS Ansatz von Luke Wroblewski (siehe Abschnitt 3.3). Es verbindet somit ein responsives Layout mit serverseitigen Komponenten, zur Optimierung von einzelnen Seitenelementen. Für die Umsetzung eines solchen Systems gibt es allerdings, wie in Abschnitt 3.3 erwähnt, keine konkreten Vorgaben. Die Nutzung dieses Konzeptes ist für einen Webdesigner somit schwer möglich, da eine Implementierung meist mit einem erheblichen serverseitigen Programmieraufwand verbunden ist. Basierend auf dieser Tatsache wird ein Framework-Prototyp entwickelt, der die serverseitige Komponente des Konzeptes integriert. Um eine Optimierung bzw. Verbesserung von Responsive Webdesign zu erreichen, wird die serverseitige Logik zur automatisierten Auslieferung von auflösungsoptimierten Bildern und der Anpassung bzw. dem Austausch von Seitenelementen, basierend auf definierten Geräteklassen, verwendet. Ziel ist, dass das Framework eine einfache Konfiguration ermöglicht, keinen server-

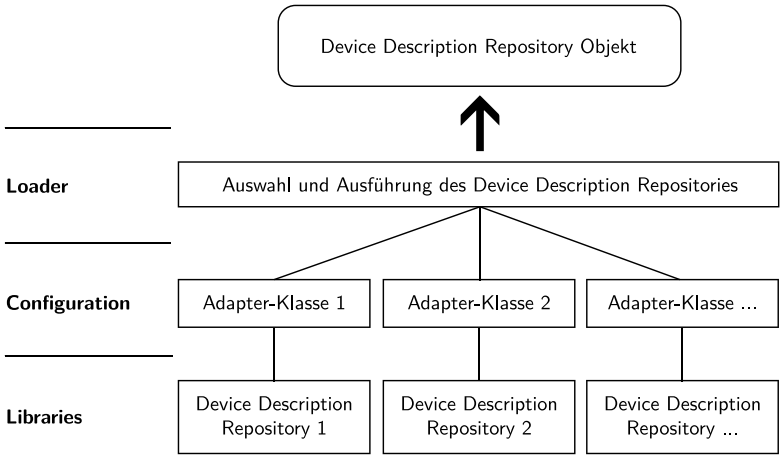


Abbildung 4.1: Um eine einheitliche Nutzung der verschiedenen *Device Description Repositories* zu ermöglichen, werden zusätzliche Adapter-Klassen benötigt.

seitigen Programmieraufwand erfordert und durch die genannten Optimierungsmaßnahmen die Performance von Responsive Webdesign verbessert.

4.2.1 Serverseitige Geräteerkennungsbibliotheken

Die angestrebte serverseitige Optimierung von diversen Seitenbestandteilen ist nur möglich, wenn der Server bestimmte Informationen über das anfragende Gerät kennt. Zu diesem Zweck bietet sich die Nutzung einer serverseitigen Geräteerkennung (siehe Abschnitt 2.1.1) an. Im Web gibt es bereits eine Reihe solcher Lösungen die eine Einbindung am eigenen Webserver ermöglichen und in verschiedenen Programmiersprachen zur Verfügung stehen. Diese Bibliotheken führen die aufwändige Analyse des *User-Agent-Strings* eines Geräts durch und extrahieren bestimmte gerätespezifische Informationen. Auf Basis der ermittelten Daten können im Anschluss verschiedene serverseitige Optimierungsmaßnahmen durchgeführt werden.

Allerdings basiert jede Bibliothek auf einer eigenen Lizenzbestimmung, die sie in ihrer Funktionsweise beschränkt bzw. die Nutzung der Lösung nur für bestimmte Projekte erlaubt. Um die Flexibilität des Frameworks zu erhöhen und die Einsatzmöglichkeit zu erweitern, werden verschiedene gängige Bibliotheken mit unterschiedlichen Lizenzbestimmungen implementiert. Dabei werden die individuellen Einstellungen eines *Device Description Repositories* über die Framework Konfiguration ermöglicht. Die Bibliotheken unterscheiden sich auch hinsichtlich ihrer Funktionsweise und der abfragbaren Gerätedaten. Es wird somit eine zusätzliche Abstraktionsschicht benötigt die die Funktionsweisen der verschiedenen Lösungen vereinheitlicht

(siehe Abb. 4.1). In diesen Adapter-Klassen wird die benötigte Konfiguration der individuellen *Repositories* und die Abfrage der gewünschten Gerätedaten durchgeführt. Somit können die *Device Description Libraries* jederzeit ausgetauscht oder aktualisiert werden, da der Code der Bibliotheken nicht angepasst oder modifiziert werden muss. Die Auswahl und der Wechsel des gewünschten *Repositories* kann über die globale Konfiguration erfolgen. Über eine entsprechende *Loader*-Methode wird die Adapter-Klasse der gewählten Bibliothek geladen, die wiederum den Aufruf des *Repositories* durchführt. Die Daten werden anschließend in einem einheitlichen *Device Description Repository* Objekt gespeichert und dem System zur Verfügung gestellt. Der Framework-Prototyp nutzt somit nur die Gerätedaten des *Device Description Repository* Objekts und benötigt keine Kenntnisse über und auch keinen direkten Zugriff auf die implementierten Geräteerkennungsbibliotheken.

4.2.2 Geräteklassifikation

Device Description Repositories ermöglichen den Zugriff auf ein breites Spektrum an Geräteinformationen. Auf Basis dieser Informationen lässt sich ein Gerät meist bis hin zu dem installierten Betriebssystem charakterisieren. Allerdings ist die Anzahl der abfragbaren Daten wiederum an bestimmte Lizenzbedingungen gebunden und hat, je nach Lösung, auch Auswirkungen auf die Performance der Abfrage. Aus diesem Grund werden die benötigten Gerätedaten für den Framework-Prototyp auf ein Minimum beschränkt.

Für eine auflösungsabhängige Optimierung von Bildern wird die Breite des Displays eines Geräts in Pixel benötigt. Auf Basis dieser Information können die Bilder einer Website im Anschluss entsprechend reduziert und gespeichert werden. Sie werden nicht mehr nur skaliert, sondern individuell an die Auflösung eines Geräts angepasst. Folglich werden an ein Gerät nur mehr Bilder gesendet, die es auch in der vollen Auflösung darstellen kann.

Zusätzlich soll der Framework-Prototyp auch den Austausch von Seitenbestandteilen, basierend auf definierten Geräteklassen, ermöglichen. Dementsprechend werden zu der Displayauflösung eines Geräts auch noch die Geräteklasse, also die Kategorie in die ein Gerät fällt, benötigt. Für die Entwicklung des Frameworks werden die gängigen Geräteklassen Desktop, Tablet und Mobile festgelegt. Als Mobile wird ein mobiles Geräte klassifiziert. Verfügt es zusätzlich über eine höhere Auflösung oder findet sich in dem analysierten *User-Agent-String* ein Hinweis, dass es sich um ein Tablet handelt, wird es als Tablet eingestuft. Wird ein Gerät nicht als mobiles Gerät erkannt, wird es als desktopartiges Gerät klassifiziert. Die für den Framework-Prototyp definierten Geräteklassen können zukünftig erweitert werden. So ist beispielsweise auch eine eigene Klasse für TVs denkbar. Allerdings ist eine mögliche Klassifizierung einer Geräteklasse immer an die verfügbaren Daten des verwendeten *Device Description Repositories* gebunden. Basierend auf der so ermittelten Information können im Anschluss einzelne Seitenbestand-

teile ausgetauscht und für die jeweilige Geräteklasse optimiert werden. Für den Fall das der *User-Agent-String* eines Geräts nicht erkannt wird, kann in der Konfiguration eine Fallback-Geräteklasse definiert werden. Schlägt die Analyse der Gerätedaten fehl oder kann ein Gerät nicht eindeutig in eine der definierten Geräteklassen eingeordnet werden, wird die Website auf Basis der festgelegten Fallback-Klasse ausgeliefert.

Mobile Geräte existieren heute, ebenso wie ihre Desktop Pendants, in den verschiedensten Größen und mit den unterschiedlichsten Auflösungen. Aus diesem Grund werden die beiden ermittelten Geräteinformationen, Auflösung und Geräteklasse, nicht kombiniert verwendet, sondern getrennt voneinander verarbeitet. Somit werden an ein modernes Smartphone, das eine hohe Auflösung aufweist, ebenfalls hoch aufgelöste Bilder ausgeliefert. Die gerätespezifischen Seitenbestandteile werden allerdings für die entsprechende Geräteklasse angepasst und ausgetauscht. Die Auflösung und die Klasse eines Geräts werden in der Adapter-Klasse des ausgewählten *Device Description Repositories* ermittelt und anschließend in einem *Device Description Repository* Objekt gespeichert. Dieses Objekt bietet dem Framework-Prototyp Zugriff auf die ermittelten Daten des anfragenden Geräts und ermöglicht die Nutzung und weitere Verarbeitung dieser Informationen.

4.2.3 Ablauf der Geräteerkennung

Eine serverseitige Geräteerkennung ist eine sehr gute Lösung um bestimmte Gerätedaten zu ermitteln. Allerdings kann sie, wie bereits in Abschnitt 2.1.1 erwähnt, auch fehlschlagen und falsche oder unvollständige Informationen liefern. Aus diesem Grund wird in den Framework-Prototyp auch eine zusätzliche clientseitige Ermittlung der Gerätedaten integriert. Dabei werden am Client ebenfalls die aktuelle Auflösung des Browserfensters und die Geräteklasse bestimmt.

Wird eine Website im Browser aufgerufen, wird zuerst überprüft ob der Besucher JavaScript und Cookies aktiviert hat. Parallel dazu wird eine serverseitige Geräteerkennung durchgeführt und die Website auf Basis der ermittelten Gerätedaten ausgeliefert (siehe Abb. 4.2). Ist JavaScript im Browser aktiv und hat der Website-Besucher das Speichern von Cookies zugelassen, werden die Geräteklasse und die aktuelle Breite des Browserfensters auch am Client ermittelt. Diese Daten werden im Anschluss mit den ermittelten Informationen der serverseitigen Geräteerkennung abgeglichen. Hierfür wird eine Kommunikationsmöglichkeit zwischen Server und Client benötigt. Dabei werden die am Client ermittelten Gerätedaten in ein Cookie gespeichert, anschließend auf der Serverseite extrahiert und mit den Daten des *Device Description Repository* Objekts verglichen. Ist die Ermittlung auf der Clientseite fehlgeschlagen oder sind die Informationen nicht vollständig, werden die entsprechenden Angaben von den ermittelten Daten des Servers übernommen. Widersprechen sich die Daten von Server und Client, wird

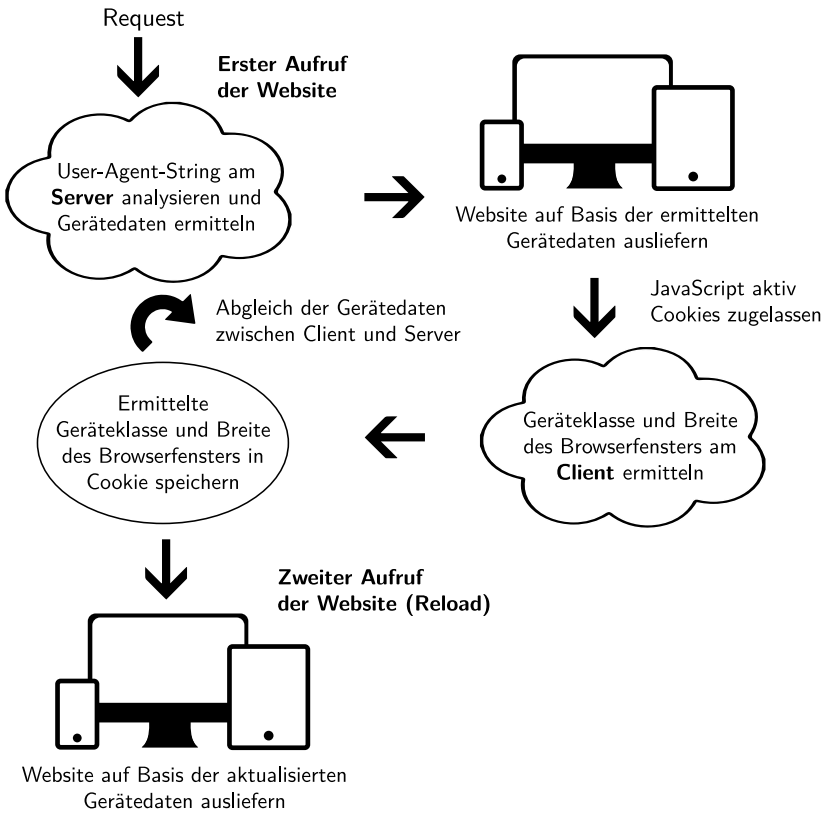


Abbildung 4.2: Beispiel für den vollständigen Workflow bei der Geräteerkennung auf Server- und Clientseite und dem anschließenden Abgleich der ermittelten Gerätedaten.

auf die ermittelten Gerätedaten des Clients vertraut, da bei einer Überprüfung auf Clientseite immer auf die aktuellen Werte des Browsers zugegriffen werden kann. Ändert ein Besucher beispielsweise die Breite des Browserfensters, kann diese neue Information am Client ermittelt werden, wohingegen der Server nur die vollständige Auflösung des Geräts aus der Datenbank des *Device Description Repositories* abrufen kann. Wird die Website nun neu geladen, wird der Inhalt auf Basis der aktualisierten Geräteinformationen ausgeliefert. Diese kombinierte Vorgehensweise ist notwendig, da eine clientseitige Ermittlung der Gerätedaten erst nach dem ersten Laden der Website möglich ist. Die Informationen stehen dann bereits zur Verfügung. Die austauschbaren Websitekomponenten können, wenn nötig, aber erst nach einem erneuten Laden der Website aktualisiert werden. Zusätzlich dient die serverseitige Geräteerkennung auch als Fallback-Lösung. Hat ein Besucher JavaScript bzw. Cookies in seinem Browser deaktiviert, ist eine clientseitige Ermittlung der Gerätedaten nicht möglich. Allerdings kann auf Basis der

am Server ermittelten Daten ebenfalls ein optimiertes Ergebnis ausgeliefert werden.

4.2.4 Optimierung von Bildern

Bilder stellen, wie bereits in Abschnitt 4.1.1 erwähnt, eine der größten Herausforderungen des Responsive Webdesign Konzeptes dar. Aus diesem Grund wird eine automatisierte, auflösungsabhängige Optimierung dieser Medien in den Framework-Prototyp integriert. Dabei können Bilder über das *src* Attribut des gängigen *<img>* HTML-Elements in die Website eingebunden werden. Je nach Auflösung des anfragenden Geräts werden sie im Anschluss am Server automatisch reduziert und ausgeliefert. Auf diese Weise wird der Client entlastet und der nötige Rechenaufwand auf den Server verlagert. Zusätzlich werden keine Drittanbieter-Tools benötigt und die Website bleibt dementsprechend unabhängig und performant. In der Konfiguration können die *Breakpoints* des individuellen responsiven Layouts angegeben werden. Basierend auf diesen Pixelangaben wird eine Ordnerstruktur am Webserver erstellt. Wird nun eine Website von einem Gerät aufgerufen, wird die aktuelle Breite des Browserfensters ermittelt (siehe Abschnitt 4.2.3). Liegt diese Breite innerhalb eines definierten Layout-*Breakpoints* wird überprüft, ob in dem entsprechenden Ordner bereits eine Bilddatei in der benötigten Auflösung vorliegt. Ist diese noch nicht vorhanden, wird die hochauflösende Bilddatei automatisch reduziert und für zukünftige Anfragen in dem erstellten Ordner abgelegt (siehe Abb. 4.3). Folglich entfällt der händische Reduzierungsaufwand, da dieser vom Server übernommen wird.

Diese Vorgehensweise ermöglicht nicht nur die Reduzierung von eingebundenen Bilddateien, sondern auch die manuelle Erstellung und automatisierte Einbindung von unterschiedlichen Bildversionen oder auch verschiedenen Bildausschnitten. Soll auf einem mobilen Gerät mit einer niedrigeren Auflösung beispielsweise eine andere Bilddatei angezeigt oder ein spezieller Bildausschnitt vergrößert werden, kann die Datei manuell erstellt und in dem entsprechenden Ordner gespeichert werden. Wird die Website dann von einem mobilen Gerät aufgerufen, wird die hochauflösende Bildversion nicht reduziert, sondern die manuell erstellte Bilddatei ausgeliefert. Zusätzlich kann in der Konfiguration auch eine Überprüfung der Netzwerkgeschwindigkeit aktiviert werden. Dabei wird bei einem Aufruf der Website die aktuell verfügbare Bandbreite des Geräts ermittelt und gespeichert. Grundsätzlich werden an ein mobiles Gerät mit einem hochauflösenden Display auch hochauflösende Bilder gesendet. Verfügt dieses allerdings über eine schlechte Netzwerkverbindung, können somit, je nach Konfiguration, auch niedrig aufgelöste Bilder ausgeliefert werden. Folglich wird die Performance bei einer niedrigen Bandbreite erhöht. Verfügt das Gerät zu einem späteren Zeitpunkt wieder über eine bessere Netzwerkverbindung, werden wiederum die hochauflösenden Bilder angezeigt.

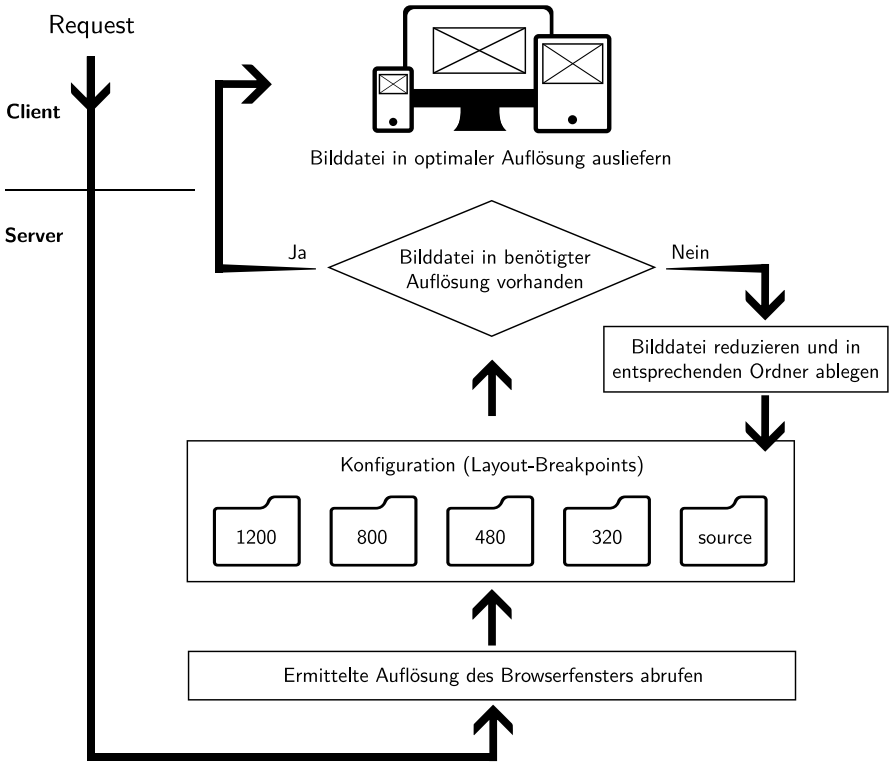


Abbildung 4.3: Auf Basis der ermittelten Auflösung eines Geräts wird eine optimale Bilddatei ausgewählt. Ist keine Datei in der benötigten Auflösung vorhanden, wird die hochauflösende Version reduziert und gespeichert.

Exkurs <picture>, srcset und sizes

Wie in den Anforderungen erwähnt, werden die neuen HTML-Elemente und Attribute *<picture>*, *srcset* und *sizes* noch nicht von jedem Browser unterstützt (siehe Abschnitt 3.2.1). Sollen diese Elemente bereits heute bei der Erstellung einer Website verwendet werden, muss somit immer eine zusätzliche Fallback-Lösung implementiert werden. Dies wird gewöhnlich über ein JavaScript Polyfill Plugin realisiert (siehe Abschnitt 3.2.2).

Die in dem Framework integrierte, auflösungsabhängige Bildoptimierung kann jedoch auch mit diesen neuen Elementen kombiniert werden, wodurch eine zusätzliche Polyfill Lösung obsolet wird. Durch die automatische Optimierung der hochauflösenden Bilder, entfällt außerdem der händische Reduzierungsaufwand. Folglich müssen die einzelnen erstellten Bildversionen nur mehr in dem *<picture>* Element bzw. über das *srcset* Attribut spezifiziert werden. Sollten die neuen Elemente von einem Browser nicht unterstützt werden, ermöglichen sie die Angabe einer zusätzlichen Fallback-Bilddatei.

Diese wird wiederum über das gängige und von allen Browsern unterstützte *src* Attribut eines *<img>* Elements spezifiziert. Dementsprechend kann auch hier die, in dem Framework integrierte und im vorangegangenen Abschnitt 4.2.4 beschriebene, Bildoptimierung angewendet werden. Die Vorgehensweise ist dabei identisch. Somit werden an jedes Gerät optimal aufgelöste Bilder ausgeliefert. Werden die HTML-Elemente bzw. Attribute von einem Browser unterstützt, erfolgt die Auswahl und Auslieferung der korrekten Bilddatei über die entsprechende HTML-Angabe. Ist dies nicht der Fall, wird die Reduzierung und Auslieferung der optimalen Bilddatei durch die integrierte Optimierungsfunktion gewährleistet.

4.2.5 Gerätespezifische Inhalte

Der Austausch von Seitenbestandteilen, basierend auf definierten Geräteklassen, ist mit clientseitigem Responsive Webdesign nicht möglich. Eine responsive Website kennt die Klasse bzw. Kategorie eines anfragenden Geräts nicht. Sie passt ihr Layout nur dynamisch an die verfügbare Breite des Browserfensters an. Durch die im Framework integrierte Analyse der Gerätedaten kann jedoch die entsprechende Geräteklasse ermittelt werden. Dabei werden erkannte Geräte in die drei Geräteklassen Desktop, Tablet und Mobile eingeteilt. Sollte ein Gerät nicht erkannt werden, wird die, in der Konfiguration festgelegte, Fallback-Geräteklasse verwendet.

Auf Basis dieser Information können im Anschluss bestimmte Inhalte für jede Geräteklasse optimiert und ausgetauscht werden. Die Realisierung und effiziente Nutzung einer solchen Lösung wird durch die Implementierung einer Template-Engine ermöglicht. Diese wird in den Framework-Prototyp integriert und erfordert keinen serverseitigen Programmieraufwand. Wird eine Website mit der Template-Engine erstellt, können einzelne Seitenbestandteile in eigene Partial Dateien ausgelagert werden. Diese werden anschließend in gerätespezifischen Template Ordnern abgelegt. Die entsprechenden Ordner stehen für jede spezifizierte Geräteklasse zur Verfügung. Somit können einzelne Websitebestandteile, wie die Navigation, in ein eigenes Partial ausgelagert und für jede Geräteklasse individuell optimiert werden. Die Einbindung und Nutzung der einzelnen Partials erfolgt anschließend über simple Verknüpfungen. Dabei muss nur der Name der Partial Datei in einem entsprechenden Tag angegeben werden. Wird die Website von einem Gerät aufgerufen, wird die Geräteklasse ermittelt und anschließend die Partials, aus dem gerätespezifischen Ordner, in das übergeordnete Layout eingesetzt (siehe Abb. 4.4). Diese Vorgehensweise ermöglicht auch die Ausnahme von Inhalten, wenn diese beispielsweise nicht benötigt oder von einer Geräteklasse nicht unterstützt werden. Touch-Icons werden beispielsweise nur auf mobilen Geräten, allerdings nicht auf einem Desktop PC benötigt. Dementsprechend kann die Partial Datei für den Desktop PC entfallen. Ist in einem gerätespezifischen Template Ordner keine Datei vorhanden, wird auch kein

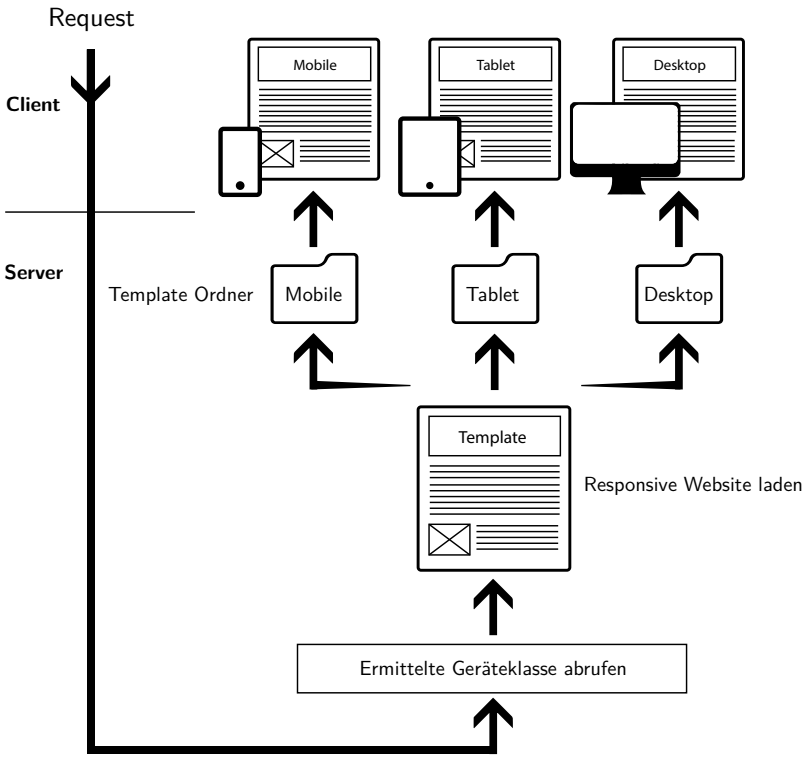


Abbildung 4.4: Auf Basis der ermittelten Geräteklasse werden die Partials in der responsiven Website ausgetauscht.

Inhalt eingebunden, was zu einer Verbesserung der Performance durch die Reduzierung der Website-Größe führt. Um doppelte Inhalte zu vermeiden, können Partials auch untereinander verlinkt werden. So werden die soeben genannten Touch-Icons, sowohl auf einem Smartphone als auch auf einem Tablet benötigt. Um eine doppelte Implementierung zu vermeiden, kann das Tablet Partial auf die Partial Datei im Mobile Ordner verlinkt werden. Dementsprechend wird auch auf einem Tablet der Inhalt des mobilen Partials eingebunden. Da die ermittelte Geräteklasse entsprechend gespeichert wird, besteht auch die Möglichkeit die angezeigte gerätespezifische *View* manuell zu ändern. So kann beispielsweise auf einem mobilen Gerät die Website mit den Desktop Partials angezeigt werden.

Durch die serverseitige Ermittlung der Gerätedaten ist die Nutzung einer Template-Engine auch mit Responsive Webdesign möglich. Dabei können einzelne Websitekomponenten für die entsprechende Geräteklasse optimiert und ausgetauscht werden. Dies erhöht die Flexibilität einer Website und ermöglicht erhebliche Performance-Optimierungen.

Kapitel 5

Umsetzung

Im Zuge dieser Arbeit wurde ein Framework-Prototyp entwickelt, der Responsive Webdesign um eine serverseitige Komponente erweitert. Dabei liegt der Fokus, wie in Kapitel 4 beschrieben, auf einer auflösungsoptimierten Auslieferung von Bildern, einem gerätespezifischen Austausch von Seitenbestandteilen und einer damit verbundenen Verbesserung der Performance. Wie das System aufgebaut ist und im Detail funktioniert, wird in dem folgenden Kapitel erläutert.

5.1 Architektur

Die Umsetzung des Framework-Prototyps ist in verschiedene Bereiche und Komponenten unterteilt (siehe Abb. 5.1). Dabei bilden die verwendeten Bibliotheken, die für die Funktionalität des Frameworks essentiell sind, die unterste Ebene. Diese umfasst die verschiedenen serverseitigen *Device Description Repositories*, JavaScript Bibliotheken für die clientseitige Geräteerkennung sowie eine Funktion für die Optimierung von Bilddateien.

Darüber befindet sich eine zusätzliche Abstraktionsschicht. Diese umfasst die Implementierung der Optimierungsfunktion für eingebundene Bilddateien sowie Adapter-Klassen für die serverseitigen Geräteerkennungsbibliotheken. In diesen Wrapper-Klassen werden die Funktionsweisen der verschiedenen *Repositories* vereinheitlicht und die benötigten Gerätedaten ermittelt. Parallel dazu erfolgt die Ermittlung der clientseitigen Gerätedaten über die eingebundene JavaScript Bibliothek. Die am Client ermittelten Informationen werden in ein Cookie gespeichert und mit den Daten der serverseitigen Geräteerkennung verglichen bzw. kombiniert.

Die Funktionalität des Frameworks wird über eine *Autoload* Datei eingebunden. Diese integriert die serverseitige Komponente in eine responsive Website (siehe auch Abschnitt 5.3). Eine externe Konfigurationsdatei ermöglicht dabei die Auswahl und Konfiguration des gewünschten *Device Description Repositories*, sowie weitere Einstellungen, wie die Konfiguration der in-

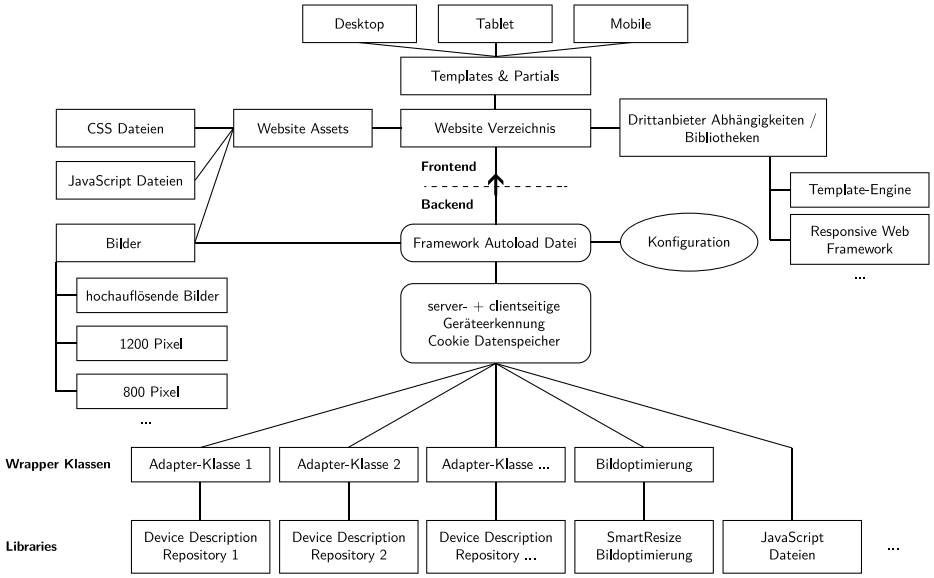


Abbildung 5.1: Die grundlegenden Komponenten und deren Zusammenspiel innerhalb des Framework-Prototyps.

tegrierten Bildoptimierungsfunktion. Hierfür werden die Layout-*Breakpoints* des individuellen Designs angegeben. Basierend auf diesen Pixelwerten wird eine entsprechende Ordnerstruktur im Assets Verzeichnis der Website angelegt. Je nach Auflösung eines Geräts werden die hochauflösenden Bilder in weiterer Folge automatisch reduziert und in den jeweiligen Ordnern gespeichert. Das Website Assets Verzeichnis dient zusätzlich zur Ablage von JavaScript und CSS Dateien.

Die HTML-Dateien einer responsiven Website werden in einem eigenen Website Verzeichnis gespeichert. Dieses bietet Zugriff auf benötigte Drittanbieter Bibliotheken, wie der verwendeten Template-Engine oder beispielsweise einem optionalen Responsive Web Framework. Die Bibliotheken werden in einem entsprechenden Verzeichnis gespeichert und über eine integrierte Abhängigkeitsverwaltung installiert bzw. verwaltet. Dies ermöglicht den einfachen Austausch oder die Aktualisierung dieser Komponenten. Erstellte Website Partials werden in gerätespezifischen Ordnern abgelegt. Dementsprechend können sie im Anschluss, je nach ermittelter Geräteklasse, ausgetauscht und in ein übergeordnetes responsives Layout integriert werden.

Der Aufbau und die Struktur der einzelnen Verzeichnisse des Frameworks werden in den folgenden Abschnitten detailliert beschrieben.

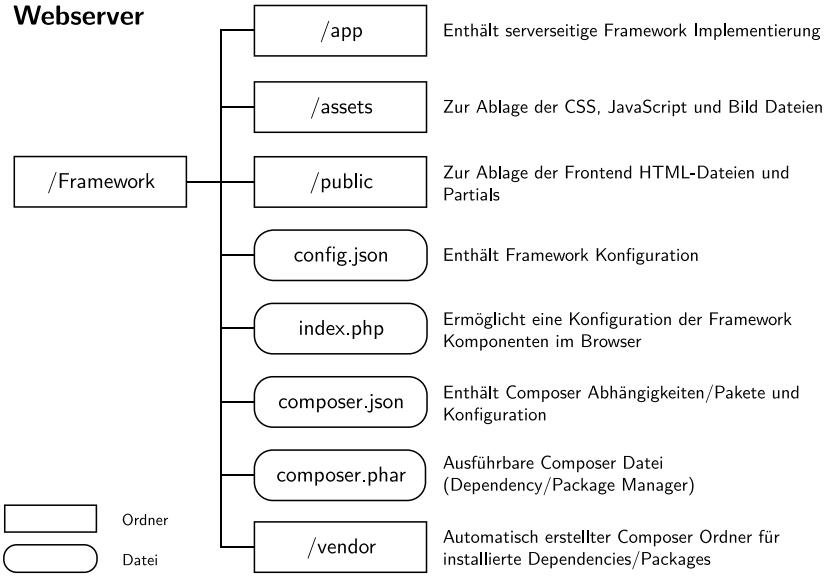


Abbildung 5.2: Erste Ebene der Ordnerstruktur des Framework-Prototyps auf einem Webserver.

5.1.1 Verzeichnisstruktur

Das System besteht aus verschiedenen Komponenten die in einer definierten Struktur auf einem Webserver abgelegt werden. Dabei ist der serverseitige Part des Frameworks in der Programmiersprache PHP realisiert. PHP ist eine flexible Skriptsprache die zur Erstellung von dynamischen Webanwendungen verwendet wird und zeichnet sich durch die Verfügbarkeit zahlreicher Bibliotheken aus [34]. So stehen auch die verwendeten *Device Description Repositories* in dieser Sprache zur Verfügung, was eine optimale Integration in das System ermöglicht. Zudem kann PHP auf fast allen gängigen Webservern ausgeführt werden und erhöht somit die Flexibilität und die Einsatzmöglichkeiten des Framework-Prototyps. Eine Voraussetzung für die Nutzung des Systems auf einem Webserver ist somit die Unterstützung der PHP Version 5.3.2+.

Abbildung 5.2 zeigt die erste Ebene der Ordnerstruktur und den Grundaufbau des Framework-Prototyps. Eine zentrale Konfiguration ermöglicht die Auswahl der lizenzgebundenen Bestandteile des Frameworks. Diese kann direkt über die Datei *index.php*, bei Aufruf des Stammverzeichnisses im Browser, als visuelles Formular ausgeführt werden. Die Konfigurationseinstellungen werden anschließend in dem kompakten Datenaustauschformat JSON (JavaScript Object Notation) [35], in der Datei *config.json*, gespeichert und können jederzeit geändert werden. Das *app* Verzeichnis enthält

den serverseitigen Framework Code. Dieser muss nicht modifiziert oder erweitert werden und arbeitet autonom. Der *assets* Ordner dient einem Nutzer zur Ablage der CSS, JavaScript und Bild Dateien. Das *public* Verzeichnis beinhaltet die responsive Website selbst. Hier kann ein Nutzer die HTML Frontend Dateien und die erstellten Partials der implementierten Template-Engine ablegen. Auf die Inhalte der einzelnen Verzeichnisse wird in den folgenden Abschnitten näher eingegangen.

Composer

Um die Verwaltung von benötigten Drittanbieter-Bibliotheken zu erleichtern integriert das Framework die Abhängigkeitsverwaltung Composer. Composer ermöglicht die einfache Verwaltung von Paketen und Abhängigkeiten in PHP Software. Dabei können die benötigten Pakete in der Datei *composer.json* festgelegt werden. Wird die Composer Datei *composer.phar* im Anschluss ausgeführt, werden die Pakete bzw. Abhängigkeiten automatisch heruntergeladen bzw. aktualisiert und in dem Verzeichnis *vendor* abgelegt. Neben dem Download der Bibliotheken integriert Composer zusätzlich eine *Autoload* Datei. Diese ermöglicht eine einfache Einbindung und Nutzung der installierten Pakete. Eine ausführliche Dokumentation über die Funktionsweise der Abhängigkeitsverwaltung findet sich auf der Website des Tools [42].

5.1.2 app Verzeichnis

Das *app* Verzeichnis enthält die serverseitigen Komponenten des Framework-Prototyps (siehe Abb. 5.3). Es ist in die beiden Unterverzeichnisse *config* und *libs* unterteilt. Der *libs* Ordner enthält dabei die Bibliotheken, die für die Funktionalität des Frameworks benötigt werden. Diese umfassen *Repositories* für eine serverseitige und clientseitige Geräteerkennung sowie eine PHP Funktion zur Bildoptimierung. Die Abbildung zeigt nur einen Ausschnitt der implementierten Bibliotheken. Eine genaue Auflistung und nähere Beschreibung der verwendeten Lösungen findet sich in Abschnitt 5.2. Die Bibliotheken im *libs* Verzeichnis sind nicht modifiziert und können daher problemlos ausgetauscht und auf eine neue Version aktualisiert werden.

Das *config* Verzeichnis enthält die eigentliche Funktionalität des Frameworks. Die Datei *autoload.php* lädt dabei den Framework Code und integriert die serverseitige Komponente in eine responsive Website (siehe auch Abschnitt 5.3). Die gewünschte Konfiguration wird durch die Datei *configuration.php* aus der externen *config.json* Datei extrahiert. Um eine einheitliche Nutzung der *Device Description Repositories* zu ermöglichen, greift das Framework über die, bereits in Abschnitt 4.2.1 erwähnten, Adapter-Klassen auf die entsprechenden Bibliotheken zu. Wie in Abbildung 5.3 zu sehen, existiert für jede eingebundene Bibliothek ein entsprechendes Konfigurationsver-

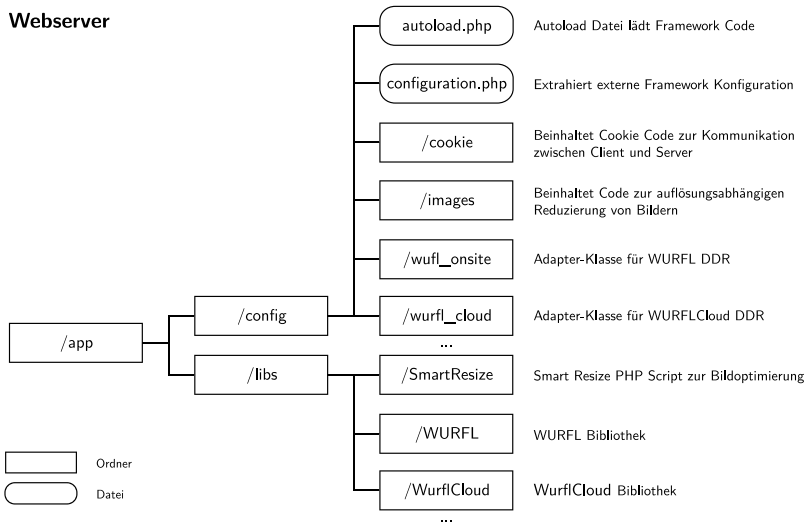


Abbildung 5.3: Das *app* Verzeichnis unterteilt sich in die beiden Ordner *config* und *libs* und enthält die serverseitige Funktionalität des Systems.

zeichnis. In diesen Ordnern befinden sich die jeweiligen Adapter-Klassen und eventuell benötigte Konfigurationsdateien. Die definierten Gerätedaten werden in der Adapter-Klasse ermittelt, in einem *Device Description Repository* Objekt gespeichert und im Anschluss über ein Cookie mit den ermittelten Gerätedaten des Clients verglichen. Der entsprechende Code für die Ermittlung und den Abgleich der Daten befindet sich im *cookie* Verzeichnis. Auch die auflösungsabhängige Optimierung der eingebundenen Bilddateien greift auf die ermittelten und kombinierten Gerätedaten zu. Der Code für eine anschließende Optimierung und Reduzierung der eingebundenen Bilder befindet sich dabei im Ordner *images*. Der genaue Ablauf, das Zusammenspiel und die Funktionsweisen der einzelnen Komponenten wird in Abschnitt 5.3 näher erläutert.

5.1.3 assets Verzeichnis

Das *assets* Verzeichnis dient zur Ablage der CSS, JavaScript und Bilddateien eines Projektes. Die entsprechende Ordnerstruktur für die Ablage der Dateien steht bereits zur Verfügung (siehe Abb. 5.4). Wird das Framework zum ersten Mal verwendet, ist im *images* Verzeichnis nur der Unterordner *source* vorhanden. In diesem Ordner kann ein Nutzer die Bilddateien der erstellten Website in der höchsten verfügbaren Auflösung speichern. Die auflösungsabhängige Ordnerstruktur wird im Anschluss, basierend auf der individuellen Konfiguration, automatisch erstellt. Wie in Abschnitt 4.2.4 beschrieben, werden die hochauflösenden Bilder aus dem *source* Verzeichnis

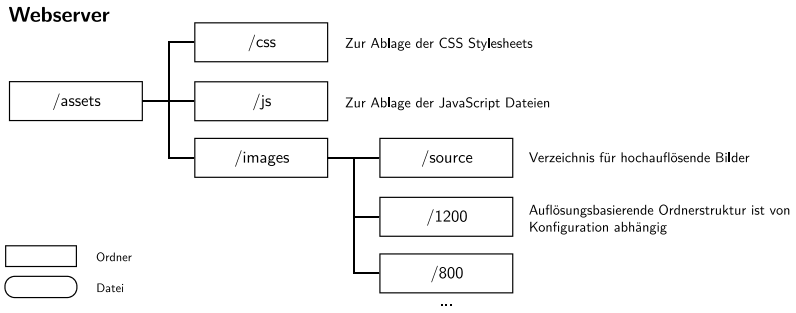


Abbildung 5.4: Im *assets* Verzeichnis kann ein Nutzer die CSS, JavaScript und Bild Dateien der erstellten Website ablegen.

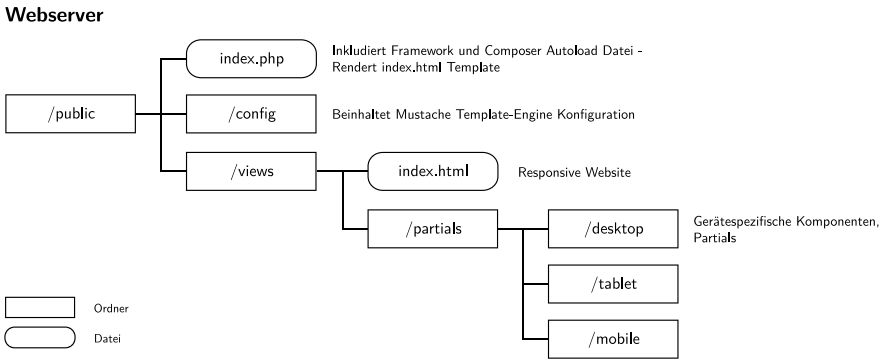


Abbildung 5.5: Das *public* Verzeichnis ermöglicht die Ablage der HTML Frontend Dateien und der erstellten Partials.

anschließend, je nach Auflösung des Geräts, automatisch reduziert und in den erstellten Ordnern abgelegt.

5.1.4 public Verzeichnis

Das *public* Verzeichnis dient zur Ablage der Frontend Dateien einer Website. Es ist in ein *config* und ein *views* Verzeichnis untergliedert (siehe Abb. 5.5). Im *config* Ordner befindet sich die Konfiguration der verwendeten Template-Engine. Dabei wird die *Autoload* Datei des Frameworks eingebunden und die einzelnen Framework-Funktionen an entsprechende Tags der Template-Engine gekoppelt. Dies ermöglicht in weiterer Folge eine einfache Verwendung der Framework-Funktionalität ohne serverseitigen Programmieraufwand. Das *views* Verzeichnis enthält die HTML Templates und Partials der Website. Dabei werden die Partials in gerätespezifischen Unterordnern abgelegt und anschließend, je nach Geräteklasse, automatisch ausgetauscht.

Tabelle 5.1: Eine Übersicht über die implementierten *Device Description Repositories*.

Bezeichnung	*Version*	*Kostenlose Version verfügbar*
WURFL	Lokal	für private Nutzung
WURFL Cloud	Cloud	mit Einschränkungen
DeviceAtlas Cloud	Cloud	14 Tage Testversion
Detector Beta	Lokal	ohne Einschränkungen
51Degrees	Lokal	mit Einschränkungen

5.2 Bibliotheken Auswahl

Im folgenden Abschnitt werden die Bibliotheken und Funktionen vorgestellt, die bei der Implementierung des Framework-Prototyps verwendet wurden. Die Auswahl wurde basierend auf den in Kapitel 4 beschriebenen Anforderungen getroffen.

5.2.1 Serverseitige Geräteerkennungsbibliotheken

Wie bereits in Abschnitt 4.2.1 erwähnt, sind die gängigen *Device Description Repositories* an die verschiedensten Lizenzbestimmungen gebunden, die ihre Nutzung entsprechend einschränken. Um die Wiederverwendbarkeit und die Flexibilität des Frameworks zu erhöhen, wurden verschiedene bekannte und verbreitete Geräteerkennungsbibliotheken implementiert (siehe Tab. 5.1). Die Auswahl des *Repositories* ist dabei dem Nutzer überlassen und kann in der Konfiguration jederzeit geändert werden.

WURFL

Eines der bekanntesten *Device Description Repositories* ist WURFL, das von ScientiaMobile entwickelt wird [16, S. 115]. Es besteht aus zwei Komponenten, einer API für die Abfrage der Gerätedaten und einem Ressourcenfile das die entsprechenden Daten bereitstellt. Das *Repository* wurde bis zur Version 2.2 unter einer Open Source Lizenz veröffentlicht. Danach wurde das WURFL Ressourcenfile allerdings rechtlich von ScientiaMobile geschützt und erlaubt seitdem keine kostenlose kommerzielle Nutzung mehr. Folglich steht die kostenlose Datenabfrage nur für private nicht kommerzielle Webdienste zur Verfügung. Soll das System für eine kommerzielle Website genutzt werden, muss eine entsprechende Lizenz von ScientiaMobile erworben werden [9, S. 32]. Wie bereits erwähnt, basiert WURFL auf einem Ressourcenfile das lokal am Webserver platziert wird. Die API greift anschließend auf die Gerätedaten zu und ermöglicht eine entsprechende Klassifizierung. Weitere Informationen über die Funktionalität des Systems und die genauen

Lizenzbestimmungen finden sich auf der Website des *Repositories* [36].

WURFL Cloud

ScientiaMobile stellt das WURFL *Device Description Repository* auch in einer Cloud Version zur Verfügung. Für die Cloud Lösung existieren verschiedene Lizenzversionen, die auf der ScientiaMobile Website detailliert beschrieben werden [37]. Dabei ist jede Version sowohl privat als auch kommerziell nutzbar. Die Lizenzgebühren werden monatlich verrechnet, was eine kostengünstigere Alternative zur lokalen WURFL Lösung darstellt. Die verschiedenen Lizenzbestimmungen unterscheiden sich hinsichtlich der Anzahl der monatlich möglichen Abfragen und der abfragbaren Gerätedaten. Zusätzlich existiert auch eine kostenlose Lizenz, die ebenfalls eine kommerzielle Nutzung erlaubt. Dabei ist die Abfrage für die Klassifizierung auf fünf Gerätedaten und 5000 Abfragen per Monat begrenzt. Da es sich hierbei um eine Cloud Lösung handelt entfällt darüber hinaus der manuelle Wartungsaufwand, da die Gerätedaten am ScientiaMobile Webserver automatisch aktuell gehalten werden. Für die Nutzung muss allerdings eine Registrierung auf der Produkt-Website erfolgen. Anschließend können die gewünschten Gerätedaten ausgewählt und der benötigte Lizenzschlüssel gespeichert werden. Dieser wird für die lokale Nutzung der WURFL Cloud API benötigt [9, S. 39–40].

DeviceAtlas Cloud

Neben WURFL zählt auch DeviceAtlas zu den bekanntesten *Device Description Repositories* [9, S. 32]. Dabei steht die implementierte Cloud Version ebenfalls in verschiedenen Lizenzversionen zur Verfügung. Der Unterschied besteht in der Anzahl der monatlichen Abfragen und der verfügbaren Gerätedaten. Allerdings bietet DeviceAtlas, im Gegensatz zu WURFL Cloud, keine kostenlose Lizenz an. Es steht lediglich eine 14 Tage Testversion zur Verfügung. Für die Nutzung des *Repositories* muss sich ein Nutzer auf der DeviceAtlas Website registrieren und erhält im Anschluss Zugang zu der API und dem benötigten Lizenzschlüssel [38]. Da es sich um eine Cloud Lösung handelt, entfällt der manuelle Wartungsaufwand und die verfügbaren Gerätedaten werden am Webserver täglich aktualisiert. Zusätzlich wirbt DeviceAtlas mit einer schnellen Zugriffszeit und einer dementsprechend guten Performance.

Detector Beta

Detector zählt zu den weniger bekannten *Device Description Repositories* und wurde von der Privatperson Dave Olsen entwickelt. Im Gegensatz zu WURFL oder DeviceAtlas arbeitet Detector nach einem völlig anderen Prinzip. Es nutzt für die Abfrage der Gerätedaten kein zentrales Verzeichnis,

sondern basiert, zusätzlich zu einer serverseitigen Überprüfung, auf der JavaScript *Feature Detection Modernizr*[1]. Die Ergebnisse werden anschließend in einer Session Variable und nach einem Reload im lokalen Cache gespeichert. Wird eine Website im Browser aufgerufen, wird zuerst überprüft ob eine offene Session für den Nutzer existiert. Ist dies der Fall, werden die bereits ermittelten Daten aus der Session Variable übernommen. Existiert keine valide Session, wird der *User-Agent-String* des Geräts mit einer Liste der bereits ermittelten *User-Agents* verglichen. Wird eine Übereinstimmung gefunden, werden die entsprechenden Daten verwendet. Findet sich keine Übereinstimmung, wird der *User-Agent-String* des Geräts auf der Serverseite analysiert. Parallel dazu werden, um bestimmte Gerätedaten zu ermitteln, *Modernizr* Tests durchgeführt. Die Ergebnisse werden in einem Cookie gespeichert und nach einem Reload auf den Server übertragen [9, S. 41]. Das Gerätedaten Verzeichnis erstellt sich dabei selbst und muss somit nicht händisch gewartet bzw. aktualisiert werden. Folglich muss jedes Gerät bzw. jeder *User-Agent-String* nur einmal ermittelt und überprüft werden. Ruft ein ähnliches Gerät die Website auf, wird auf die bereits ermittelten Daten zurückgegriffen. Das *Repository* basiert auf einer toleranten Lizenz die eine kostenlose Verwendung und damit verbundene vielfältige Einsatzmöglichkeiten erlaubt. Da Detector auf einer JavaScript *Feature Detection* basiert, eignet sich die Lösung vor allem für die Erkennung von aktuellen Geräten. Eine detaillierte Beschreibung der Funktionsweise findet sich auf der Website des *Device Description Repositories* [66].

51Degrees

51Degrees basiert auf einem ähnlichen Konzept wie WURFL und DeviceAtlas [9, S. 41]. Das *Repository* nutzt ebenfalls ein Ressourcenfile, das lokal am Webserver platziert wird. Über eine entsprechende API kann im Anschluss auf die Gerätedaten zugegriffen und eine Klassifizierung durchgeführt werden. 51Degrees steht in verschiedenen Lizenzversionen zur Verfügung. Diese unterscheiden sich hinsichtlich der verfügbaren abfragbaren Gerätedaten und der Häufigkeit der zur Verfügung stehenden Aktualisierungen. Die vollständige API ist dabei kostenlos und kommerziell nutzbar. Zusätzlich steht ein limitiertes Ressourcenfile für eine kostenlose Ermittlung der Gerätedaten zur Verfügung. Die Anzahl der monatlichen Abfragen ist dabei nicht begrenzt, allerdings stehen in der kostenlosen Version nicht alle Gerätedaten zur Verfügung [39].

5.2.2 WURFL.js

Wie in Kapitel 4 beschrieben ist in den Framework-Prototyp auch eine clientseitige Geräteerkennung integriert. Die Ermittlung der aktuellen Breite

[1] Modernizr Website: http://modernizr.com/

des Browserfensters erfolgt dabei über JavaScript. Eine Klassifizierung der Geräteklasse ist jedoch mit reinem JavaScript nur schwer möglich und kann nur durch eine händische Analyse eines *User-Agent-Strings* durchgeführt werden. Aus diesem Grund wurde die JavaScript Bibliothek WURFL.js implementiert [40]. WURFL.js leitet den HTTP Request eines anfragenden Geräts an den WURFL.io Webserver weiter, wo dieser im Anschluss analysiert und das Ergebnis in einem JavaScript Objekt zurückgesendet wird. Zusätzlich werden auch clientseitige Überprüfungen mit JavaScript durchgeführt. Diese Vorgehensweise ermöglicht eine genaue Klassifizierung des anfragenden Geräts. Beispielsweise ist eine Unterscheidung von iPhone 5 und iPhone 5s möglich [46]. Über das JavaScript Objekt kann anschließend die Bezeichnung, der Formfaktor und ob es sich um ein mobiles Gerät (Mobile, Tablet) handelt, abgefragt werden. WURFL.js kann für die Ermittlung der Gerätedaten kostenlos verwendet werden, so lange die entsprechende Website öffentlich und ohne Gebühren zugänglich ist.

5.2.3 SmartResize

Um die in Abschnitt 4.2.4 beschriebene, automatische auflösungsabhängige Optimierung der eingebundenen Bilddateien zu ermöglichen, müssen die hochauflösenden Bildversionen auf dem Webserver entsprechend reduziert werden. Die Programmiersprache PHP bietet, unter der Verwendung der *GD Library*, bereits eine Reihe von Möglichkeiten und Funktionen um eine serverseitige Bildmanipulation durchzuführen. Das PHP Script SmartResizeImage repräsentiert eine bereits implementierte Lösung zur Reduzierung von Bilddateien und fast die benötigte Funktionalität in einer Funktion zusammen [47]. SmartResizeImage unterstützt die gängigen Bildformate JPEG, PNG und GIF. Als Parameter werden der Funktion unter anderem der Pfad zu der hochauflösenden Bilddatei, die benötigte Auflösung und der Pfad für die zu erstellende reduzierte Bildversion übergeben. Bei einer Reduzierung werden sowohl die korrekten Proportionen eines Bildes, als auch eine eventuell vorhandene Transparenz bei PNG und GIF Dateien, berücksichtigt.

5.2.4 Mustache Template-Engine

Ein Austausch und eine Optimierung von einzelnen Seitenbestandteilen, basierend auf definierten Geräteklassen, ist nur durch die Einbindung einer entsprechenden Template-Engine möglich. Diese soll eine einfache Nutzung ermöglichen und keinen Programmieraufwand in den Templates und Partials erfordern. Mustache ist eine sogenannte *Logic-Less* Template-Engine [41]. Das bedeutet, dass Mustache keine Business-Logik, wie *if*, *else* oder *for* Schleifen in den Templates unterstützt und somit die Programmierlogik von der Präsentation trennt. Stattdessen basiert das System auf Tags. Diese

sind vielfältig einsetzbar und erlauben auch die Integration von Lambda-Funktionen. Folglich können die Funktionen des Framework-Prototyps an entsprechende Mustache Tags gekoppelt werden, was eine einfache Nutzung der implementierten Funktionalitäten erlaubt. Auch die Verwendung von Partials ist mit Mustache möglich. Dabei wird ein Verzeichnis für die Partial Dateien spezifiziert. Diese können im Anschluss über ein Tag und den Namen der Partial Datei in ein übergeordnetes Template integriert werden. Mustache ermöglicht die Erstellung der Templates und Partials in normalen HTML-Dateien, was die Nutzung der Template-Engine zusätzlich vereinfacht. Die genutzten Tags und ihre Funktion im Framework werden in Abschnitt 5.4 beschrieben.

5.2.5 Minify

Um die Performance einer Website zu steigern, werden gewöhnlich auch die eingebundenen CSS und JavaScript Dateien komprimiert. Dies kann durch verschiedene Tools erfolgen und ist meist mit einem händischen Aufwand verbunden. Um diesen Vorgang zu vereinfachen und zu automatisieren ist das Composer Paket Minify in den Framework-Prototyp integriert. Das Script ermöglicht eine einfache Komprimierung der, im *assets* Verzeichnis abgelegten, CSS und JavaScript Dateien. Dabei werden die entsprechenden Dateien komprimiert und anschließend im selben Verzeichnis abgelegt. Eine Komprimierung der Website Ressourcen kann allerdings auf verschiedene Arten erfolgen. Die Vorgehensweise ist dabei gewöhnlich von den Vorlieben und dem Workflow eines Entwicklers abhängig. Dementsprechend ist die Nutzung dieses zusätzlichen Paketes optional und stellt keine Kernkomponente des Frameworks dar. Es soll lediglich die Erweiterungsmöglichkeiten des Systems, anhand eines praktischen Beispieles aufzeigen. Weitere Details über die Funktionalität des Paketes finden sich auf der Minify Website [64].

5.3 Framework Implementierung

Im folgenden Abschnitt wird auf die konkrete Implementierung der einzelnen Komponenten eingegangen. Dabei werden die technischen Herausforderungen aufgezeigt und essentielle Codeabschnitte detailliert beschrieben. Die Umsetzung des Framework-Prototyps erfolgte in den Programmiersprachen PHP und JavaScript.

5.3.1 Ablauf der Geräteerkennung

Wie in Kapitel 4 beschrieben, basiert das Framework auf einer serverseitigen Geräteerkennung. Diese ermöglicht eine entsprechende Geräteklassifizierung und in weiterer Folge serverseitige Optimierungsmaßnahmen, wie eine Reduzierung der eingebundenen Bilder und einen Austausch von Seitenbestand-

teilen, basierend auf definierten Geräteklassen. Zu diesem Zweck wurden die in Abschnitt 5.2.1 beschriebenen *Device Description Repositories* implementiert. Da sich die Bibliotheken jedoch hinsichtlich ihrer Funktionsweise und der abfragbaren Gerätedaten unterscheiden, wird eine zusätzliche Abstraktionsschicht benötigt. Diese wird über Adapter-Klassen realisiert, in denen die Konfiguration des individuellen *Repositories* und die Abfrage der benötigten Gerätedaten erfolgt. Der Aufbau der Adapter-Klassen orientiert sich dabei an der in [9, S. 35–38] beschriebenen Klassen-Struktur und wird in weiterer Folge auszugsweise anhand des WURFL Cloud *Repositories* beschrieben.

In einer Adapter-Klasse wird über die Methode `fillConfigArray()` die zentrale Nutzer-Konfiguration, wie beispielsweise ein optionaler API Lizenzschlüssel oder die gewünschte Fallback-Geräteklasse geladen und auf die benötigten Verzeichnisse bzw. Dateien der Bibliothek zugegriffen. Die Konfigurationsdaten werden anschließend in einer Klassen-Variable gespeichert, was einen globalen Zugriff auf die Informationen innerhalb der Klasse erlaubt. In der Methode `createWURFLObject()` wird ein Objekt der Bibliothek erzeugt, das im Anschluss einen Zugriff auf die Gerätedatenbank ermöglicht. Die Abfrage der gewünschten Gerätedaten wird in der Methode `getDeviceClass()` durchgeführt (siehe Prog. 5.1). Dabei erfolgt am Beginn die Konfiguration des WURFL Cloud Clients. Dieser wird anschließend ausgeführt und ermöglicht den Zugriff auf die verfügbaren Geräteeigenschaften. Bei der Ermittlung der Auflösung, bzw. Fensterbreite wird zuerst eine Standardbreite von 1440 Pixel definiert. Dieser Durchschnittswert wird für Desktop PCs verwendet, da WURFL in diesem Fall einen sehr niedrigen und nicht zeitgemäßen Wert von 600 Pixel liefert. Sollte die Ermittlung der Fensterbreite nicht möglich sein oder fehlschlagen, wird ebenfalls auf diesen Wert zurückgegriffen. Anschließend erfolgt die Ermittlung der Geräteklasse. WURFL Cloud stellt einem Nutzer hierbei bereits entsprechende Geräteeigenschaften für eine Klassifizierung zur Verfügung. Die Eigenschaft *is_wireless_device* triff beispielsweise auf alle mobilen Geräte zu. Sollte die Klassifizierung eines Geräts nicht möglich sein, wird die, in der Konfiguration gewählte, Fallback-Geräteklasse verwendet.

Parallel dazu wird eine clientseitige Geräteerkennung mit JavaScript durchgeführt. Ist JavaScript im Browser aktiviert wird zuerst überprüft ob das Speichern von Cookies möglich ist:

```
COOKIE.cookieEnabled = function () {
  var cookie = !!(navigator.cookieEnabled);
  if (typeof navigator.cookieEnabled == 'undefined' && !cookieEnabled) {
    document.cookie = 'testCookieSupport';
    cookie = (document.cookie.indexOf('testCookieSupport') != -1);
  }
  return cookie;
};
```

Ist die Option im Browser des Website-Besuchers deaktiviert, wird auf die

Programm 5.1: Der Code zeigt den Aufbau der Methode zur Ermittlung der Gerätedaten in der Adapter-Klasse des WURFL Cloud *Repositories*.

```
function getDeviceClass()
{
  // Setup the WURFLCloud-Client and do the device detection
  $this->wurfl_client =
    new WurflCloud_Client_Client($this->wurfl_config);
  $this->wurfl_client->detectDevice();

  // Set default width
  $width = 1440;

  // Detect screenwidth of device
  if($this->wurfl_client->getDeviceCapability('max_image_width') &&
  !$this->wurfl_client->getDeviceCapability('ux_full_desktop')) {
    $width = $this->wurfl_client
      ->getDeviceCapability('max_image_width');
  }

  // Use capabilities to detect device class
  if ($this->wurfl_client->getDeviceCapability('is_tablet')) {
    // Dispatch HTTP request to tablet view
    return array('device' => 'tablet', 'screen_width' => $width);

  } else {
    if ($this->wurfl_client->getDeviceCapability('is_wireless_device'))
    {
      // Dispatch HTTP request to mobile view
      return array('device' => 'mobile', 'screen_width' => $width);

    } else {
      if ($this->wurfl_client->getDeviceCapability('ux_full_desktop')) {
        // Dispatch HTTP request to desktop view
        return array('device' => 'desktop', 'screen_width' => $width);

      } else {
        // Dispatch HTTP request to fallback view
        if ($this->fallback == 3) {
          return array('device' => 'mobile', 'screen_width' => $width);
        } else if ($this->fallback == 2) {
          return array('device' => 'tablet', 'screen_width' => $width);
        } else {
          return array('device' => 'desktop', 'screen_width' => $width);
        }
      }
    }
  }
}
```

serverseitig ermittelten Gerätedaten zurückgegriffen. War das Speichern eines Test-Cookies erfolgreich, wird eine clientseitige Geräteerkennung durchgeführt. Dabei wird zuerst überprüft, ob bereits ein entsprechendes Cookie existiert, sprich das Gerät die Website schon einmal aufgerufen hat. Ist dies der Fall, wird die Information aus dem bestehenden Cookie verwendet und keine erneute Überprüfung durchgeführt. Existiert kein Cookie wird eine Geräteerkennung auf Basis der JavaScript-Bibliothek WURFL.js ausgeführt:

```
// Return detected device class (from User Agent)
COOKIE.getDeviceClass = function () {
  if (WURFL.is_mobile && WURFL.form_factor == 'Tablet') {
    return 'tablet';
  } else if (WURFL.is_mobile) {
    return 'mobile';
  } else if (!WURFL.is_mobile && WURFL.form_factor == 'Desktop') {
    return 'desktop';
  } else {
    return 'unknown';
  }
};
```

Dabei wird über das WURFL Objekt auf die verfügbaren Geräteeigenschaften zugegriffen und eine entsprechende Klassifizierung durchgeführt. Auch ein manuelles setzen der Geräteklasse ist auf diese Weise möglich:

```
// Switch site version to assigned view
switchView = function (device) {
  if (COOKIE.cookieEnabled()) {
    if (device == 'desktop' || device == 'tablet' || device == 'mobile'
    || device == 'detected') {
      COOKIE.storeDeviceInformation(device);
      window.location.reload(true);
    } else {
    console.log('Error: Wrong device class assigned for switchView() -
    Please use detected, desktop, tablet, or mobile')
    }
  }
};
```

Die Funktion kann mit einer gewünschten Geräteklasse aufgerufen werden. Diese wird anschließend an die Funktion `storeDeviceInformation()` übergeben. Damit die Websitekomponenten auch richtig dargestellt werden, erfolgt anschließend ein Reload der Seite direkt vom Server und nicht vom Cache des Browsers. Die Funktion `storeDeviceInformation()` ist für die Verwaltung und das Speichern der Geräteinformationen in einem Cookie verantwortlich. Wird die Funktion, wie oben angeführt, mit einer korrekten Geräteklasse aufgerufen, wird diese gespeichert und keine erneute Ermittlung über die JavaScript-Bibliothek WURFL.js durchgeführt. Erfolgt ein Aufruf der Funktion jedoch ohne einer Geräteklasse oder dem Parameter *detected*, wird wiederum eine Geräteerkennung über die Funktion `getDeviceClass()` durchgeführt. Zusätzlich wird die aktuelle Breite des Browserfensters über

die JavaScript Funktion `window.innerWidth()` ermittelt und die beiden Informationen anschließend in einem *DeviceInformation* Cookie gespeichert.

Darüber hinaus besteht die Möglichkeit, einen Wechsel der Geräteklasse auch auf einem Desktop PC zu simulieren. Wird diese Option aktiviert, werden an zwei *Breakpoints* (Tablet und Mobile) die eingebundenen Bilder und die gerätespezifischen Inhalte ausgetauscht. Dabei wird die Auflösung des Browserfensters mit einem `window.onresize()`-Event überwacht. Wird die Breite des Fensters von einem Website-Besucher geändert und über- oder unterschreitet die neue Auflösung einen der gewählten *Breakpoints*, wird die neue Geräteklasse an die Funktion `switchView()` übergeben und die Website mit den neuen Inhalten und optimierten Bildern geladen. Um eine Reload-Schleife zu vermeiden, wird die alte Breite des Browserfensters gespeichert und bei einer Änderung der Fenstergröße mit der neuen Auflösung verglichen. Übersteigt die Differenz einen definierten Wert, wird die Seite entsprechend aktualisiert. Damit sich die Scrollposition eines Besuchers dabei nicht verändert, wird diese zusätzlich gespeichert und nach dem Reload neu gesetzt. Diese Vorgehensweise ermöglicht einen Test der Website für alle definierten Geräteklassen und jede Auflösung auf einem Desktop PC.

Anschließend werden die auf der Clientseite ermittelten Informationen auf der Serverseite extrahiert und mit den Daten der serverseitigen Geräteerkennung kombiniert:

```
// Look for DeviceInformation Cookie stored on pageload (client side)
if (isset($_COOKIE['DeviceInformation'])) {
  $cookie = $_COOKIE['DeviceInformation'];

  // If Cookie exists - explode information
  if ($cookie) {
    $values = explode('|', $cookie);
    foreach ($values as $value) {
      $capability = explode('.', $value);
      $clientDeviceInformation[$capability[0]] = $capability[1];
      $clientDeviceInformation[$capability[2]] = $capability[3];
    }
  }
  getDeviceInformation($serverDeviceInformation,
    $clientDeviceInformation, $device_detection_library);

} else {
  // Select Device Detection Library - perform server side Device Detection
  $serverDeviceInformation = selectDeviceDetectionLibrary(
    $device_detection_library);
  getDeviceInformation($serverDeviceInformation,
    $clientDeviceInformation, $device_detection_library);
}
```

Es wird zuerst überprüft, ob bereits ein *DeviceInformation* Cookie existiert. Ist dies der Fall, werden die gespeicherten Informationen extrahiert und in eine entsprechende Variable gespeichert. Wird eine Website das ers-

te Mal aufgerufen besteht noch kein Cookie, da dieses erst nach einem Reload auf der Serverseite zur Verfügung steht. In diesem Fall wird über die Funktion `selectDeviceDetectionLibrary()` das, in der Konfiguration ausgewählte, *Device Description Repository* ausgeführt und die Gerätedaten serverseitig ermittelt. Anschließend wird in beiden Fällen die Funktion `getDeviceInformation()` aufgerufen. In dieser Funktion werden die serverseitigen und clientseitigen Gerätedaten verglichen. Wird beispielsweise die Geräteklasse durch die WURFL.js JavaScript-Bibliothek nicht erkannt und es wird ein *unknown* zurückgeliefert, wird eine zusätzliche serverseitige Geräteerkennung und damit verbundene Ermittlung der Klassifikation versucht. Sind die ermittelten Daten auf der Clientseite fehlerhaft oder nicht vollständig, werden die benötigten Informationen von den Gerätedaten des Servers übernommen. Widersprechen sich die ermittelten Informationen von Server und Client, wird wie in Abschnitt 4.2.3 beschrieben, auf die aktuellen Daten des Clients vertraut. Die kombinierten Gerätedaten werden anschließend in eine globale Variable gespeichert und stehen somit den anderen Funktionen des Framework-Prototyps zur Verfügung.

5.3.2 Optimierung von Bildern

Die ermittelte und gespeicherte Auflösung wird im Anschluss für eine auflösungsabhängige Optimierung der eingebundenen Bilder verwendet. Je nach Auflösung eines Geräts werden die Bilddateien am Server automatisch reduziert und ausgetauscht. Die implementierte Lösung orientiert sich dabei an dem in [9, S. 72–80] beschriebenen Konzept. In der Konfiguration können die *Breakpoints* des individuellen responsiven Layouts angegeben werden. Zusätzlich steht im *assets* Verzeichnis bereits ein Ordner zur Ablage der Bilddateien bereit (siehe Abschnitt 5.1.3). Wird das Framework das erste Mal verwendet, existiert im *images* Verzeichnis nur ein *source* Ordner. Dieser dient zur Ablage der hochauflösenden Bilddateien. Diese Bilder werden in weiterer Folge automatisch reduziert und in einer, basierend auf der individuellen Konfiguration, erstellten Ordnerstruktur im *images* Verzeichnis gespeichert. Die Verzeichnisstruktur ist dabei folgendermaßen aufgebaut:

```
assets/images/source
- image.jpeg (1400x1050 Pixel)
assets/images/1200
- image.jpeg (1200x900 Pixel)
assets/images/800
- image.jpeg (800x600 Pixel)
```

Die Dateinamen für die reduzierten Bilder werden dabei von den hochauflösenden Grafiken übernommen. Somit bleibt der Name einer Bilddatei immer identisch. Nur der Pfad wird, je nach ermittelter Auflösung, entsprechend angepasst.

Die Funktion `getCurrentResolution()` ist für die Ermittlung der idealen Auflösung verantwortlich. Dabei vergleicht sie die gespeicherte Geräteauflösung mit den Werten aus der Konfiguration:

```
function getCurrentResolution()
{
  if (!is_null($this->current_resolution)) {
    return $this->current_resolution;
  }
  $resolutions = $this->resolutions;

  // Delete resolutions lower than screen width
  foreach ($resolutions as $k => $val) {
    if ($val < $this->screen_width) {
      unset($resolutions[$k]);
    }
  }

  if (empty($resolutions)) {
    return 'source';
  } else {
    return min($resolutions);
  }
}
```

In der Funktion wird zuerst überprüft ob die benötigte Auflösung bereits ermittelt wurde. Die Varibale `$current_resolution` entspricht dabei dem niedrigsten Wert in der Konfiguration, der höher oder gleich der ermittelten Geräteauflösung ist. Wurde die Auflösung bereits ermittelt, wird der entsprechende Wert zurückgegeben. Ist dies nicht der Fall, wird eine Kopie der Konfigurationswerte in der Variable `$resolutions` erstellt. Anschließend werden alle Werte, die kleiner als die ermittelte Auflösung des Geräts sind, aus dem `$resolutions` Array entfernt. Ist das Array nach diesem Vorgang leer, verfügt das Gerät über eine höhere Auflösung als die definierten *Breakpoints* der Konfiguration und es wird auf die hochauflösende Datei im *source* Ordner verwiesen. Andernfalls wird der niedrigste vorhandene Wert aus der Konfiguration zurückgegeben.

Die Methode `getImage()` nutzt die Funktion `getCurrentResolution()` um den Pfad zu einer Bilddatei in einer idealen Auflösung zu ermitteln. Existiert ein Bild oder ein entsprechendes Verzeichnis noch nicht, wird es von der Funktion automatisch erstellt. Der Funktion wird dabei der Name der Bilddatei übergeben. Auf Basis dieser Information wird zuerst überprüft, ob eine hochauflösende Bilddatei mit dieser Bezeichnung im *source* Ordner existiert:

```
$image_source = $this->source_dir . 'source/' . $filename;
$img_file_exists = file_exists($image_source);
// Source file doesn't exist
if (!$img_file_exists) {
  throw new Exception('source file does not exists: source/'.$filename);
}
```

Ist eine hochauflösende Datei vorhanden, wird über die angeführte Funktion `getCurrentResolution()` die benötigte Auflösung ermittelt:

```
// Source file exists
$current_resolution = $this->getCurrentResolution();
// Build a file path for the file
$dir_resolution_base = $this->source_dir . $current_resolution . "/";
$target_img_file = $dir_resolution_base . $filename;

// If file exists - create path relative to site root
if (file_exists($target_img_file)) {
  return '..' . str_replace(BASE_PATH, "", $target_img_file);
}
```

Anschließend wird auf Basis des zurückgelieferten Wertes ein Pfad erstellt und überprüft ob bereits eine reduzierte Bilddatei in dem entsprechenden Verzeichnis vorhanden ist. Ist dies der Fall, wird der Pfad zu der benötigten Datei zurückgegeben. Existiert noch keine reduzierte Datei und ist die Auflösung der hochauflösenden Bilddatei kleiner als die ermittelte Geräteauflösung, wird keine Überprüfung durchgeführt und sofort der Pfad zu der hochauflösenden Grafik retourniert. Ist die Auflösung der hochauflösenden Bilddatei jedoch größer als die ermittelte Auflösung des anfragenden Geräts, wird eine Reduzierung durchgeführt. Dabei wird zuerst überprüft, ob das entsprechende auflösungsabhängige Verzeichnis bereits existiert. Zusätzlich wird analysiert, ob der übergebene Dateiname eine Verzeichnisstruktur enthält. Besteht die benötigte Verzeichnisstruktur noch nicht, wird diese zuerst automatisch erstellt und anschließend eine Reduzierung der hochauflösenden Bilddatei durchgeführt:

```
require_once(LIBRARIES_DIR . "/SmartResize/smart_resize_image.function.
    php");
if (is_numeric($current_resolution)) {
  $res = smart_resize_image($image_source, $target_img_file,
    $current_resolution);
  return '..' . str_replace(BASE_PATH, "", $target_img_file);
}
```

Die Reduzierung wird über das in Abschnitt 5.2 beschriebene PHP Script SmartResizeImage realisiert. Der Funktion werden dabei der Pfad zu der hochauflösenden Bilddatei, der Pfad für die zu erstellende Datei und die benötigte Auflösung übergeben. Die reduzierte Datei wird im Anschluss automatisch erstellt und in dem entsprechenden Verzeichnis abgelegt. Daraufhin wird noch der Pfad zu der reduzierten Bilddatei zurückgegeben.

Die Funktion `getImage()` kann folglich für die Ermittlung des richtigen Bildpfades verwendet werden. Dabei muss der Funktion nur der korrekte Dateiname der hochauflösenden Bilddatei übergeben werden. Anschließend wird, je nach Geräteauflösung, der richtige Pfad zurückgeliefert. Ist eine Bilddatei zu groß, wird diese automatisch reduziert und in einem entsprechenden Verzeichnis abgelegt.

Wie in Abschnitt 4.2.4 beschrieben, kann in der Konfiguration auch eine Überprüfung der Netzwerkgeschwindigkeit aktiviert werden. Dies ermöglicht eine auflösungsunabhängige Auslieferung der eingebundenen Bilddateien. Das bedeutet, dass auch an ein mobiles Gerät mit einer hohen Auflösung niedriger aufgelöste Bilder gesendet werden, wenn dieses über eine langsame Netzwerkverbindung verfügt. Ist zu einem späteren Zeitpunkt wieder eine bessere Netzwerkverbindung verfügbar, werden wiederum die hochauflösenden Bilder geladen.

Dieses Konzept wurde durch die Network Information API realisiert[2]. Dabei wird die aktuell verfügbare Bandbreite überprüft und der Wert in einem Cookie gespeichert:

```
function BandwidthChange() {
  highBandwidth = (!connection.metered && connection.bandwidth > 2);
  // Store Bandwidth Information in Cookie
  COOKIE.setCookie('Bandwidth', highBandwidth ? "high" : "low");
}
```

`connection.bandwidth` liefert dabei die aktuelle Bandbreite in MB/s (Megabytes per second). Das Flag `connection.metered` wird von der API gesetzt, wenn die Bandbreite von einem Internet Service Provider begrenzt bzw. limitiert wird. Beispielsweise bei einer kommerziellen *Pay-per-use* Verbindung. Jedoch wird die Network Information API zum jetzigen Zeitpunkt nur von Firefox und Chrome unterstützt. Aus diesem Grund wurde eine zusätzliche Fallback-Lösung mit JavaScript implementiert. Diese wird ausgeführt wenn die Network Information API von einem Browser nicht unterstützt wird. Dabei wird eine kleine Bilddatei fünf mal hintereinander von einem Webserver heruntergeladen und die Zeit für den Download gemessen. Ausschnitt aus der `testLatency()` Funktion:

```
var tStart = new Date().getTime();
if (i < timesToTest - 1) {
  dummyImage.src = testImage + '?t=' + tStart;
  dummyImage.onload = function () {
    var tEnd = new Date().getTime();
    var tTimeTook = tEnd - tStart;
    arrTimes[i] = tTimeTook;
    // Recursive call
    testLatency(cb);
    i++;
  };
}
```

Die ermittelte Zeit wird anschließend mit einem festgelegten Grenzwert verglichen und das Ergebnis ebenfalls in einem Cookie gespeichert.

Der so ermittelte Wert (*high* oder *low*) wird auf der Serverseite aus dem Cookie extrahiert und verarbeitet. Ist das Gerät über eine gute Netzwerkverbindung verbunden, werden die Bilder, basierend auf der Auflösung des

[2]W3C Network Information API Website: http://w3c.github.io/netinfo/

Geräts, ausgeliefert. Verfügt es über eine schlechte Verbindung, wird die auflösungsabhängige Optimierung deaktiviert und es werden, um die Performance zu verbessern, niedriger aufgelöste Bilddateien eingebunden.

5.3.3 Autoload Datei

Damit der entwickelte Backend Framework Code genutzt werden kann, muss nur die zentrale *Autoload* Datei in eine Website eingebunden werden:

```
// Require Framework Autoloader
require '../app/config/autoload.php';
```

Diese spezifiziert die Ordnerstruktur des Frameworks in Form von nutzbaren Konstanten und lädt anschließend den Framework Code und die entsprechende Konfiguration:

```
define ("APPLICATION_DIR", str_replace('/config', '', __DIR__) .
    DIRECTORY_SEPARATOR);
define("BASE_PATH", str_replace('/app/', '', APPLICATION_DIR));
define ("CONFIGURATION_DIR", __DIR__ .DIRECTORY_SEPARATOR);
define ("LIBRARIES_DIR", APPLICATION_DIR . 'libs' .DIRECTORY_SEPARATOR);
define ("PUBLIC_DIR", BASE_PATH . '/public'. '/');
define ("ASSETS_DIR", BASE_PATH . '/assets' . '/');

...

// Initialize Cookie Storage
require_once 'cookie/cookie.php';

// Initialize Image Optimization
require_once CONFIGURATION_DIR . 'images/app.php';
$IMGObject = new Images();
$IMGObject->setResolutions($image_resolutions);
```

Zusätzlich stellt die *Autoload* Datei Wrapper-Methoden für die integrierten Framework-Funktionen zur Verfügung. Beispielsweise liefert folgende Funktion, abhängig von der Auflösung eines Geräts, den Pfad zu einer optimalen Bilddatei:

```
// Return path to image with optimal resolution
function img($filename)
{
  global $IMGObject;
  return $IMGObject->getImage($filename);
}
```

Die Datei ermöglicht somit eine einfache Einbindung und Nutzung der implementierten serverseitigen Framework-Funktionalität.

5.3.4 Gerätespezifische Inhalte

Um einen Austausch und eine Optimierung von einzelnen Seitenbestandteilen, basierend auf den definierten Geräteklassen, zu ermöglichen, wurde die,

in Abschnitt 5.2 beschriebene, Mustache Template-Engine integriert. Da es sich dabei um eine Bibliothek für die Realisierung des Website-Frontends handelt, ist das Paket über die Abhängigkeitsverwaltung Composer installiert (siehe auch Abschnitt 5.1.1). Dies ermöglicht in weiterer Folge eine einfache Aktualisierung der Bibliothek.

Damit die Template-Engine auf die Funktionen des Frameworks zugreifen kann, ist eine Konfiguration und Anbindung an die implementierten serverseitigen Komponenten erforderlich. Diese erfolgt in einer entsprechenden Konfigurationsdatei. Dabei werden die *Autoload* Dateien des Frameworks und der Abhängigkeitsverwaltung Composer eingebunden. Folglich kann in der Datei auf die Framework-Funktionalität und die Composer Pakete zugegriffen werden, was eine Registrierung der Mustache *Autoload* Datei ermöglicht. Diese lädt den benötigten Code der Template-Engine:

```
Mustache_Autoloader::register();

$options = array('extension' => '.html');

// Template and Partial - Filesystem Loader
$mustache = new Mustache_Engine(array(
  'loader' => new Mustache_Loader_FilesystemLoader(PUBLIC_DIR. '/views',
     $options),
  'partials_loader' => new Mustache_Loader_FilesystemLoader(PUBLIC_DIR.
    '/views/partials/' . $GLOBALS['comparedDeviceInformation']['Device
    Class'], $options),
));
```

Anschließend wird der Dateityp für die Template- und Partial Dateien festgelegt. Mustache erlaubt die Erstellung von Templates und Partials in gängigen HTML-Dateien. Der Mustache *Filesystem Loader* ermöglicht die Spezifikation der Template und Partial Verzeichnisse. Dabei beinhaltet der *views* Ordner die HTML Templates, also die übergeordneten responsiven Seiten einer Website. Die austauschbaren Partials werden in gerätespezifischen Unterordnern abgelegt (siehe auch Abschnitt 5.1.4). Der `partials_loader` nutzt für die Spezifikation des Unterordners die ermittelte Geräteklasse. Dies ermöglicht in weiterer Folge einen automatischen Austausch der gerätespezifischen Seitenbestandteile, basierend auf der gespeicherten Geräteklassifikation. Die erstellten HTML Templates und Partials werden dabei in folgender Verzeichnisstruktur abgelegt:

```
public/views
- index.html (Responsive Website)
public/views/partials/desktop
- navigation.html (Navigation Desktop Partial)
public/views/partials/tablet
- navigation.html (Navigation Tablet Partial)
public/views/partials/mobile
- navigation.html (Navigation Mobile Partial)
```

Dabei stellt `index.html` die übergeordnete responsive Website dar. Das Partial `navigation.html` wird anschließend in die Website integriert und, basierend auf der ermittelten Geräteklasse, ausgetauscht. Um doppelte Inhalte zu vermeiden, können Partial Dateien auch untereinander verlinkt werden. Ist die implementierte Navigation in der Tablet Datei beispielsweise identisch zu der Navigation der Mobile Datei, kann das Tablet Partial auf die Partial Datei im *mobile* Ordner verlinkt werden. Dementsprechend wird auch auf einem Tablet der Inhalt des mobilen Partials geladen und es entfällt eine doppelte Implementierung.

Da Mustache eine *Logic-Less* Template-Engine darstellt und somit keine Programmierlogik in den HTML-Dateien erlaubt, besteht die Möglichkeit bestimmte Informationen und Funktionen an die Templates und Partials zu übergeben:

```
// Data that is required in Templates and Partials
$data = array(
  'viewDesktop'      => 'javascript:switchView("desktop")',
  'viewTablet'       => 'javascript:switchView("tablet")',
  'viewMobile'       => 'javascript:switchView("mobile")',

  'img'              =>  function ($filename) {
    return img($filename);
  },
);
```

Die über ein `$data` Array übertragenen Informationen und integrierten Lambda-Funktionen können in den Templates und Partials über eigene Mustache Tags genutzt werden. So kann beispielsweise die `switchView()` JavaScript Funktion aufgerufen werden und eine manuelle Änderung der Geräteklasse ermöglichen. Über eine Lambda-Funktion kann auch ein Dateiname an die implementierte `getImage()` Funktion übergeben werden. Das entsprechende Tag wird im Browser dann durch den Pfad zu einer idealen Bilddatei ersetzt. Folglich sind zur Nutzung der serverseitigen Funktionen des Frameworks keine Programmierkenntnisse erforderlich, da diese an simple Mustache Tags gekoppelt sind. Eine detaillierte Beschreibung zur Nutzung und Funktion dieser Tags findet sich im folgenden Abschnitt 5.4.

5.4 Framework Verwendung

Wird eine Website mit dem Framework erstellt, muss dieses zuerst entpackt und auf einem Webserver platziert werden. Die individuelle Konfiguration kann anschließend im Webbrowser erfolgen. Dabei muss nur das Stammverzeichnis des Framework-Prototyps im Browser geöffnet werden (siehe Abb. 5.6). Die Framework-Konfiguration ermöglicht verschiedene Einstellungen, wie die Auswahl des gewünschten *Device Description Repositories*. Diese werden anschließend gespeichert und können in weiterer Folge jederzeit wieder geändert bzw. angepasst werden.

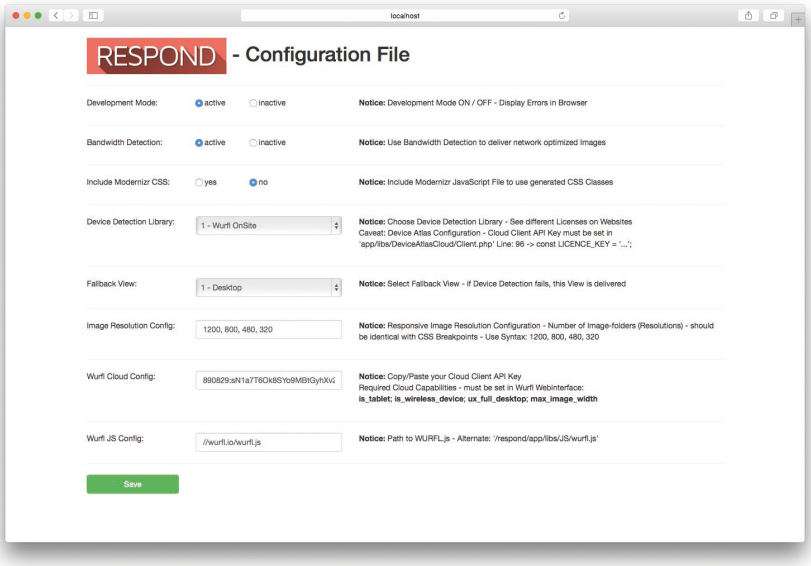


Abbildung 5.6: Die Konfiguration des Frameworks ist über ein User Interface im Browser möglich.

Die Erstellung der Website erfolgt im *public* Verzeichnis des Frameworks. Die erstellten HTML-Dateien für das Website-Frontend werden dabei im *views* Ordner gespeichert. Austauschbare Seitenbestandteile, wie beispielsweise die Navigation der Website, werden in den gerätespezifischen Unterordnern des *partials* Verzeichnisses abgelegt.

Die erstellten Partials können dabei über folgendes Mustache Tag in den HTML Code der übergeordneten responsiven Seite integriert werden:

```
<div>
  {{> Partial }}
</div>
```

Es muss in dem Mustache Tag nur der Name der Partial HTML-Datei angegeben werden. Die entsprechende Einbindung und der gerätespezifische Austausch der Dateien wird anschließend automatisch ausgeführt. Dies ermöglicht verschiedene Optimierungsmaßnahmen. Wird ein Video beispielsweise von einem mobilen Gerät nicht unterstützt, kann es durch einen alternativen Inhalt, wie einem statischen Bild, ersetzt werden. Wird ein Inhalt von einer Geräteklasse nicht benötigt, kann dieser auch zur Gänze entfernt werden. Dabei darf für die entsprechende Geräteklasse keine Partial Datei erstellt werden. Ist kein Partial für die benötigte Geräteklasse vorhanden, wird auch kein Inhalt eingebunden. Zusätzlich besteht auch die Möglichkeit, Partial Dateien untereinander zu verlinken. Soll beispielsweise ein Tablet denselben Inhalt wie ein mobiles Gerät erhalten, kann das Tablet Partial

auf die mobile Partial Datei verlinkt werden:

```
{{> ../mobile/Partial }}
```

Folglich wird auch auf einem Tablet der Inhalt des mobilen Partials geladen und es entfällt eine doppelte Implementierung.

Auch für die Nutzung der integrierten Bildoptimierungsfunktion steht ein Mustache Tag zur Verfügung:

```
<img src=" {{# img }}example.jpg{{/ img }} "/>
```

Mit diesem Tag wird der Dateiname einer hochauflösenden Grafik, die im *source* Ordner des *images* Verzeichnisses gespeichert wird, an die serverseitige Funktion übergeben. Wird die Website anschließend in einem Browser geöffnet, wird die Breite des Browserfensters ermittelt und das Tag durch den Pfad zu einer Bilddatei in der idealen Auflösung ersetzt. Ist diese noch nicht vorhanden, wird sie, wie bereits in Abschnitt 5.3.2 erwähnt, automatisch erstellt und in einem entsprechenden Ordner abgelegt. Folglich muss nur mehr der Name einer Bilddatei und nicht, wie bisher, der gesamte Pfad angegeben werden, was eine zusätzliche Erleichterung für einen Nutzer darstellt.

Schlägt die Erkennung einer Geräteklasse einmal fehl oder möchte ein Nutzer bewusst die Website mit den Inhalten einer anderen Geräteklasse anzeigen, ist dies ebenfalls möglich. Dabei wird über das folgende Tag beispielsweise die entsprechende JavaScript `switchView()` Funktion für die Geräteklasse Desktop aufgerufen:

```
<a href=" {{viewDesktop}} ">Desktop</a>
```

Diese Funktion steht in der Standardkonfiguration für alle Geräteklassen zur Verfügung. Sie soll weitere Kombinationsmöglichkeiten zwischen den Framework Funktionen und der Mustache Syntax anhand eines praktischen Beispiels aufzeigen.

Mustache Tags erlauben eine optimale Integration der Framework Komponenten in den Erstellungsprozess eines Website-Frontends. Durch die einfache Syntax ist kein Programmieraufwand erforderlich und die Nutzung des Systems kann schnell erlernt werden.

Kapitel 6

Evaluierung

Dieses Kapitel dient der Evaluierung des entwickelten Framework-Prototyps. Dabei erfolgt eine Reflexion der in Kapitel 4 gestellten Anforderungen und eine Gegenüberstellung mit den implementierten Komponenten des Frameworks. Anschließend wird eine Website, mit klassischem clientseitigen Responsive Webdesign, mit dem neuen *<picture>* HTML-Element und der Attribute *srcset* und *sizes* und mit dem implementierten Framework umgesetzt. Die erstellten Websites werden daraufhin einem Leistungsvergleich unterzogen. Dieser Vergleich soll die möglichen Performanceverbesserungen, die durch die Integration einer serverseitigen Komponenten möglich werden, aufzeigen.

6.1 Anforderungsreflexion

Der folgende Abschnitt beschreibt, wie und in welchem Ausmaß die Anforderungen an den erstellten Framework-Prototyp erfüllt wurden. Zusätzlich wird das eigene System mit anderen Ansätzen, die ebenfalls eine Verbesserung von Responsive Webdesign in den genannten Bereichen anstreben, verglichen und auf die Vor- und Nachteile des jeweiligen Konzeptes eingegangen.

Responsive Webdesign ermöglicht die Erstellung einer einzigen Website, die auf allen gängigen Geräten gleichermaßen funktioniert (siehe auch Abschnitt 2.3). Dabei passt sich das Layout einer Webpräsenz automatisch an die Displaygröße des jeweiligen Geräts an. Die Nutzung dieses Verfahrens ist allerdings auch mit Problemen und Einschränkungen verbunden. Im Gegensatz zu Adaptive Webdesign (siehe Abschnitt 2.2), nutzt Responsive Webdesign keine serverseitigen Optimierungsmaßnahmen. Folglich wird an alle Geräte dieselbe Datenmenge gesendet, was sich, vor allem auf mobilen Geräten, negativ auf die Performance auswirken kann. Zusätzlich können Bilder am Client meist nicht geräte- und auflösungsabhängig optimiert werden. Auch ein Austausch von einzelnen Seitenbestandteilen, basierend

auf verschiedenen Geräteklassen und eine damit verbundene Realisierung von gerätespezifischen Inhalten, ist mit clientseitigem Responsive Webdesign nicht möglich. Wie in Kapitel 4 beschrieben, sollten diese Einschränkungen und Nachteile durch die Implementierung eines Framework-Prototyps, der, auf Basis des RESS Konzeptes (siehe auch Abschnitt 3.3), Responsive Webdesign um eine serverseitige Komponente erweitert, adressiert werden. Die implementierten Optimierungsmaßnahmen sollen im Anschluss auch zu einer Verbesserung der Performance einer responsiven Website beitragen (siehe auch Abschnitt 6.2). Das Framework sollte dabei eine einfache Nutzung ermöglichen und keine serverseitigen Programmierkenntnisse erfordern. Auf die einzelnen Anforderungen wird in weiterer Folge detailliert eingegangen.

6.1.1 Optimierung von Bildern

Wie in Abschnitt 4.1.1 beschrieben, stellen Bilder seit jeher eines der größten Probleme des Responsive Webdesign Konzeptes dar. Diese werden auf mobilen Geräten meist nur skaliert und nicht komprimiert was, je nach Netzwerkverbindung, zu langen Ladezeiten und einer schlechten Performance führen kann. Die Einführung des neuen *<picture>* HTML-Elements und der Attribute *srcset* und *sizes* soll in Zukunft eine Realisierung von responsiven Bildern mit reinem HTML ermöglichen (siehe auch Abschnitt 3.2.1). Allerdings werden diese neuen Elemente nicht von jedem Browser unterstützt und es muss somit immer eine zusätzliche Fallback-Lösung eingebunden werden. Diese wird, wie in Abschnitt 3.2.2 beschrieben, meist durch ein JavaScript Polyfill Plugin realisiert. Dementsprechend ist eine problemlose Funktionalität immer an die Ausführbarkeit der Fallback-Lösung gebunden. Hat ein Website-Besucher jedoch JavaScript im Browser deaktiviert, kann diese nicht korrekt ausgeführt werden.

Als Anforderung zur Aufhebung dieser Einschränkung wurde eine Möglichkeit, die eine automatische Einbindung und Optimierung von flexiblen Bildern ohne JavaScript ermöglicht, definiert. Die Lösung sollte zusätzlich auf dem gängigen *<img>* HTML-Element bzw. *src* Attribut basieren und somit von allen Browsern unterstützt werden.

Wie eingangs erwähnt, erweitert das erstellte Framework Responsive Webdesign um eine serverseitige Komponente. Basierend auf dieser Tatsache konnte eine serverseitige Bildoptimierungsfunktion integriert werden, die der gestellten Anforderung gerecht wird (siehe auch Abschnitt 5.3.2). Dabei ermöglicht die implementierte Lösung eine Spezifikation der Layout-*Breakpoints* des individuellen responsiven Designs. Die hochauflösenden Bilddateien können anschließend in einem definierten Ordner am eigenen Webserver abgelegt werden. Über ein entsprechendes Mustache Tag wird der Dateiname einer Bilddatei im *src* Attribut eines *<img>* HTML-Elements spezifiziert (siehe Abschnitt 5.4). Basierend auf der ermittelten Auflösung eines Geräts und der individuellen Konfiguration, werden die eingebundenen Bil-

der im Anschluss wenn nötig reduziert, in einer generierten Ordnerstruktur am Webserver abgelegt und automatisch ausgetauscht.

Durch die Integration der Funktion in das Framework werden keine Drittanbieter-Tools, wie beispielsweise Responsive Webserver, benötigt und eine erstellte Website bleibt dementsprechend unabhängig und performant (siehe auch Abschnitt 3.2.3).

Die Funktionsweise der entwickelten Optimierungsfunktion orientiert sich an dem Konzept von *Adaptive Images* (siehe Abschnitt 3.2.4). Das Script nutzt allerdings eine *.htaccess* Datei um entsprechende Anfragen für Bilder an die Funktion weiterzuleiten und die Bilddateien automatisch zu reduzieren. *Adaptive Images* benötigt somit keinen manuellen Funktionsaufruf, kann aber nur auf das gängige *<img>* HTML-Element angewendet werden. Die reduzierten Bilder werden dabei allerdings nicht am Webserver gespeichert sondern nur im Cache abgelegt. Dementsprechend ändert sich auch die URL für die jeweilige Bilddatei nicht und es besteht keine Möglichkeit einzelne Grafiken für bestimmte Geräte händisch zu erstellen oder zusätzlich zu optimieren. Soll auf einem Gerät mit einer niedrigeren Auflösung ein anderes Bild oder ein anderer Bildausschnitt angezeigt werden, ist dies mit der im Framework integrierten Lösung möglich. Dabei kann eine Bildversion auch händisch erstellt bzw. optimiert und in dem jeweiligen Ordner gespeichert werden. Ist bereits eine Bilddatei mit dem benötigten Dateinamen vorhanden, wird diese verwendet und keine erneute Reduzierung der hochauflösenden Grafik durchgeführt.

Da die Funktion auf die serverseitige Implementierung des Framework-Prototyps zurückgreift, wird für die Funktionalität der Lösung kein JavaScript benötigt. Dementsprechend wird der Client entlastet und der nötige Rechenaufwand auf den Server verlagert. Zusätzlich kann die implementierte Optimierungsfunktion mit dem neuen *<picture>* HTML-Element und *srcset* Attribut kombiniert werden. Dabei entfällt, wie in Abschnitt 4.2.4 beschrieben, die manuelle Erstellung der einzelnen Bildversionen, da diese vom Server automatisch durchgeführt wird. Anschließend können die einzelnen Bildversionen in dem *<picture>* Element bzw. über das *srcset* Attribut spezifiziert werden. Als Fallback-Lösung kann, anstellen eines zusätzlichen JavaScript Plugins, die implementierte Optimierungsfunktion verwendet werden. Hierzu ermöglichen die beiden neuen Elemente die Angabe einer Fallback-Bilddatei. Sollten sie von einem Browser nicht unterstützt werden, wird diese Datei verwendet. Der Pfad zu der Fallback-Datei wird ebenfalls über das gängige *src* Attribut eines *<img>* HTML-Elements angegeben. Folglich kann über das entsprechende Mustache Tag wiederum auf die integrierte Bildoptimierungsfunktion zugegriffen werden und es werden an jedes Gerät, unabhängig davon ob es die neuen Elemente unterstützt, optimal aufgelöste Bilder ausgeliefert.

Grundsätzlich werden die eingebundenen Bilder auch auf einem mobilen Gerät auflösungsabhängig ausgetauscht. Das bedeutet, dass an ein mobiles

Gerät mit einem hochauflösenden Display auch hochauflösende Bilder ausgeliefert werden. Allerdings besteht die Möglichkeit in der Konfiguration eine Überprüfung der Netzwerkgeschwindigkeit zu aktivieren. Ist diese Option aktiviert und ist ein Gerät über eine langsame Netzwerkverbindung verbunden, werden niedriger aufgelöste Bilder gesendet, auch wenn das Gerät über ein hochauflösendes Display verfügt (siehe auch Abschnitt 5.3.2). Besteht zu einem späteren Zeitpunkt wieder eine bessere Verbindung, werden wiederum die hochauflösenden Bilder geladen.

6.1.2 Gerätespezifische Inhalte

Eine Realisierung von gerätespezifischen Inhalten ist, wie in Abschnitt 4.1.2 beschrieben, mit clientseitigem Responsive Webdesign nicht möglich. Allerdings ist ein Austausch und eine Optimierung von verschiedenen Seitenbestandteilen, basierend auf Geräteklassen, oftmals erforderlich. Beispielsweise muss ein Navigation-Pattern für die jeweilige Eingabemethode eines Geräts angepasst werden. Das implementierte Pattern muss auf einem Desktop PC eine Bedienung mit Maus und Tastatur und auf einem mobilen Gerät mit Touchdisplay, eine Bedienung per Finger ermöglichen. Andere Inhalte, wie eine Flash-Animation werden möglicherweise von einem mobilen Gerät nicht unterstützt oder werden im mobilen Kontext nicht benötigt und sollten durch andere Informationen ersetzt werden. Responsive Webdesign kennt allerdings die Klasse eines Geräts nicht. Folglich können Inhalte nur an bestimmten *Breakpoints* wechselweise über CSS *Media Queries* ausgeblendet werden. Die ausgeblendeten Inhalte werden jedoch im Hintergrund trotzdem geladen und können somit zu langen Ladezeiten und einer schlechten Performance führen. Der theoretische RESS Ansatz von Luke Wroblewski, erweitert Responsive Webdesign um eine serverseitige Komponente (siehe auch Abschnitt 3.3). Die serverseitige Logik soll im Anschluss für eine Optimierung von einzelnen Websiteelementen verwendet werden. Allerdings gibt es für die Umsetzung eines solchen Systems keine konkreten Vorgaben und die Implementierung ist dementsprechend meist mit einem erheblichen serverseitigen Programmieraufwand verbunden.

Zur Realisierung dieses Konzeptes wurde als Anforderung die Implementierung einer Template-Engine, die eine Auslagerung von einzelnen Seitenbestandteilen in Partials ermöglicht und dabei keine Programmierkenntnisse erfordert, definiert. Die erstellten Partials sollten im Anschluss, je nach Geräteklasse, automatisch ausgetauscht werden.

Durch die im Framework integrierte server- und clientseitige Geräteerkennung, kann auch auf die entsprechende Klasse eines Geräts zugegriffen werden. Erkannte Geräte werden in die drei gängigen Geräteklassen Desktop, Tablet und Mobile eingeteilt. Wird ein Gerät nicht erkannt, wird auf eine Fallback-Geräteklasse zurückgegriffen. Diese kann in der Konfiguration des Framework-Prototyps definiert werden. Basierend auf dieser Klas-

sifizierung konnte eine entsprechende Anbindung und Integration der Mustache Template-Engine für das Framework realisiert werden (siehe auch Abschnitt 5.3.4). Diese ermöglicht eine Optimierung und einen Austausch von einzelnen Seitenbestandteilen, basierend auf den definierten Geräteklassen. Einzelne Bestandteile einer responsiven Website können dabei in eigene Partial HTML-Dateien ausgelagert und somit für jede Geräteklasse optimiert werden. Die Einbindung dieser austauschbaren Bestandteile in das übergeordnete responsive Layout, erfolgt durch ein simples Mustache Tag. Basierend auf der ermittelten Geräteklasse werden die Partials im Anschluss automatisch ausgetauscht. Diese Vorgehensweise ermöglicht auch die Ausnahme von Inhalten, wenn diese von einer bestimmten Geräteklasse nicht benötigt werden. Dementsprechend erhöht sich die Flexibilität einer responsiven Website was wiederum Performance-Optimierungen für jede Geräteklasse ermöglicht.

Wird ein RESS-System entwickelt, werden die einzelnen austauschbaren Inhalte gewöhnlich auf verschiedene PHP Dateien aufgeteilt. Folglich ist eine Einbindung und ein Austausch dieser Dateien mit einem wesentlichen PHP Programmieraufwand verbunden. Die Mustache Template-Engine ermöglicht die Erstellung von Templates und Partials in gängigem HTML. Da es sich um eine sogenannte *Logic-Less* Template-Engine handelt, wird in den erstellten Dateien keine Programmierlogik unterstützt. Die Funktionen des Frameworks wurden allerdings an spezielle Mustache Tags gekoppelt (siehe Abschnitt 5.4). Dementsprechend sind für die Nutzung der Framework-Funktionalität in den erstellten Templates und Partials keine Programmierkenntnisse erforderlich.

6.1.3 Individuelle Konfiguration

Wie in Abschnitt 4.1.3 beschrieben, wird bei der Erstellung einer Website gewöhnlich auf eine Reihe von Drittanbieter-Lösungen zurückgegriffen. Beispielsweise wird für die Umsetzung des Website-Frontends ein Responsive Web Framework oder für die Ermittlung der Gerätedaten, ein gängiges *Device Description Repository* verwendet. Diese Lösungen sind meist an die verschiedensten Lizenzbestimmungen gebunden. Aus diesem Grund sollte der Framework-Prototyp ein hohes Maß an Flexibilität aufweisen und einem Webdesigner eine einfache Konfiguration ermöglichen.

Um die Wiederverwendbarkeit und Flexibilität des Frameworks zu erhöhen, wurden verschiedene bekannte *Device Description Repositories* implementiert (siehe auch Abschnitt 5.2.1). Eine zentrale Konfigurationsdatei ermöglicht dabei die Auswahl und Konfiguration der gewünschten Bibliothek. Diese ist dabei direkt über einen Browser aufrufbar und benötigt keine Programmierkenntnisse. Darüber hinaus stellt das visuelle Konfigurationsformular eine Reihe von weiteren Einstellungsmöglichkeiten, wie das Setzen der Fallback-Geräteklasse, die Konfiguration der Bildoptimierungsfunktion

oder die Aktivierung der genannten Bandbreitenüberprüfung, zur Verfügung (siehe Abb. 5.6).

Für die Entwicklung des Website-Frontends wurde die Abhängigkeitsverwaltung Composer in das Framework integriert (siehe Abschnitt 5.1.1). Composer ermöglicht eine einfache Verwaltung und Aktualisierung von benötigten Drittanbieter-Bibliotheken und kann somit auch zu einer Erweiterung des Framework-Prototyps genutzt werden. Beispielsweise kann über Composer ein Responsive Web Framework, wie *Twitter Bootstrap*, heruntergeladen und eingebunden werden. Auch die verwendete Mustache Template-Engine wurde über die Abhängigkeitsverwaltung installiert und kann somit in weiterer Folge einfach aktualisiert werden.

Durch den flexiblen Aufbau ist das erstellte Framework vielseitig einsetzbar. Es ist anpassungsfähig, einfach erweiterbar und ermöglicht eine individuelle Konfiguration.

6.2 Leistungsvergleich

Um eventuelle Performanceverbesserungen durch die Integration einer serverseitigen Komponente zu ermitteln, wird im folgenden Abschnitt ein Leistungsvergleich durchgeführt. Zu diesem Zweck wird ein responsives Layout entworfen und mit clientseitigem Responsive Webdesign, mit dem neuen *<picture>* HTML-Element und der Attribute *srcset* und *sizes* sowie auf Basis des Framework-Prototyps umgesetzt. Die erstellten Website-Versionen werden anschließend auf einen Webserver geladen und für jede Geräteklasse entsprechende Performance-Messungen durchgeführt.

6.2.1 Umsetzung der Testanwendung

Im ersten Schritt wird ein einfaches und gängiges responsives Layout entworfen und für die verschiedenen definierten Geräteklassen optimiert (siehe Abb. 6.1). Das Layout der Website orientiert sich dabei an dem aktuell im Trend liegenden One-Page Design. Das bedeutet, dass die Website auf Unterseiten verzichtet und alle Inhalte untereinander angeordnet werden [45]. Das Layout unterteilt sich dabei in verschiedene Bereiche. Im oberen Teil der Website befindet sich eine *Slideshow* mit drei wechselnden Bildern. Diese wird auf einem Tablet skaliert und auf einem mobilen Gerät durch ein statisches kleines Bild ersetzt. Darunter findet sich ein dreigeteilter Abschnitt mit Text und grafischen Icons sowie eine längere Textpassage. Unterhalb des Textabschnittes werden weitere Bilder in das Layout integriert. Die eingebundene Tabelle kann auf einem mobilen Gerät nicht optimal dargestellt bzw. konvertiert werden. Aus diesem Grund wird für die Darstellung auf einem Tablet und einem mobilen Gerät ein anderes Layout entworfen und der Abschnitt entsprechend ausgetauscht. Am Ende der Website wird noch ein Youtube Video über den verfügbaren *iframe*-Code eingebettet. Dieses

Abbildung 6.1: Das erstellte responsive Layout in den drei definierten Ansichten (Desktop, Tablet und Mobile).

wird nur in der Desktop Version der Website angezeigt und in den anderen Versionen ausgeblendet. Aufgrund der minimalistischen Navigation kann die Optimierung für die verschiedenen Geräteklassen in diesem Beispiel mit CSS erfolgen. Jedoch wird das Logo der Website, aufgrund der Platzverhältnisse auf einem Tablet und einem mobilen Gerät ausgetauscht und auf diesen Geräten durch eine schmalere Bilddatei ersetzt.

Das Layout wird anschließend in drei verschiedenen Website-Versionen umgesetzt. Die erste Version basiert auf clientseitigem Responsive Webdesign und nutzt keine zusätzlichen Verbesserungsmaßnahmen. Das bedeutet, dass die Website mit reinem HTML und CSS umgesetzt wird und alle Bilder über das gängige *<img>* HTML-Element eingebunden werden. Ein Austausch von Websitekomponenten, basierend auf den definierten Geräteklassen, ist mit clientseitigem Responsive Webdesign nicht möglich und kann nur, wie in Abschnitt 6.1.2 erwähnt, über CSS realisiert werden.

Die zweite Version basiert ebenfalls auf clientseitigem Responsive Webdesign, nutzt jedoch für die Einbindung von Bilddateien das neue *<picture>* HTML-Element und die neuen Attribute *srcset* und *sizes* (siehe auch Abschnitt 3.2.1). Zusätzlich wird das, in Abschnitt 3.2.2 beschriebene, JavaScript Polyfill Plugin *Picturefill.js* als Fallback-Lösung eingebunden. Basierend auf dieser Tatsache wird jede Bilddatei in verschiedenen Auflösungen erstellt und im Anschluss entsprechend eingebunden. Zusätzlich ermöglicht das *<picture>* HTML-Element auch den Austausch einer Bilddatei. Folglich können die unterschiedlichen Logo Grafiken über das *<picture>* Element ausgetauscht werden und es entfällt eine doppelte Einbindung. Ein gerätespezifischer Wechsel von anderen Seitenelementen ist in dieser Version allerdings ebenfalls nicht möglich.

Die letzte Website-Version basiert auf dem erstellten Framework. Es erweitert somit das responsive Layout um eine serverseitige Komponente. Diese wird für die Optimierung der eingebundenen Bilddateien und den Austausch von Seitenbestandteilen verwendet. Dementsprechend werden einzelne Websitekomponenten, wie die erwähnte Tabelle, in Partials ausgelagert und je nach Geräteklasse serverseitig ausgetauscht. In der Konfiguration des Frameworks wird für die serverseitige Geräteerkennung das WURFL *Device Description Repository* ausgewählt. Zusätzlich wird die Bildoptimierungsfunktion entsprechend konfiguriert und an die *Breakpoints* des erstellten Layouts gekoppelt. Um gleiche Voraussetzungen zu schaffen, wird die optionale Bandbreitenüberprüfung für den Leistungsvergleich deaktiviert.

Die Umsetzung der Test-Website erfolgt in allen drei Fällen mit dem Responsive Web Framework *Bootstrap* (siehe auch Abschnitt 3.1). Die benötigten Dateien werden dabei über einen Verweis auf das *Bootstrap Content Delivery Network* (CDN) eingebunden. Zusätzlich wird die aktuelle Version der JavaScript Bibliothek *jQuery* eingebunden, da diese von *Bootstrap* für die Realisierung der *Slideshow* benötigt wird. Die eigenen CSS Regeln sowie der JavaScript Code für die *Slideshow* und die Navigation, werden in sepa-

Tabelle 6.1: Die Tabelle enthält die Geräte die für die Performance-Messungen verwendet werden.

Geräteklasse	*Gerät*	*Betriebssystem*	*Browser*
Desktop	Apple iMac 27, Late 2012	OS X Yosemite 10.10.3	Google Chrome 43.0.2357.81 (64-bit)
Tablet	Apple iPad Mini 2	iOS 8.3	Safari, Google Chrome 42.0.2311.47
Mobile	Apple iPhone 6	iOS 8.3	Safari, Google Chrome 42.0.2311.47

raten Dateien definiert. Diese werden direkt eingebunden und nicht komprimiert. Der HTML, CSS und JavaScript Code der drei Websites ist somit, bis auf die Einbindung der Bilddateien, identisch. In der Website-Version, die auf dem erstellten Framework basiert, werden zusätzlich noch einzelne Seitenbestandteile in Partials ausgelagert. Um die Standardkonformität der drei erstellten Website-Versionen zu überprüfen und zu gewährleisten, werden diese im Anschluss über das *W3C Markup Validation Service*[1] validiert.

6.2.2 Vergleichsprozess

Um bei dem Leistungsvergleich einheitliche Voraussetzungen zu schaffen, werden die drei Websites auf demselben Webserver platziert. Es handelt sich dabei um einen Domainserver des Anbieters World4You Internet Services GmbH[2] mit Standort in Linz. Für alle Tests wird im Anschluss dieselbe Hardware und Internetverbindung verwendet. Die Testgeräte sind dabei über eine 16 Mbit/s Netzwerkverbindung des Providers A1 Telekom Austria mit dem Internet verbunden.

Anschließend werden für die definierten Geräteklassen (siehe Tab. 6.1) verschiedene Performance-Messungen mit dem frei verfügbaren Analyse-Tool *YSlow*[3] durchgeführt. *YSlow* analysiert die Performance des DOMs einer Website. Dabei erfasst das Tool verschiedene Messwerte und führt eine umfassende Performance-Analyse durch. *YSlow* ist als Browser-Plugin für gängige Browser verfügbar. Zusätzlich bietet es auch die Möglichkeit die Analyse über eine JavaScript Funktion aufzurufen. Diese kann als Lesezeichen gespeichert werden und ermöglicht somit auch die Überprüfung einer Website auf einem mobilen Gerät.

Darüber hinaus wird die Ladedauer der einzelnen Website-Versionen für jede Geräteklasse ermittelt. Für die Analyse werden die *Chrome DevTools*[4]

[1] W3C Validator Website: http://validator.w3.org/
[2] World4You Website: http://www.world4you.at/
[3] YSlow Website: http://yslow.org/
[4] Chrome DevTools Website: https://developer.chrome.com/devtools/

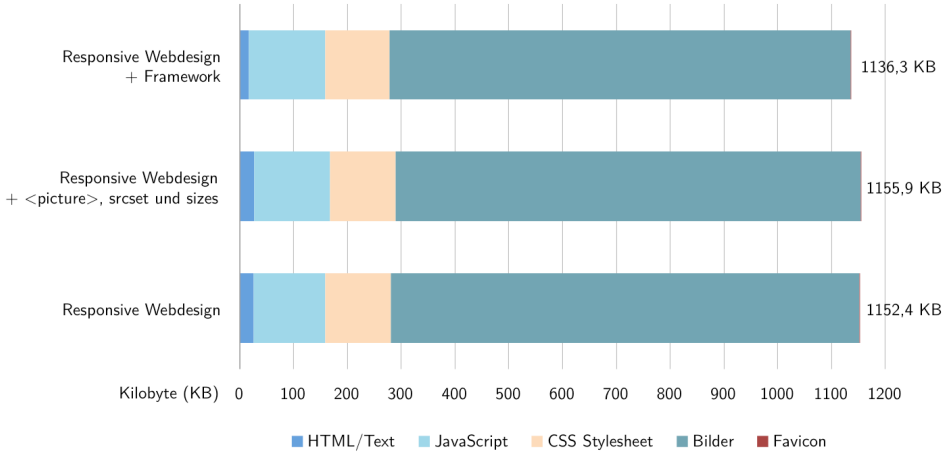


Abbildung 6.2: Verteilung der Dateigrößen der jeweiligen Website-Version auf einem Desktop PC.

von Google herangezogen. Jede Website wird dabei, bei deaktiviertem Cache, fünf mal hintereinander aufgerufen. Anschließend werden die ermittelten Ladezeiten für jede Geräteklasse notiert und der jeweilige Mittelwert bestimmt (siehe Abb. 6.6).

6.2.3 Ergebnisse

Das Analyse-Tool *YSlow* erstellt eine Übersicht über die geladenen Ressourcen einer Website. Des weiteren wird die Dateigröße jeder Ressource angezeigt, die Daten in sprechende Kategorien zusammengefasst und summiert. Abbildung 6.2 zeigt die Verteilung der Dateigrößen, der einzelnen Website-Versionen, auf einem Desktop PC. Da in dieser Layout-Version hochauflösende Bilder und alle Websitekomponenten geladen werden, verwundert es nicht, dass die Dateigrößen der einzelnen Websites beinahe identisch ausfallen. Durch die Nutzung des Frameworks können jedoch einzelne austauschbare Seitenbestandteile, wie die integrierte Tabelle, in Partials ausgelagert werden, was eine Reduzierung der Dateigröße im Bereich HTML/Text ermöglicht. Vergleicht man das Ergebnis mit der Abbildung 6.3, welche die Anzahl der HTTP-Requests der jeweiligen Test-Website darstellt, ist ein Zusammenhang ersichtlich. Die Version die auf dem Framework basiert, muss den Framework Code laden und sendet deshalb mehr Requests. Allerdings werden einzelne Bilddateien, wie die austauschbare Logo Grafik, aufgrund der Partials nicht doppelt eingebunden, was wiederum zu einer Reduzierung der Anfragen führt. Die beiden Website-Versionen die auf clientseitigem Responsive Webdesign basieren, müssen austauschbare Inhalte doppelt laden, da ein serverseitiger Austausch dieser Seitenbestandteile nicht möglich ist.

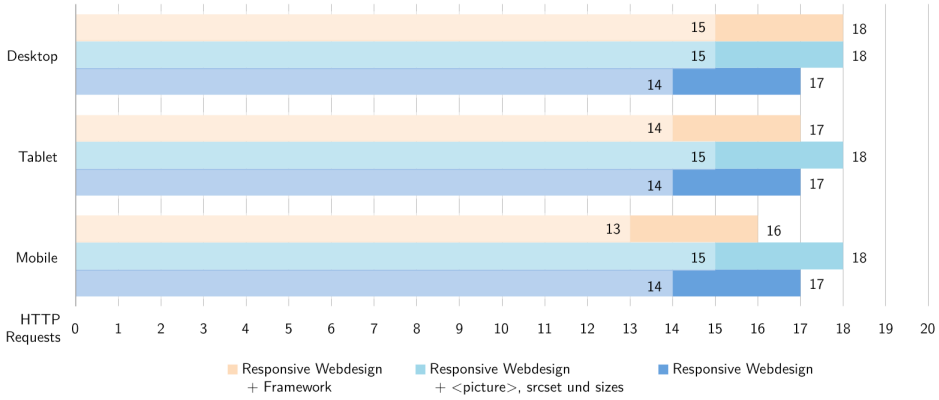


Abbildung 6.3: Anzahl der HTTP-Requests der einzelnen Website-Versionen bei leerem Cache und nachdem einige Ergebnisse der Anfragen im Cache gespeichert wurden (transparent).

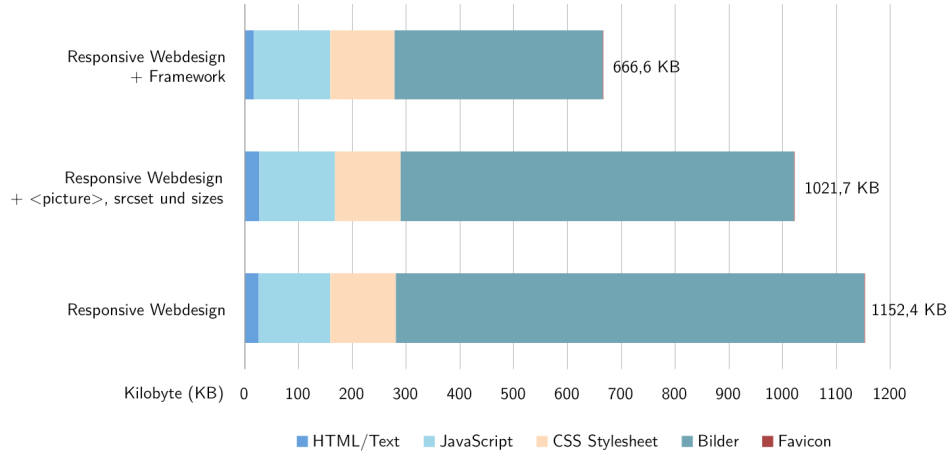


Abbildung 6.4: Verteilung der Dateigrößen der jeweiligen Website-Version auf einem Tablet.

Zur Nutzung des *<picture>* HTML-Elements und der Attribute *srcset* und *sizes* wurde in der entsprechenden Website-Version das JavaScript Polyfill Plugin *Picturefill.js* als Fallback-Lösung eingebunden, was ebenfalls einen zusätzlichen Request verursacht. Dies führt auf einem Desktop PC zu einem beinahe identischen und ausgeglichenen Ergebnis. Auch ein Vergleich mit den, in Abbildung 6.6 dargestellten, Ladezeiten deckt sich mit dieser Aussage.

Werden die Test-Websites jedoch auf einem Tablet aufgerufen, ist bereits, wie in Abbildung 6.4 ersichtlich, eine signifikante Änderung der Dateigrö-

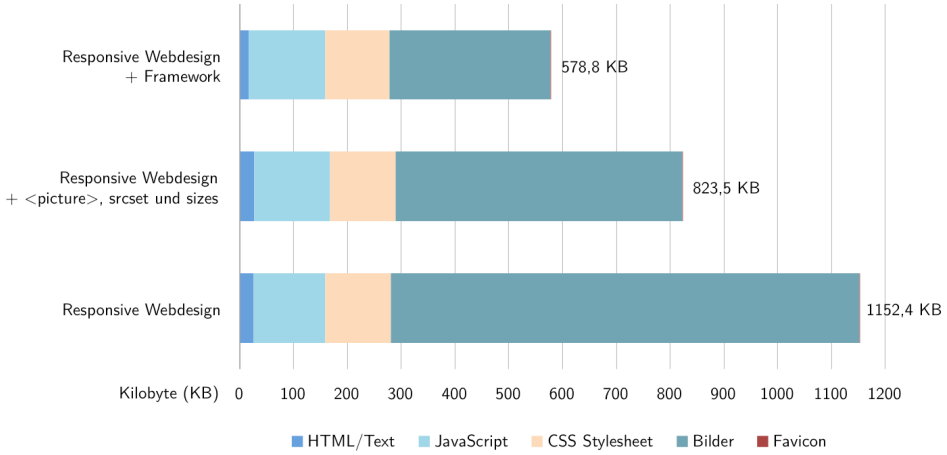


Abbildung 6.5: Verteilung der Dateigrößen der jeweiligen Website-Version auf einem mobilen Gerät.

ßen wahrnehmbar. Die im Framework integrierte Optimierungsfunktion reduziert die eingebundenen Bilder und vermindert somit die Dateigröße der Website von 1136,3 KB auf 666,6 KB um ca. 41,3 %. Auch die Nutzung des *<picture>* Elements und der *srcset* und *sizes* Attribute ermöglicht eine Reduzierung der Dateigröße einer Website. Allerdings kann über die Attribute *srcset* und *sizes* nicht exakt bestimmt werden, welche Bilddatei von einem Browser geladen werden soll. Es können aktuell nur entsprechende Vorschläge definiert werden. Die Wahl der geeigneten Grafik liegt schlussendlich bei dem genutzten Browser [57]. Dieser wählt auf einem Tablet auch teilweise niedriger aufgelöste Bilddateien aus, wobei das über die Attribute definierte volle Reduzierungspotential jedoch nicht ausgeschöpft wird. Nutzt eine responsive Website keine zusätzlichen Verbesserungsmaßnahmen und werden dementsprechend alle Bilder über das *<img>* HTML-Element eingebunden, können diese auf einem Gerät mit einer niedrigeren Auflösung auch nicht ausgetauscht bzw. reduziert werden. Folglich verfügt die Website auch auf einem Tablet über dieselbe Dateigröße wie auf einem Desktop PC. In diesem Fall ist ebenfalls der Zusammenhang zwischen der Dateigröße und der, in Abbildung 6.6 dargestellten, Ladezeit der einzelnen Website-Versionen ersichtlich. Durch die Reduzierung der Dateigröße, verkürzt sich auch die Ladedauer der jeweiligen Website. Ein gerätespezifischer Austausch von Seitenbestandteilen ist mit clientseitigem Responsive Webdesign nicht möglich. Dementsprechend ändert sich auch, wie in Abbildung 6.3 ersichtlich, die Anzahl der HTTP-Request für die beiden Website-Versionen nicht. Einzelne Inhalte, wie das eingebettete Video, können somit nur über CSS ausgeblendet werden. Das Framework ermöglicht eine Auslagerung von Seitenbestandteilen in Partials. Folglich wird das Video nur in der Desktop Version der

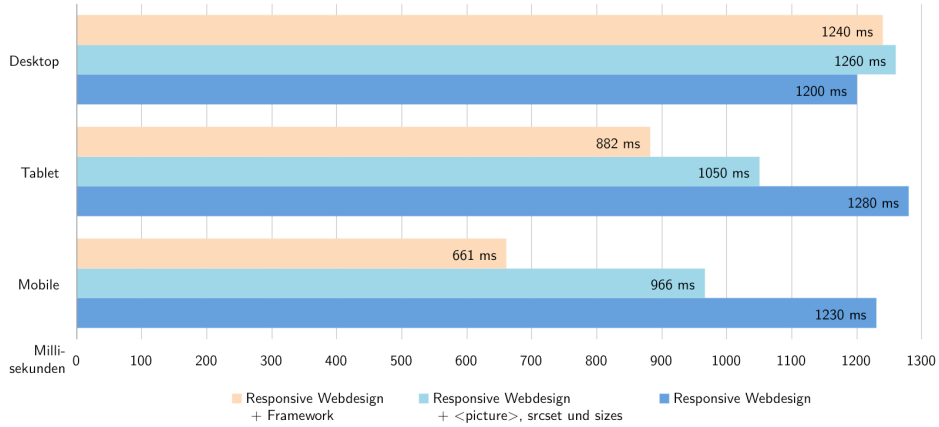


Abbildung 6.6: Durchschnittliche Ladedauer der einzelnen Website-Versionen für jede Geräteklasse.

Website eingebunden, was auf einem Tablet und einem mobilen Gerät zu einer Reduzierung der Requests führt.

In der mobilen Version der erstellten Websites wird die *Slideshow* durch ein statisches kleineres Bild ersetzt. Die Website-Version die auf dem erstellten Framework basiert, ermöglicht die Auslagerung der *Slideshow* in eine separate Partial Datei. Diese wird auf einem mobilen Gerät einfach ausgetauscht, was wiederum, wie in Abbildung 6.3 ersichtlich, eine Reduzierung der Requests herbeiführt. In den anderen beiden Versionen kann der Austausch nur über CSS durchgeführt werden. Die Anzahl der HTTP-Requests bleibt bei diesen Websites somit, unabhängig von dem verwendeten Gerät, identisch. Auch die eingebundenen Bilddateien werden im Framework für die niedrigere Auflösung des mobilen Geräts noch einmal serverseitig reduziert und ausgetauscht (siehe Abb. 6.5). Die zusätzliche Optimierung ermöglicht eine Reduzierung der Dateigröße von 666,6 KB auf 578,8 KB um 13,2 %. Dies entspricht einer Reduzierung der Gesamtdateigröße von 1136,3 KB um fast 50 %. Auch die Nutzung des *<picture>* Elements und der Attribute *srcset* und *sizes* bringt eine Reduzierung der Gesamtdateigröße von 1155,9 KB auf 823,5 KB um immerhin 28,8 %. Die Dateigröße der rein auf Responsive Webdesign basierenden Version bleibt wie erwartet auf allen drei Geräten identisch. Die erneute Reduzierung der HTTP-Requests sowie der Dateigrößen der einzelnen Website-Versionen wirkt sich auch auf die Ladedauer dieser aus. Wie in Abbildung 6.6 dargestellt, reduziert sich die Ladezeit der Website durch die Nutzung des Frameworks von 1240 ms, auf einem Desktop PC, auf 661 ms, auf einem mobilen Gerät, um mehr als 46 %. Durch die Verwendung des *<picture>* Elements und der *srcset* und *sizes* Attribute, ist ebenfalls eine signifikante Reduzierung der Ladezeit messbar. Wird Re-

sponsive Webdesign ohne zusätzliche Verbesserungsmaßnahmen eingesetzt, werden dieselben Daten an alle Geräte gesendet. Dementsprechend ändert sich auch die Ladedauer auf einem Tablet und einem mobile Gerät nicht, was bei einer langsamen Netzwerkverbindung zu einer schlechteren Performance führen kann.

Die Evaluierung zeigt, dass die Erweiterung von Responsive Webdesign um eine serverseitige Komponente, signifikante Performanceverbesserungen ermöglicht. Dabei werden eingebundene Bilddateien exakt optimiert und, je nach Auflösung eines Geräts, entsprechend ausgetauscht, was eine erhebliche Reduzierung der Dateigröße einer responsiven Website darstellt. Zusätzlich können einzelne Seitenbestandteile in Partials ausgelagert und gerätespezifisch ausgewechselt werden. Dies führt zu einer höheren Flexibilität, kürzeren Ladezeiten und einer damit verbundenen Verbesserung der Performance. Auch die Verwendung des neuen *<picture>* HTML-Elements und der Attribute *srcset* und *sizes* ermöglicht eine Reduzierung der Dateigröße und eine Verkürzung der Ladedauer. Allerdings ist über die Attribute derzeit noch keine exakte Spezifikation der gewünschten Bilder, sondern nur eine Art Empfehlung möglich. Die Auswahl der geeigneten Bilddatei ist im Anschluss dem Browser überlassen. Dementsprechend kann auch an ein Gerät mit einem niedrig aufgelösten Display eine hochauflösende Bilddatei gesendet werden, auch wenn über das *srcset* Attribut eine niedriger aufgelöste Bilddatei für die Auflösung des Geräts spezifiziert wurde [57]. Des weiteren werden die Elemente noch nicht von jedem Browser unterstützt, somit muss immer eine zusätzliche Fallback-Lösung implementiert werden. Dennoch ermöglicht die Nutzung dieser Elemente und Attribute deutliche Verbesserungen hinsichtlich der Dateigröße und der Ladezeit einer Website. Wird eine Website mit reinem clientseitigen Responsive Webdesign erstellt, werden dieselben Daten an alle Geräte gesendet. Einzelne austauschbare Seitenbestandteile werden nur ausgeblendet und im Hintergrund trotzdem geladen. Folglich weisen responsive Websites auch auf einem mobilen Gerät eine hohe Dateigröße auf. Dies führt in weiterer Folge zu langen Ladezeiten, was vor allem bei einer langsamen Netzwerkverbindung zu einem schlechten Nutzungserlebnis beitragen kann.

Kapitel 7

Schlussbemerkungen

Die Nutzung des Internets hat sich in den letzten Jahren stark verändert. Eine Vielzahl an verschiedenen mobilen Geräten löst den Desktop PC als primäres Medium zur Internetnutzung zunehmend ab. All diese Geräte weisen hinsichtlich der Darstellung und Bedienung einer Website verschiedene Eigenschaften auf und bieten unterschiedliche Interaktionsmöglichkeiten. Folglich wächst auch der Bedarf für flexible und anpassungsfähige Umsetzungsstrategien. Responsive Webdesign hat sich über die letzten Jahre schnell weiterentwickelt und ist zu einem der bekanntesten Lösungsansätze im mobilen Web geworden. So ist eine Website, die mit Responsive Webdesign erstellt wurde zukunftssicher, da sie auf jedem neuen Gerät und jeder Displayauflösung ein entsprechendes Ergebnis liefert. Darüber hinaus entfällt ein mehrfacher Wartungsaufwand, da eine einzige Website an alle Geräte ausgeliefert wird. Dieser Vorteil ist jedoch gleichzeitig auch ein wesentlicher Nachteil des Konzeptes. Aktuell bietet Responsive Webdesign keine Möglichkeit einzelne Seitenbestandteile für bestimmte Geräteklassen zu optimieren. Auch eine Reduzierung der eingebundenen Bilddateien ist mit einem erheblichen zusätzlichen Aufwand verbunden und meist nur durch die Nutzung einer entsprechenden Drittanbieter-Lösungen realisierbar. Durch die Einführung des neuen *<picture>* HTML-Elements und der Attribute *srcset* und *sizes* soll eine Optimierung und ein Austausch von eingebundenen Bildern in Zukunft mit reinem HTML möglich werden. Aktuell werden diese Elemente allerdings noch nicht von jedem Browser unterstützt wodurch immer eine zusätzliche Fallback-Lösung genutzt werden muss. Das theoretische RESS Konzept von Luke Wroblewski erweitert Responsive Webdesign um eine serverseitige Komponente. Es kombiniert somit die Vorteile einer separaten mobilen Website bzw. Adaptive Webdesign mit einem flexiblen responsiven Layout und stellt damit eine zukunftsfähige Lösung dar. Allerdings gibt es für die Umsetzung eines solchen Systems keine konkreten Vorgaben. Die Nutzung ist dementsprechend mit einem erheblichen serverseitigen Programmieraufwand verbunden.

Der entwickelte Framework-Prototyp repräsentiert eine innovative Möglichkeit zur Umsetzung eines RESS Konzeptes. Durch die Nutzung einer Geräteerkennung können bestimmte Geräteeigenschaften ermittelt und eine entsprechende Klassifizierung durchgeführt werden. Da eine serverseitige Geräteerkennung auch fehlschlagen kann, werden die ermittelten Daten über eine zusätzliche clientseitige Überprüfung durch das entwickelte System kontrolliert. Die Geräteinformationen werden im Anschluss für eine serverseitige Reduzierung der eingebundenen Bilddateien und zum Austausch von Seitenbestandteilen, basierend auf den definierten Geräteklassen, verwendet. Anhand der Kombination und Integration dieser Verbesserungsmaßnahmen in einem Framework, wird in weiterer Folge eine einfache Nutzung des Konzeptes, ohne serverseitigen Programmieraufwand, ermöglicht.

Ein Vergleich mit klassischem Responsive Webdesign zeigt, dass durch die Implementierung und Integration einer serverseitigen Komponente signifikante Performanceverbesserungen erreicht werden. Die auflösungsabhängige Optimierung der eingebundenen Bilddateien ermöglicht eine deutliche Reduzierung der Dateigröße einer Website. Zusätzlich können einzelne Seitenbestandteile für jede Geräteklasse optimiert und ausgetauscht werden, was zu einer Reduzierung der HTTP-Requests beiträgt. Werden bestimmte Seiteninhalte oder eingebundene Dateien von einer Geräteklasse nicht benötigt, werden diese auch nicht an das Gerät gesendet. Folglich verkürzt sich nicht nur die Ladezeit einer responsiven Website. Durch die Optimierung einzelner Websitekomponenten für jede Geräteklasse kann auch ein optimales Nutzungserlebnis garantiert werden.

Responsive Webdesign hat sich in den letzten Jahren von einem Trend zur Zukunft des geräteübergreifenden Webs entwickelt. Neue Spezifikationen des W3C, wie das *<picture>* HTML-Element der RICG (Responsive Images Community Group), unterstreichen diese Aussage. Sie zeigen, dass die Entwicklung des Responsive Webdesign Konzeptes noch lange nicht abgeschlossen ist, sondern permanent an einer Weiterentwicklung und Verbesserung gearbeitet wird. Dennoch bleibt eine der größten Einschränkungen von Responsive Webdesign bestehen. Um eine optimale User Experience (siehe Abschnitt 2.3.3) über alle Geräte hinweg zu gewährleisten, muss für jede Geräteklasse eine entsprechende Optimierung von bestimmten kontext- und gerätebezogenen Websitekomponenten erfolgen. Eine gerätespezifische Optimierung von einzelnen Seitenbestandteilen ist mit clientseitigem Responsive Webdesign jedoch nicht möglich und aktuell nur durch eine serverseitige Erweiterung des Konzeptes realisierbar. Ob es, wie bei Adaptive Webdesign, allerdings auch bei Responsive Webdesign zu einer Kombination aus clientseitigem Layout und serverseitiger Komponente und somit zu einer allgemein gültigen Lösung kommt, bleibt abzuwarten.

Anhang A

Inhalt der CD-ROM/DVD

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

A.1 Masterarbeit

Pfad: /

Poemer_Christoph_2015.pdf Masterarbeit (Gesamtdokument)

A.2 Online-Literatur

Pfad: /Literatur

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

A.3 Abbildungen

Pfad: /Abbildungen

*.pdf . . . . . Vektorgrafiken

*.jpg, *.png . . . . . Original Rasterbilder

A.4 Leistungsvergleich

Pfad: /Leistungsvergleich

/Testanwendungen . . . Test-Websites für Leistungsvergleich

/Ergebnisse . . . . . Ergebnisse der Performance-Messungen

A.5 Projektdateien

Pfad: /Projekt

/respond	Framework-Prototyp zur serverseitigen Optimierung von Responsive Webdesign
respond.zip	Framework-Prototyp im ZIP-Dateiformat

Quellenverzeichnis

Literatur

[1] Ronan Cremin. „Lightening Your Responsive Website Design With RESS“. In: *Responsive Web Design, Vol. 2*. Hrsg. von Vitaly Friedman. 2. Aufl. Freiburg: Smashing Magazine, 2014, S. 93–110 (siehe S. 24, 25, 46).

[2] Lyza Danger Gardner und Jason Grigsby. *Mobiles Web von Kopf bis Fuß*. Köln: O'Reilly, 2012 (siehe S. 6, 7, 9–12, 14, 16–20).

[3] Christian Fernandez und Christian Kühn. „Mobile Web & Responsive Webdesign“. In: *i-com* 11.1 (2012), S. 53–56 (siehe S. 26, 48).

[4] Maximiliano Firtman. *Programming the Mobile Web*. 2. Aufl. Sebastopol: O'Reilly, 2013 (siehe S. 7, 8, 44).

[5] Vitaly Friedman. „About This Book“. In: *Responsive Web Design: Solutions For Responsive Images*. Hrsg. von Vitaly Friedman. Freiburg: Smashing Magazine, 2014, S. 2 (siehe S. 30).

[6] Aaron Gustafson und Jeffrey Zeldman. *Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement*. Tennessee: Easy Readers, 2011 (siehe S. 11, 13).

[7] Shawn Jansepar. „Automate Your Responsive Images With Mobify.js“. In: *Responsive Web Design, Vol. 2*. Hrsg. von Vitaly Friedman. 2. Aufl. Freiburg: Smashing Magazine, 2014, S. 47–61 (siehe S. 50).

[8] Tim Kadlec. *Praxiswissen Responsive Webdesign*. Köln: O'Reilly, 2013 (siehe S. 36, 37).

[9] Joanna Krenz-Kurowska und Jerzy Kurowski. *RESS Essentials*. Birmingham: Packt Publishing Ltd, 2013 (siehe S. 22, 24, 26, 40, 44, 45, 66–68, 71, 75).

[10] Bruce Lawson. „Why We Shouldn't Make Separate Mobile Websites“. In: *Essentials Of Mobile Design*. Hrsg. von Talita Telma Stöckle. Freiburg: Smashing Magazine, 2012, S. 16–25 (siehe S. 24).

[11] Rachel Mccollin. „Responsive Images With WordPress' Featured Images". In: *Responsive Web Design: Solutions For Responsive Images*. Hrsg. von Vitaly Friedman. Freiburg: Smashing Magazine, 2014, S. 97–103 (siehe S. 39, 40).

[12] Jordan Moore. „Responsible Considerations For Responsive Web Design". In: *Responsive Web Design, Vol. 2*. Hrsg. von Vitaly Friedman. 2. Aufl. Freiburg: Smashing Magazine, 2014, S. 4–20 (siehe S. 4, 21, 28).

[13] Jürgen Ortmann. „Anforderungen an mobile Webseiten". In: *der webdesigner* 4 (2012), S. 14–19 (siehe S. 4, 17, 48).

[14] Eric Portis. „Responsive Images Done Right: A Guide To <picture> And srcset". In: *Responsive Web Design: Solutions For Responsive Images*. Hrsg. von Vitaly Friedman. Freiburg: Smashing Magazine, 2014, S. 13–24 (siehe S. 31–34).

[15] Ilya Pukhalski. „Rethinking Responsive SVG". In: *Responsive Web Design: Solutions For Responsive Images*. Hrsg. von Vitaly Friedman. Freiburg: Smashing Magazine, 2014, S. 116–124 (siehe S. 38).

[16] Jon Arne Saeteras. „Improve Mobile Support With Server-Side-Enhanced Responsive Design". In: *Responsive Web Design, Vol. 2*. Hrsg. von Vitaly Friedman. 2. Aufl. Freiburg: Smashing Magazine, 2014, S. 111–121 (siehe S. 66).

[17] Stephen Thomas. „Simple Responsive Images With CSS Background Images". In: *Responsive Web Design: Solutions For Responsive Images*. Hrsg. von Vitaly Friedman. Freiburg: Smashing Magazine, 2014, S. 81–96 (siehe S. 38).

[18] Tim R. Todish. „Not Your Parent's Mobile Phone: UX Design Guidelines For Smartphones". In: *Essentials Of Mobile Design*. Hrsg. von Talita Telma Stöckle. Freiburg: Smashing Magazine, 2012, S. 5–15 (siehe S. 1, 49).

[19] Estelle Weyl. „Clown Car Technique: Solving Adaptive Images In Responsive Web Design". In: *Responsive Web Design: Solutions For Responsive Images*. Hrsg. von Vitaly Friedman. Freiburg: Smashing Magazine, 2014, S. 65–80 (siehe S. 39).

[20] Tim Wright. „Picturefill 2.0: Responsive Images And The Perfect Polyfill". In: *Responsive Web Design: Solutions For Responsive Images*. Hrsg. von Vitaly Friedman. Freiburg: Smashing Magazine, 2014, S. 4–12 (siehe S. 30, 35).

[21] Christoph Zillgens. *Responsive Webdesign: Reaktionsfähige Websites gestalten und umsetzen*. München: Carl Hanser Verlag GmbH & Company KG, 2012 (siehe S. 11).

Online-Quellen

[22] URL: http://mediaqueri.es (besucht am 17. 03. 2015) (siehe S. 17).

[23] URL: http://wiki.selfhtml.org/wiki/Margin (besucht am 17. 03. 2015) (siehe S. 18).

[24] URL: http://wiki.selfhtml.org/wiki/Padding (besucht am 17. 03. 2015) (siehe S. 18).

[25] URL: http://responsivenav.com/responsive-nav.png (siehe S. 25).

[26] URL: http://getbootstrap.com (besucht am 12. 04. 2015) (siehe S. 28, 29).

[27] URL: http://foundation.zurb.com (besucht am 12. 04. 2015) (siehe S. 29).

[28] URL: http://responsiveimages.org (besucht am 26. 03. 2015) (siehe S. 30).

[29] URL: http://caniuse.com/#search=picture (besucht am 23. 04. 2015) (siehe S. 34).

[30] URL: http://caniuse.com/#search=srcset (besucht am 23. 04. 2015) (siehe S. 34).

[31] URL: http://www.mobify.com/mobifyjs/v2/docs/capturing (besucht am 22. 04. 2015) (siehe S. 36).

[32] URL: http://www.resrc.it (besucht am 22. 04. 2015) (siehe S. 36).

[33] URL: https://developers.google.com/speed/webp (besucht am 01. 04. 2015) (siehe S. 39).

[34] URL: http://php.net (besucht am 10. 05. 2015) (siehe S. 62).

[35] URL: http://json.org (besucht am 10. 05. 2015) (siehe S. 62).

[36] URL: http://wurfl.sourceforge.net (besucht am 10. 05. 2015) (siehe S. 67).

[37] URL: http://www.scientiamobile.com/cloud (besucht am 10. 05. 2015) (siehe S. 67).

[38] URL: https://deviceatlas.com (besucht am 10. 05. 2015) (siehe S. 67).

[39] URL: https://51degrees.com/products/device-detection (besucht am 10. 05. 2015) (siehe S. 68).

[40] URL: http://web.wurfl.io (besucht am 10. 05. 2015) (siehe S. 69).

[41] URL: https://mustache.github.io (besucht am 10. 05. 2015) (siehe S. 69).

[42] Nils Adermann und Jordi Boggiano. *Composer - Dependency Manager for PHP*. 2015. URL: https://getcomposer.org (besucht am 10. 05. 2015) (siehe S. 63).

[43] Sherri Alexander. *Choosing A Responsive Image Solution*. 2013. URL: http://www.smashingmagazine.com/2013/07/08/choosing-a-responsive-image-solution (besucht am 01. 04. 2015) (siehe S. 36, 37).

[44] Magnus Anders Andersen. *Getting started with RESS*. 2012. URL: http://www.creativebloq.com/responsive-web-design/getting-started-ress-5122956 (besucht am 12. 04. 2015) (siehe S. 46).

[45] Ines Angerstein. *Single Pages – Vor- und Nachteile des neuen Trends*. 2014. URL: http://www.interface-medien.de/blog/single-pages (besucht am 08. 06. 2015) (siehe S. 89).

[46] Jon Arne Sæteråas und Luca Passani. *Server-Side Device Detection With JavaScript*. 2014. URL: http://www.smashingmagazine.com/2014/07/01/server-side-device-detection-with-javascript (besucht am 13. 05. 2015) (siehe S. 69).

[47] Maxim Chernyak. *Smart Resize Image*. 2009. URL: https://github.com/maxim/smart_resize_image (besucht am 10. 05. 2015) (siehe S. 69).

[48] Chris Coyier. *FitVids.JS*. 2015. URL: http://fitvidsjs.com (besucht am 22. 04. 2015) (siehe S. 40).

[49] Thomas Drew. *Why Responsive Web Design Has To Win Out*. 2013. URL: http://www.smashingmagazine.com/2013/02/14/responsive-web-design-planning-future (besucht am 13. 03. 2015) (siehe S. 21–23).

[50] Maurice Florence. *Responsive Design oder mobile Website*. 2012. URL: http://www.pc-magazin.de/ratgeber/responsive-design-oder-mobile-website-1333388.html (besucht am 13. 03. 2015) (siehe S. 5, 7, 8, 10, 15, 24, 27).

[51] Brad Frost. *Mobile-First Responsive Web Design*. 2009. URL: http://bradfrost.com/blog/web/mobile-first-responsive-web-design (besucht am 18. 03. 2015) (siehe S. 20).

[52] Tobias Gebauer. *Die 10 besten responsive Frameworks*. 2014. URL: http://www.the-webdesign.net/die-besten-10-responsive-frameworks (besucht am 12. 04. 2015) (siehe S. 29).

[53] Marc Grabanski und Christopher Schmitt. *HiSRC*. 2014. URL: https://github.com/teleject/hisrc (besucht am 22. 04. 2015) (siehe S. 35).

[54] Jonas Hellwig. *Adaptive Website vs. Responsive Website*. 2014. URL: http://blog.kulturbanause.de/2012/11/adaptive-website-vs-responsive-website (besucht am 15. 03. 2015) (siehe S. 11–16).

[55] Jonas Hellwig. *Flexible Videos im Responsive Webdesign*. 2011. URL: http://blog.kulturbanause.de/2011/09/flexible-videos-im-responsive-webdesign (besucht am 12. 04. 2015) (siehe S. 40, 41).

[56] Jonas Hellwig. *Media Queries / Media Query*. 2013. URL: http://blog.kulturbanause.de/webdesign-lexikon/media-queries-media-query (besucht am 15. 03. 2015) (siehe S. 12).

[57] Jonas Hellwig. *Responsive Images – <picture>, srcset, sizes & Co.* 2015. URL: http://blog.kulturbanause.de/2014/09/responsive-images-srcset-sizes-adaptive (besucht am 26. 03. 2015) (siehe S. 30–34, 39, 95, 97).

[58] Jonas Hellwig. *SVG-Grafiken erstellen und einbinden*. 2015. URL: http://blog.kulturbanause.de/2014/02/svg-grafiken-erstellen-und-einbinden (besucht am 01. 04. 2015) (siehe S. 38).

[59] Nico Hemkes. *Mobile Browser mittels User Agent erkennen*. 2014. URL: http://www.netzware.net/wp-content/uploads/user-agent-aufbau.png (besucht am 13. 03. 2015) (siehe S. 7).

[60] Mario Hernandez u. a. *Responsive Images in Drupal 8*. 2015. URL: https://www.drupal.org/documentation/modules/responsive_image (besucht am 02. 04. 2015) (siehe S. 40).

[61] Pete LePage. *Darstellungsbereich festlegen*. 2014. URL: https://developers.google.com/web/fundamentals/layouts/rwd-fundamentals/set-the-viewport (besucht am 17. 03. 2015) (siehe S. 17).

[62] Ethan Marcotte. *Fluid Grids*. 2009. URL: http://alistapart.com/article/fluidgrids (besucht am 17. 03. 2015) (siehe S. 18, 19).

[63] Ethan Marcotte. *Responsive Web Design*. 2010. URL: http://alistapart.com/article/responsive-web-design (besucht am 13. 03. 2015) (siehe S. 16).

[64] Matthias Mullie. *Minify*. 2015. URL: https://github.com/matthiasmullie/minify (besucht am 10. 05. 2015) (siehe S. 70).

[65] Jakob Nielsen und Don Norman. *The Definition of User Experience*. URL: http://www.nngroup.com/articles/definition-user-experience (besucht am 21. 03. 2015) (siehe S. 1, 26).

[66] Dave Olsen. *Detector*. 2012. URL: https://github.com/dmolsen/Detector (besucht am 10. 05. 2015) (siehe S. 68).

[67] Dave Olsen. *RESS, Server-Side Feature-Detection and the Evolution of Responsive Web Design*. 2012. URL: http://dmolsen.com/2012/02/21/ress-and-the-evolution-of-responsive-web-design (besucht am 13. 03. 2015) (siehe S. 46).

[68] Dave Olsen. *The Server Side of Responsive Design*. 2013. URL: http://dmolsen.com/2013/10/24/the-server-side-of-responsive-design (besucht am 13. 03. 2015) (siehe S. 47).

[69] James Pearce. *How to use src.sencha.io*. 2011. URL: http://www.sencha.com/learn/how-to-use-src-sencha-io (besucht am 01. 04. 2015) (siehe S. 36).

[70] Amy Schade. *Responsive Web Design (RWD) and User Experience*. 2014. URL: http://www.nngroup.com/articles/responsive-web-design-definition (besucht am 21. 03. 2015) (siehe S. 26).

[71] Andreas Schäfer. *Responsive, Fluid, Adaptive – Welche Unterschiede gibt es?* 2014. URL: http://www.dieproduktmacher.com/responsive-fluid-adaptive-welche-unterschiede-gibt-es (besucht am 15. 03. 2015) (siehe S. 13, 15).

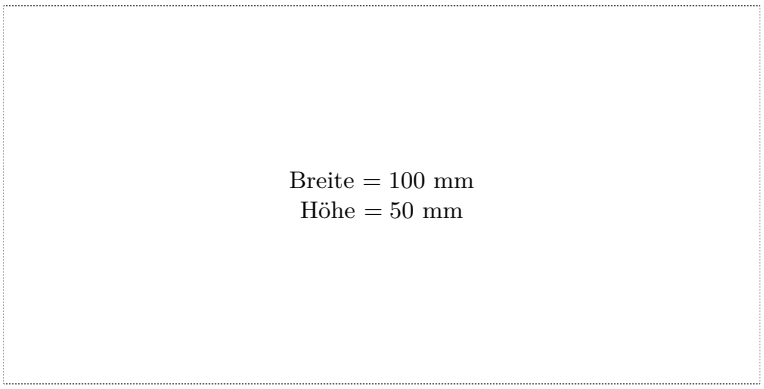
[72] Ingo Schmitt. *Responsive Images in TYPO3: So könnt ihr ab Version 6.2 LTS die Standard-Lösung anpassen und erweitern*. 2014. URL: http://t3n.de/magazin/typo3-6-2-lts-responsive-images-235194 (besucht am 02. 04. 2015) (siehe S. 39).

[73] Matt Wilcox. *Adaptive Images*. 2012. URL: http://adaptive-images.com (besucht am 22. 04. 2015) (siehe S. 37).

[74] Luke Wroblewski. *RESS: Responsive Design + Server Side Components*. 2011. URL: http://www.lukew.com/ff/entry.asp?1392 (besucht am 13. 03. 2015) (siehe S. 42, 43, 46).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —