universität wien

Diplomarbeit

Titel der Diplomarbeit

„Erfolgsfaktoren für Web Intelligence Services auf mobilen Geräten“

Verfasser

Michael Föls

Angestrebter akademischer Grad

Magister der Philosophie (Mag. phil.)

Wien, Jänner 2015

Studienkennzahl lt. Studienblatt:	A 317
Studienrichtung lt. Studienblatt:	Theater-, Film- und Medienwissenschaft
Betreuer:	Prof. DDr. Arno Scharl

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Problemstellung

Die vorliegende Diplomarbeit wurde als Abschluss des Diplomstudiums der Theater-, Film-, und Medienwissenschaft an der Universität Wien verfasst. Ziel der Arbeit ist es zu erläutern welche Herausforderung mobile Geräte wie Smartphones und Tablets an komplexe Web Intelligence Services stellen, wie sich die besonderen Eigenschaften dieser Geräte auf die Bedienung einer Applikation auswirken und welche Designansätze es bei einer Anpassung für mobile Geräte gibt. Um diese Themen möglichst umfassend aufzuarbeiten, wurde ein Prototyp entwickelt, der die theoretischen Ansätze aufgreift und in welchem diese angewendet werden um eine mobile Version eines vorhandenen Web Intelligence Services zu erstellen.

2 Einleitung

Bereits seit vielen Jahren zeichnet sich ab, dass mobile Geräte wie Smartphones und Tablets stetig ihre Verbreitung vergrößern und immer mehr Benutzer erreichen. Während der Markt an Desktopnutzern nur langsam wächst, gibt es bei der Zahl an mobilen Benutzern nach wie vor große Zuwächse. Wie Abbildung 1 zeigt hat die Anzahl an mobilen Benutzern im Jahr 2014 das erste Mal die Anzahl an Desktopnutzern überschritten. Für alle Anbieter von Medien bzw. Programmen bedeutet dies, dass es nicht länger optional ist das eigene Angebot für mobile Geräte anzubieten. Da der mobile Markt, zumindestens was die Benutzerzahlen betrifft mittlerweile der größte ist, ist eine Optimierung des eigenen Angebots für mobile Geräte essentiell.

Die Marktanteile rechtfertigen für viele Neuentwicklungen auch bereits die Auswahl von mobilen Geräten als Hauptplattform, wobei es hierzu auch notwendig ist die Marktsituation aus finanzieller Sicht zu beurteilen. Da mobile Anwendungen in der Regel um deutlich geringere Preise angeboten werden als Desktop-Anwendungen gilt es hier genaue Berechnungen hinsichtlich des Break-Even-Points anzustellen. [9] Da für viele Anwendungen allerdings die Verbreitung des Mediums wichtiger ist als der initiale Kaufpreis (sei es auf Grund von Monetarisierung durch Werbung, oder als Service-Angebot für Kunden), wird in dieser Arbeit nicht näher auf die finanziellen Gegebenheiten von mobilen Anwendungen eingegangen.

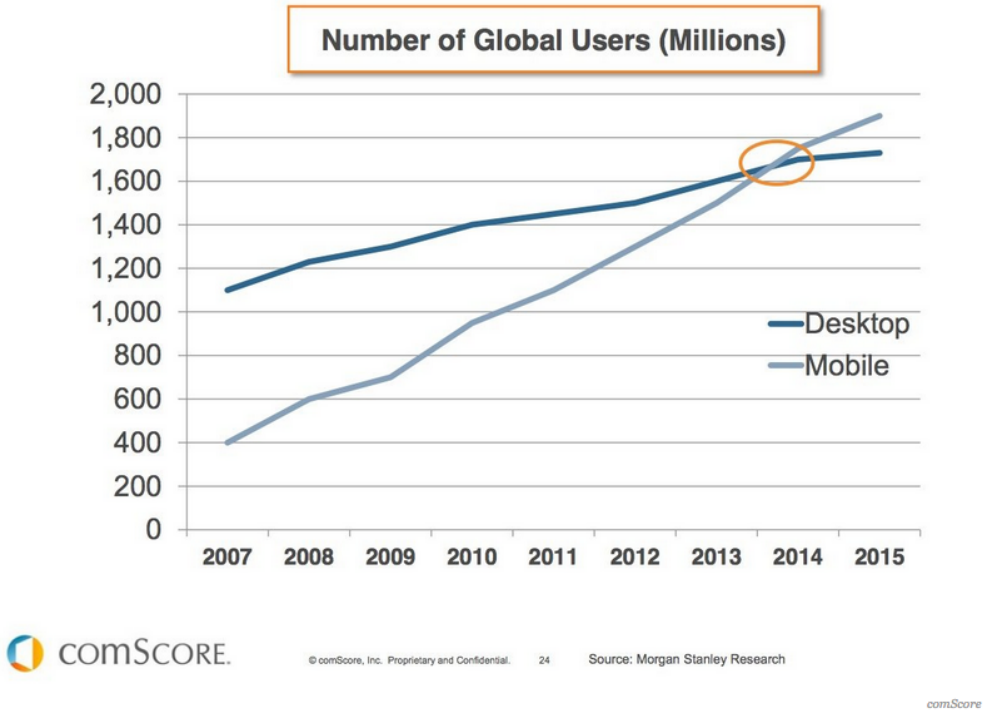


Abbildung 1: Verbreitung von mobilen Geräten im Vergleich mit traditionellen Desktopsystemen [1]

Bei sehr statischen Web-Angeboten, die vielleicht nur aus einer Ansammlung von simplen HTML-Texten bestehen, gibt es in den meisten Fällen keinerlei Probleme bei der Darstellung auf mobilen Endgeräten. Da mit den steigenden Hardwareleistungen aller Geräte, auch die Wünsche der Kunden und damit die Ziele der Entwickler größer werden, gibt es allerdings seit Jahren den Trend zu komplexeren Designs, die ohne gezielte Adaption nicht oder nur sehr schlecht auf mobilen Geräten darstellbar sind. [10]

Ebenfalls problematisch war lange Zeit die potentielle Leistungsfähigkeit von mobilen Geräten. Abbildung 2 zeigt anhand der Performance im SunSpider JavaScript Test, dass sich die Leistungsfähigkeit der Geräte aber schnell ver-

bessert. Konkret wurde die Performance zwischen dem iPad 1 und dem iPad 3 innerhalb von 2 Jahren verdoppelt. Relevant ist dies besonders für jene Medien, deren Inhalte nicht statisch bzw. textbasiert sind, sondern aufwendige Animationen, animierte Grafiken, oder ähnliche performanceintensive Tasks beinhalten. Auswirkung hat diese Entwicklung unter anderem auf die Entscheidung welche Art der Adaption von Desktopanwendung zur mobilen Anwendung möglich ist (z.B. stellen responsive Designs, in welchen die Art der Inhalte verhältnismäßig konstant sind beinahe die gleichen Hardwareanforderungen an mobile Geräte und Desktopgeräte). Aber auch welche Art von Medien überhaupt adaptierbar sind bzw. welche Features in einer mobilen Anwendungen überhaupt umsetzbar sind, hängt stark von der Hardwareleistung ab. Vergleiche wie der SunSpider Test oder andere relevante Benchmarks zeigen, dass mobile Geräte konstant aufholen und bereits jetzt sehr viele Anwendungsszenarios auf mobilen Geräten möglich sind.

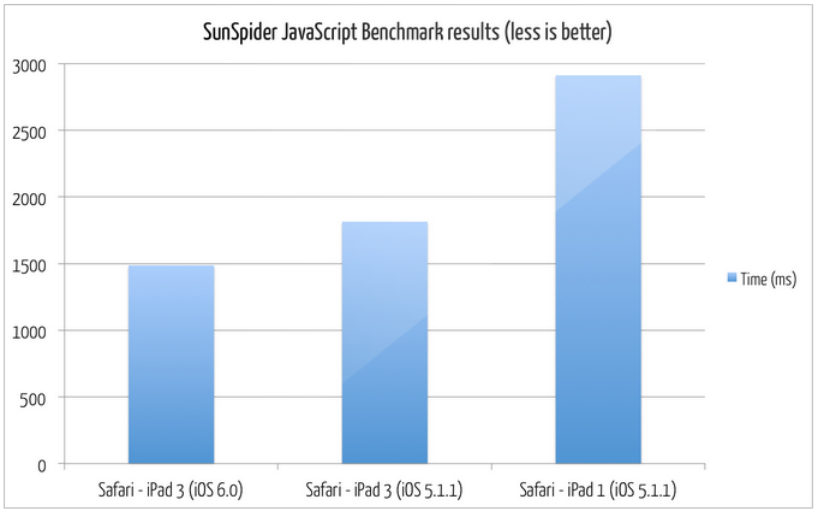


Abbildung 2: SunSpider JavaScript Benchmark - iOS Vergleich [1]

Primärer Fokus für diese Arbeit sind eben jene komplexen Desktopanwendungen, deren Umsetzung bis vor wenigen Jahren gar nicht in der aktuellen Form möglich gewesen wäre. Gerade die Adaption von komplexen, interaktiven Systemen, wie es grafische Web Intelligence Systeme häufig sind, stellen besondere Herausforderungen an die Designer und Entwickler um auf der einen Seite das Interaktionsmuster des bestehenden Systems stark genug beizubehalten um eine anhaltende Interaktionsidentifikation des Benutzers aufrecht zu erhalten und zu harte Medienbrüche zu vermeiden, aber auf der anderen Seite das System auch genug anzupassen um die Stärken von mobilen Geräten ausreichend auszunutzen.

Den Kern der Arbeit stellt dabei die Adaption eines bestehenden Desktop-Systems für mobile Geräte dar, allerdings sind die ausgearbeiteten Richtlinien und Vorschläge generell genug um für alle Szenarien des mobilen Mediendesigns Anwendung zu finden und sollen generelle Probleme und Lösungen aufzeigen. Um aktuellen Trends zu entsprechen werden mobile Geräte im Rahmen dieser Diplomarbeit als ein Überbegriff für Smartphones und Tablets definiert. Zwar wäre es auch möglich Geräte wie Notebooks in die Kategorie der mobilen Geräte aufzunehmen, allerdings sind diese Geräte hinsichtlich Betriebssystem, Bedienung, Bildschirmgröße, Nutzungsszenario und Performance so nahe an traditionellen Desktopgeräten, dass die Kategorisierung als mobiles Gerät im Rahmen der hier besprochenen Forschungsergebnisse keinen Sinn machen würde. In weiterer Folge wird unter einem mobilen Gerät gemäß aktuellen Markttrends ein Gerät mit Touchscreen verstanden. Wearables, wie Smartwatches oder Smartglasses bilden eine eigenständige Designkategorie, die gesondert analysiert werden müssste. Auf Grund ihrer aktuell minimalen Verbreitung werden diese im Rahmen dieser Arbeit aus-

geklammert. Auch ältere Telefone, die lediglich über eine Tastennavigation verfügen, allerdings dennoch über ältere WAP-Funktionen auf das Internet zugreifen können, werden wegen ihrer starken technischen Limitierung und ihrer geringen aktuellen Verbreitung im mobilen Internet in dieser Arbeit nicht berücksichtigt.

3 Touch-Interaktion

Die physische Form von mobilen Geräten, die über einen Touchscreen bedient werden, setzt sich zum aktuellen Stand der Technik aus zwei unterschiedlichen Strukturelementen zusammen. Auf der einen Seite steht der Rahmen des Geräts bzw. dessen Rücken und Seitenelemente und auf der anderen Seite steht der Touchscreen an sich. Wie unter anderem von Manovich beschrieben trägt bereits die haptische Struktur dieser Elemente wesentlich zur User-Experience bei. [11] Passend dazu steht auch der von Apple getätigte Satz „The back of our computer looks better than the front of everyone else's." [11, S. 278] Zwar hat sich diese Aussage zur damaligen Zeit auf die Rückseite von Desktop-Computer bezogen und damit hauptsächlich auf die visuelle Gestaltung, allerdings haben Geräte wie der mit einem Tragegriff versehene Computer von Apple bereits früh die haptische Bedeutung von virtuellen Informationssystemen aufgezeigt.

Im Rahmen der aktuellen Entwicklung von mobilen Geräten, laut der Smartphones immer mitgenommen werden, in die Hand genommen werden und zum permanenten Tor in die digitale Welt geworden sind, ist die haptische Eintrittsfläche zum digitalen Content noch viel relevanter geworden, da nun auch das Interface direkt mit den Fingern bedient wird und damit die Abstraktionsebene einer Maussteuerung wegfällt. Hinsichtlich dieser Entwicklung müssen auch Interfacephilosophien hinterfragt werden. Denn laut Campanelli et al. ist die Steuerung durch Berühren mit den Händen vom grundlegenden Prinzip her unterschiedlich im Vergleich zu einer Maussteuerung.

Während die Maussteuerung durch den punktuellen Cursor und die präzisen Klickevents, eine punktgenaue Navigation erzwingt, ist die Touch-Steuerung eine Methode zum Entdecken. Die Distanz zwischen dem User und dem Interface wird minimiert und das Interface sollte mit explorativen Gesten, wie Wischen, etc. bedient werden um diesen Charakter zu verstärken. Hinsichtlich der Interfacegestaltung ist allerdings zu beachten, dass das Design nicht nur die haptischen Charakteristika der Touchscreens berücksichtigt. Denn auch bei Touchscreens wechselt sich eine haptische und eine optische Interaktion (z.B. durch Betrachten von Videos oder Bildern) ab. [12]

Dabei ist der Touchscreen nicht bloß zufällig schon seit längerer Zeit ein Objekt der Begierde für die Zukunftsvorstellungen von Human Interface Designern und wurde bereits vor der Verbreitung auf dem Massenmarkt in zahlreichen Science Fiction Geschichten (unter anderem Minority Report von Steven Spielberg aus dem Jahr 2002) vorweggenommen. Der Wunsch hierbei ist die Interaktion mit einem User-Interface, das sich nahtlos in unsere Welt einfügt. Es soll nicht länger eine strikte Trennung zwischen der Interaktion mit einem Computer über eine Maus und Tastatur geben, stattdessen soll der Computer die Bilder direkt in den physischen Raum projezieren, wobei der User sie einfach greifen und mit ihnen interagieren kann. Natürlich ist dies nicht die aktuelle Anwendung von Touch-Interfaces, allerdings zeigen Projekte wie das Oculus Rift, dass sich die Industrie zunehmend in eine solche Richtung bewegt und moderne mobile Geräte mit Touch-Interface sind bereits ein erster Schritt in eine solche Zukunft. [13]

Dabei ist das grundlegende Problem der Touchscreen-Interaktion mit Geräten der aktuellen Generation, dass der User stets nur das Glas des Touchscreens

als direkte haptische Feedbackebene bekommt. Somit ist es für die direkte haptische Interaktion mit einem Touchscreen erst einmal bedeutungslos was sich auf dem Bildschirm abspielt, da das Berührungsfeedback unserer Sinne immer das gleiche ist. Um diesen Umstand zu umgehen war es zunächst notwendig den Benutzer an diese Form der direkten Interaktion zu gewöhnen und Mechanismen zu finden um die neue Form der Computerinteraktion zu etablieren. Speziell Apple versuchte dies mit der Einführung des ersten iPhone und während dem Beginn der starken Verbreitung von Touchscreen-Geräten über eine Technik namens Skeuomorphismus. Diese Design-Sprache versucht mobile Anwendungen so zu gestalten, dass sie ihren Gegenspielern in der physischen Welt ähnlich sind. [14]

Abbildung 3: Skeuomorphes Design in iOS [2]

Kennzeichen von Skeuomorphismus ist, dass die Designsprache versucht Strukturen abzubilden, die wir aus unserer Umgebung kennen. So verwendete Apple z.B. lange Zeit für den Rand ihrer Kalender-Applikation unter iOS einen Lederrand und baute den Kalender auch so auf, dass man ihn durch Touchgesten wie einen physischen Kalender bedienen konnte. Abbildung 3 zeigt wie ein solcher skeuomorpher Kalender unter iOS ausgesehen hat. Charakteristisch ist hierbei das Umblättern wie bei einem echten Kalender und die stark strukturell geprägten Ränder. Dieser Design-Ansatz birgt sowohl Vor- als auch Nachteile. Positiver Aspekt von skeuomorphen Design ist, dass die digitale Anwendung den Benutzer sofort an das entsprechende reale Objekt erinnert, das nachgestellt ist. So ist es in der Regel auch nicht notwendig zu erklären wie eine Anwendung zu bedienen ist, sondern der Benutzer orientiert sich an seinem bekannten Verhalten und berührt die Anwendung intuitiv, wie er auch mit der realen Welt interagieren würde. Dies mindert die Hemmschwelle und lässt die zuvor erwähnten Probleme von Touchscreengeräten verschwinden. Außerdem ist das verspielte und vertraute Design besonders für Benutzer die noch nicht sehr vertraut mit Touchscreengeräten sind, einladend und ermuntert zum Ausprobieren. [2] [14]

Allerdings geht mit diesem Design-Ansatz auch die starke Limitierung einher, dass sich Designer nach den Möglichkeiten und Begebenheiten der realen Welt orientieren müssen und auf die viel größeren digitalen Möglichkeiten verzichten müssen. Außerdem wird so verhindert, dass sich eine digitale Designsprache entfaltet, deren Interaktions- und Darstellungskapazitäten sich über die ihrer physischen Gegenparts hinweg begeben. Auf Grund der ausgeprägten Strukturen ist es auch schwieriger responsive Anwendungen zu gestalten, die sich an den vorhanden Platz anpassen. Durch die Verspielt-

heit des Designs und der vorhandenen Schnörkel neigen die Anwendungen auch dazu weniger performant als Anwendungen mit simpleren Interfaces zu sein. Schlussendlich hat sich in den letzten Jahren ein Trend entwickelt, der sich eher von diesem Ansatz des UI-Designs wegbewegt. Sowohl Apple, als auch Google haben sich mit den neuesten Versionen ihrer Frameworks für iOS bzw. Android von dem skeuomorphen Ansatz entfernt und setzen auf einen flachen Designansatz. Windows Phone war mit seinem Tile-Design von Beginn an auf ein flaches Design ausgerichtet. [3] [15]

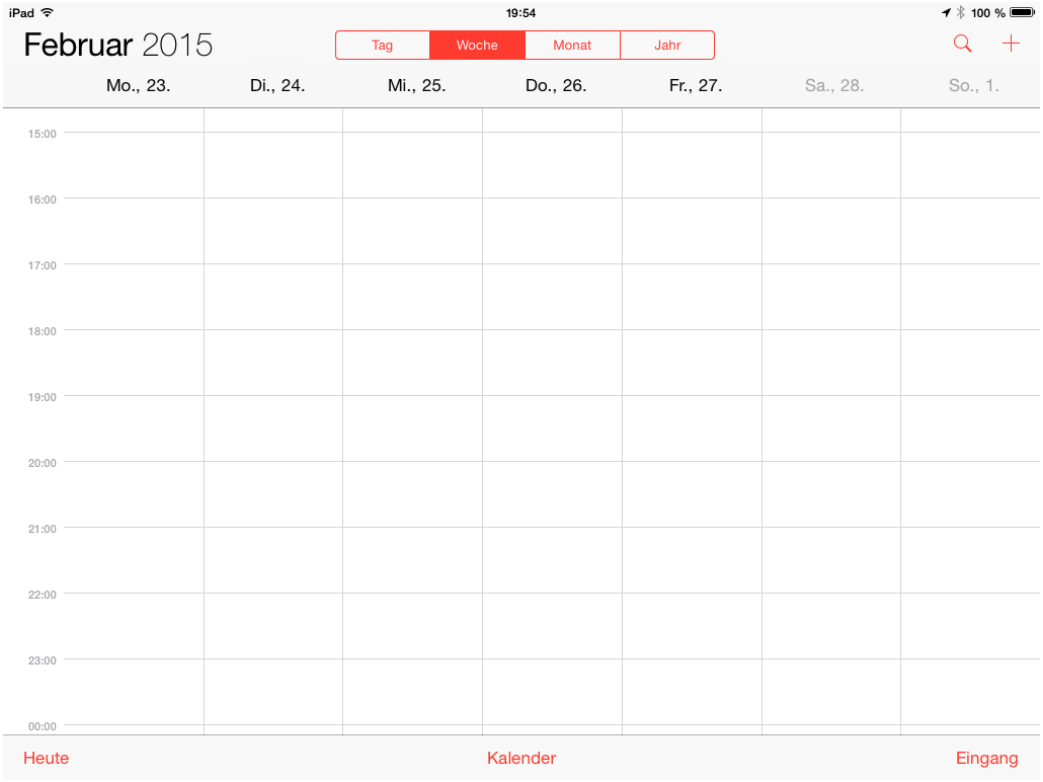


Abbildung 4: Flaches Design in iOS 8

Abbildung 4 zeigt wie dieser flache Designansatz das Design der Kalender-Applikation in iOS 8 verändert hat und soll als Beispiel für den Unterschied

zwischen flachem und skeuomorphem Design dienen. Das Design verzichtet weitgehend auf eine Anspielung auf einen realen Kalender und lässt somit auch die Materialstrukturen an den Rädern weg. Außerdem wurde auch der Mechanismus zum Interagieren mit der Applikation verändert. Während in der alten Version des Kalenders eine Bewegung ausgeführt wurde die analog zum Umblättern eines realen Kalenders ausgeführt wird, hat sich die Gestensteuerung seit diesem Zeitpunkt verändert hin zu einer rein digitalen Interaktion. Diese wurde hinsichtlich ihrer Charakteristika so verändert, dass sie für Touchscreens optimiert wurde und nicht länger mit einer analogen Interaktion zusammenpassen muss.

Besonders relevant für diesen Übergang zwischen skeuomorphem Design und flachem Design ist die Phase des Lernens für die Benutzer. Während bei der Erscheinung des ersten iPhones nur die wenigsten Nutzer Erfahrung in der Interaktion mit Touchscreens hatten, haben Touchgeräte mittlerweile eine sehr große Verbreitung und der Umgang mit den Geräten ist für die meisten User durch die tägliche Interaktion vertraut. Deshalb ist es auch nicht mehr länger notwendig die digitale Touchinteraktion so stark mit der realen Welt zu vernetzen, sondern es ist den Entwicklern möglich eigene Methoden zu finden um die Touch-Interaktion zu optimieren. Die grundlegende Interaktion ist wohl mittlerweile als gelernte, kulturelle Konvention ähnlich der Scrollbar auf Desktoprechnern, zu bezeichnen. Eine solche kulturelle Konvention zeichnet aus, dass man von den Benutzern ein grundlegendes Verständnis erwarten kann und ein solches Interkationsmuster auch nicht mehr verlernt wird - was bedeutet, dass es möglich ist die visuellen Hilfestellungen der früheren Generationen zu entfernen, ohne die Interaktionsmodelle zu schädigen. [16]

Allerdings ist es laut dem amerikanischen Psychologen J. J. Gibson notwendig in der Gestaltung von User Interfaces mit Affordances zu arbeiten. Diese Affordances sind bestimmte Mechanismen die dem Benutzer veranschaulichen was von ihm erwartet wird. Zum Beispiel veranschaulicht der Griff an einer Tasse, dass vom Benutzer erwartet wird, dass er die Tasse am Griff nimmt. Nun müssen auch digitale User-Interfaces solche Affordances besitzen. Allerdings ist es nicht notwendig bzw. in vielen Fällen auf Grund der Limitierungen sogar eher hinderlich, dass diese Affordances durch Skeuomorphismus angeboten werden. Kapitel 4 beschäftigt sich näher mit verbreiteten Pattern, die sich im mobilen Design etabliert haben und versuchen diese Affordances mit Methoden des flachen Designs anzubieten. [17] [16]

4 Best Practices für Mobile Geräte

Durch die Frameworks, die für native Anwendungen für die mobilen Betriebssysteme zahlreiche Designelemente vorgeben, haben sich Best Practices etabliert, die von den Systemherstellern etabliert wurden, und durch die nativen Anwendungen starke Verbreitung finden. Das bedeutet, dass gewisse Praktiken, Methoden und Designs den Benutzern vertraut sind und es Sinn macht diese aufzugreifen, da sie erprobt sind und häufig verwendet werden.

In diesem Kapitel wird versucht die Gemeinsamkeiten der Betriebssystem-Hersteller zu finden, und außerdem weitere erprobte Designmethoden zu besprechen, die zusammen eine Ansammlung an etablierten Elementen und Herangehensweisen bilden sollen, die beim Erstellen einer mobilen Applikation hilfreich sind und unter anderem beim später vorgestelltem Anwendungsprojekt verwendet wurden.

4.1 Allgemeines

Im Gegensatz zu Desktop-Browsern, die durch die permanente Darstellung der URL-Leiste eine dauerhafte Orientierungshilfe bieten, die es dem User erlaubt sich innerhalb der Seitenstruktur zu orientieren, wird diese URL-Leiste auf mobilen Geräten in der Regel nicht ständig angezeigt (native Applikationen verzichten ebenfalls auf eine solche permanente Navigationshilfe). Deshalb ist es wichtig im mobilen Interface-Design dem Benutzer manuell eine solche Hilfe anzubieten. Geschehen kann dies zum Beispiel über aussage-

kräftige Titel, für einzelne Anwendungsbereiche, die an einem prominenten Platz (wie zum Beispiel der oberen Navigationsleiste einer Anwendung) dargestellt werden. [18]

Neben der limiterten Darstellungsfläche ist auch die Fähigkeit zur Text-Eingabe limitiert. Besonders zu beachten ist dies wenn es für den Benutzer relevant sein könnte für die generelle Navigation der Anwendung auf eine Keyboard-Eingabe angewiesen zu sein. Genau wie für das allgemeine Webdesign gilt deshalb auch für mobile Geräte das Prinzip, dass es stets einfacher ist den Benutzer über klickbare Elemente weiter zu verweisen, als von ihm eine Eingabe jeglicher Art zu erfordern. Diese Eingaben sollten stets nur gefordert werden, wenn es keine andere Lösung gibt (z.B. Eingabe des Namens, Passworts, etc.). Allerdings ist die Minimierung der Texteingabe auch zu beachten, wenn URLs für die Applikation gestaltet werden. So ist es hilfreich wenn diese so kurz und aussagekräftig wie möglich ausfallen, da sowohl Eingabe, als auch Darstellung in der URL Leiste limitiert sind. [18]

Der W3C [19] empfiehlt in seinen „Mobile Web Best Practices 1.0 “außerdem bei der Entwicklung einer Anwendung für mobile Geräte den Umstand der mobilen Datenübertragung zu beachten. Konkret bedeutet dies, dass mobile Geräte häufig nicht aus einem WLAN oder anderem stationären Netzwerk auf das Internet zugreifen, sondern über Mobilfunknetzwerke. Diese sind hinsichtlich ihrer Bandbreite und Datenbegrenzung in der Regel limitierter. Für Entwickler bedeutet dies, dass es notwendig ist die über das Internet angeforderten Resourcen so klein wie möglich zu halten. Dies kann allerdings häufig schwierig sein, da moderne Smartphones über eine sehr hohe Pixeldichte verfügen und z.B. die User-Interface-Guidlines von Apple vorschla-

gen diese Pixeldichte auch zu nützen indem man hoch Auflösende Grafiken zur Verfügung stellt. Als Lösung wird vorgeschlagen z.B. niedrig auflösende Standard-Grafiken zu verwenden und speziell für Displays mit höheren Pixeldichten Fallback-Grafiken anzugeben. Dies lässt sich über spezielle Meta-Tags realisieren. Eine andere Lösung ist es z.B. für Icons vektorbasierte Grafiken, oder Font-basierte Frameworks wie Font-Awesome zu verwenden, da diese skalierbar sind und auf allen Bildschirmtypen gut gerendert werden. [15]

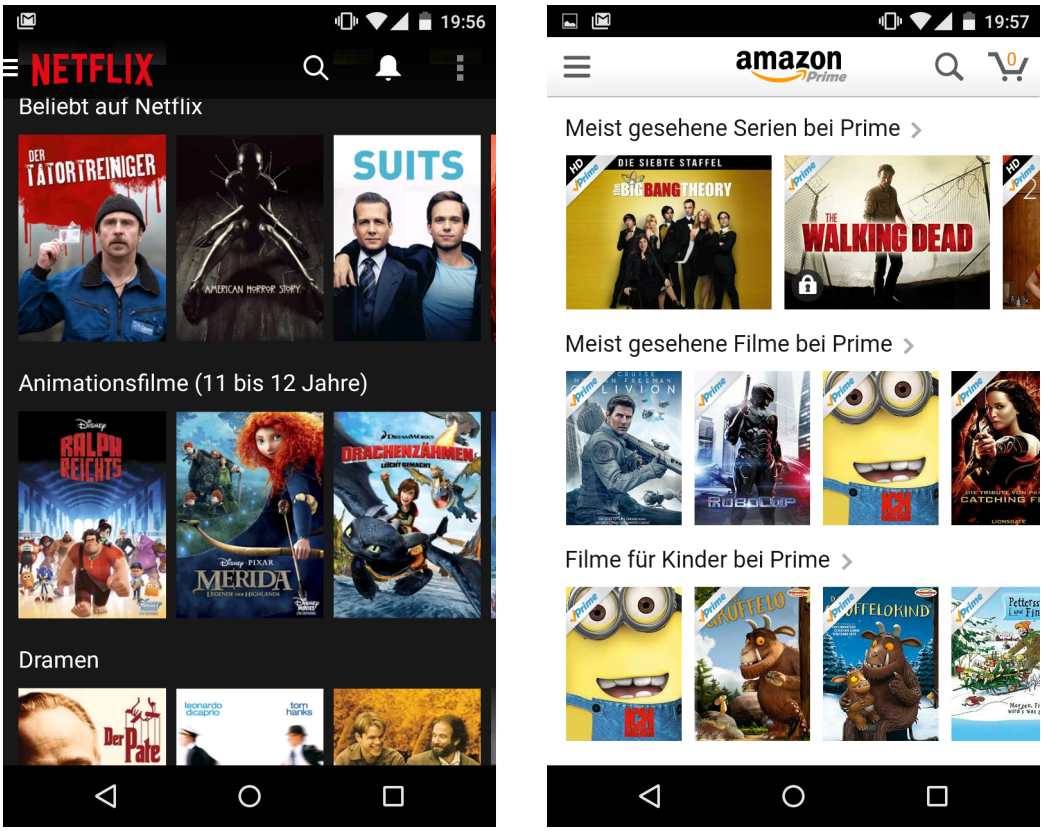


Abbildung 5: Native mobile Anwendungen von Netflix und Amazon Instant Video

Hinsichtlich des generellen Seitenlayouts ist im Vergleich zu Desktopapplikationen und Webseiten ein Umdenken nötig. Während es am Desktop mit zu-

nehmenden Bildschirmgrößen einen Trend zu Spaltenlayouts gab (und gibt) ist ein solches paralleles Layout auf mobilen Geräten auf Grund der limitierten Bildschirmgröße nicht, oder nur eingeschränkt möglich. Deshalb muss man horizontale Designs entweder umgestalten oder linearisieren. Versucht man sie umzugestalten, so ist es möglich die Inhalte in einem Slider unterzubringen, der durchgescrollt werden kann. Versucht man die Inhalte zu linearisieren, so ordnet man Inhalte, die in der Desktop Version parallel wären vertikal an. Es ist auch möglich beide Ansätze zu kombinieren, um z.B. Inhalte von verschiedenen Kategorien untereinander anzuzeigen. Dieser Ansatz ist in mobilen Anwendungen häufig gesehen und wird von den Frameworks der verschiedenen Betriebssystemen nativ durch eigene UI-Elemente unterstützt. Abbildung 5 zeigt die Anwendung dieses Prinzips innerhalb der nativen Android Anwendungen von Netflix und Amazon Instant Video. Die Anwendungen listen einzelne Blöcke von Genres untereinander auf, lassen den Benutzer aber dennoch horizontal durch die Listen scrollen. [18]

Ein weiteres Grundprinzip des mobilen Anwendungsdesign ist es, falls es eine dazugehörige Desktopanwendung gibt, sich inhaltlich möglichst nahe an der Vorlage zu orientieren und nur die Art der Aufbereitung zu verändern. Funktionelle Einschränkungen sind zwar möglich, falls es dazu gravierende Gründe gibt, wie mangelnde technische Unterstützung von mobilen Geräten, allerdings sollte als Grundprinzip dennoch gelten den Inhalt und auch grundlegende Funktionen gleich zu belassen. Allerdings ist die Priorisierung von Inhalten noch wichtiger, als bei Desktop-Applikationen. Dies geschieht durch Zentrierung, Größe und Abgrenzung des Hauptinhalts eines Bereichs zu den Navigationselementen. Außerdem ist es notwendig, dass der Inhalt einer Applikation visuell dominiert und nicht der Hauptfokus des Benutzers auf die

Navigationselemente fällt. Populäre Designpatterns um dies zu realisieren sind Navigationsleisten an den Rändern der Appliaktion, da so eine organische Grenze zwischen der Navigation und dem Inhalt entsteht und auch gewährleistet ist, dass der Inhalt stets den zentralen Bereich der Seite einnimmt. [18]

4.2 Navigation

Wie bereits im vorigen Kapitel beschrieben, sollen Texteingaben auf mobilen Geräten so oft wie möglich vermieden werden. Touchscreen Geräte ermöglichen es den Benutzern Anwendugen mit ihren Gesten zu steuern und da dies im Vergleich zu einer Maussteuerung eine Abstraktionsebene aus der Bedienung entfernt, ist es auch das Ziel diese Form der direkten gestengesteuerten Interaktion zu forcieren. So sollen sich die Navigationselemente auch durch berühren, wischen, ziehen, oder anderen gestenbasierten Navigationsformen bedienen lassen. Laut den User-Interface Guides von Google und Apple wird empfohlen sich an die etablierten Gesten zu halten, die den Anwendern bekannt sind. Diese Gesten sind: Tap (Klicken), Drag (Ziehen), Flick (Kurzes Klicken und wischen), Swipe (horizontales Wischen), Double Tap (Doppelclick), Pinch (Zwei Finger zusammenführen oder auseinanderbewegen), Touch and Hold (Klicken und Halten) und Shake (das Gerät schütteln). [15] [3]

Eine generelle Empfehlung für mobile Navigationselemente ist es die Strukturen zu vereinfachen. Beispielsweise soll die Anzahl an Hauptnavigationspunkten nicht mehr als zehn Elemente betragen. Allerdings gibt es auch Methoden, die es erlauben diesen Wert zu überschreiten. So ist es möglich

hierarchische Navigationen zu bilden und Unterpunkte nur anzuzeigen wenn der User den Hauptpunkt angewählt hat, oder zumindestens die Möglichkeit zu implementieren die Navigationespunkte ein- und auszuklappen. Der W3C empfiehlt außerdem, die Hauptnavigation in Form einer Toolbar am oberen Bildschirmbereich zu realisieren, da sich diese Lösung als quasi-Standard etabliert hat und von den Usern so erwartet wird. Da diese Topbar in der Regel nicht ausreicht um die vollständige Navigation zu beinhalten, wird sie in der Regel mit Buttons angereichert, deren Aufgabe es ist die Navigation anzuzeigen. Dies kann enweder als Dropdown von oben oder als Fade-in von den Seiten erfolgen. In Abbildung 5 ist zu sehen, dass sowohl Netflix, als auch Amazon dieses Navigationspattern in ihren nativen Android-Applikationen anwenden. Falls die Anwendung einen weiteren Bedarf an Navigationselementen hat, wird eine Implementierung am unteren Bildschirmbereich vorgeschlagen. Zentraler Aspekt ist jedoch, dass die Navigation auf das Wesentliche reduziert wird und der User beim Aufrufen der Seite nicht scrollen muss um den Hauptinhalt der Seite zu sehen. [18] [19]

Abbildung 6 zeigt die Best Practices, die Google für Android Applikationen vorschlägt, die aber auch für alle anderen mobilen Anwendungen als Vorbild dienen können, da sie eine einfache, generelle Herangehensweise an einen optimierten Benutzerfluss für mobile Geräte liefert. Unterschieden wird hierbei in Top Level Views, Category Views und Detail/Edit Views, die eine dreistufige Hierarchie bilden und den User durch die Anwendung führen sollen. Die Top Level Views bilden dabei eine Form der Überblick Darstellung, um dem User zu zeigen welche Unterkategorien vorhanden sind. Je nach Applikationsdesign ist dies entweder als Raster, Liste, oder auch als Tab-Auflistung der direkten Category Views möglich. Die Frameworks der Betriebssysteme

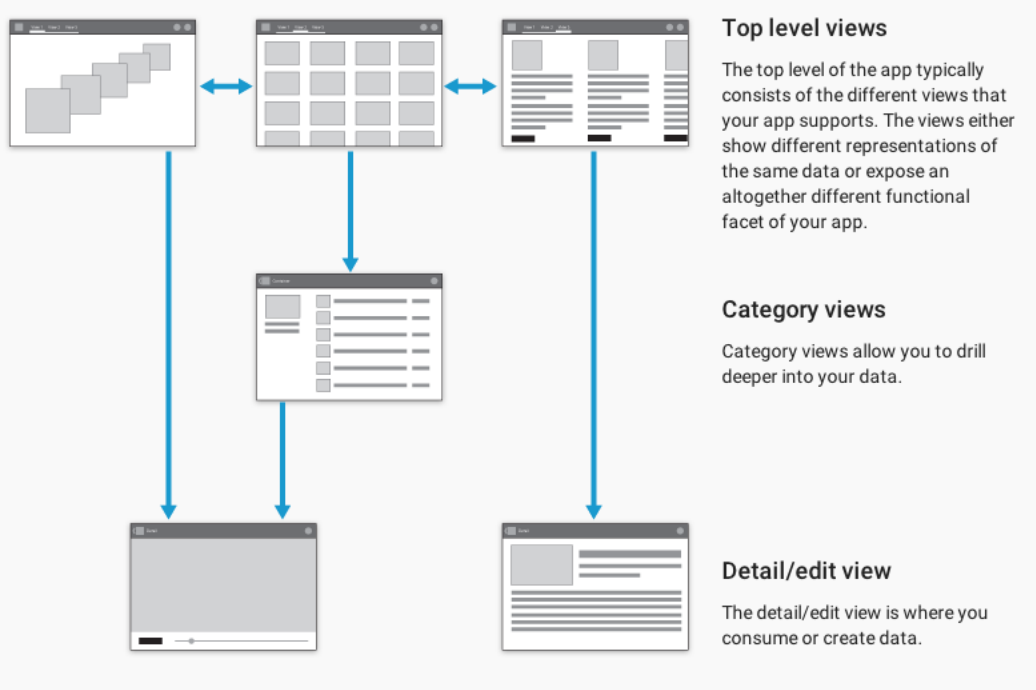


Abbildung 6: Viewdesign in Android [3]

bieten dabei fertige Komponenten um solche optimierten views automatisch zu erstellen, während für Web-Anwendungen entweder individuelle Lösungen programmiert werden müssen, oder vorhandene Frameworks wie Bootstrap genutzt werden können. Die Category Views sind weitere Unterlisten, die die Inhalte der jeweiligen Kategorien auflisten. Auch die Category Views können als Raster, Liste, oder in anderer Form implementiert werden. Von den Category Views wird von den einzelnen Elementen weitergeleitet auf die Detail/Edit Views. In diesen findet die eigentliche Inhaltsdarstellung statt, die je nach Anwendungstypus stark variiern kann. Dabei kann es sich um Videos (Youtube, Netflix), E-Mails (G-Mail), Zeitungsartikel (DerStandard), oder jede andere Inhaltsform handeln. Dabei gilt das Prinzip der minimalen View-Anzahl. Dies bedeutet, dass der Entwickler so wenige views wie möglich verwenden soll um den User übersichtlich zum Inhalt zu führen. Durch Ein-

haltung dieser Regel ist garantiert, dass der User in maximal drei Klicks bei einem relevanten Detail View angelangt und Inhalte konsumieren oder erstellen kann.

4.3 Layouts

Die meisten mobilen Anwendungen werden in irgendeiner Form auf die Hierarchie aus Category View und Detail View zugreifen, die zuvor beschrieben wurde. Dabei haben sich speziell für die Darstellung der Category Views einige Patterns ergeben, die den Entwicklern eine Orientierungshilfe geben um eine solche Auflistung darzustellen.

Abbildung 7 zeigt die drei gängigen Layout-Container die in Android und iOS zum Einsatz kommen. Das Tab-Layout bildet dabei eine Art Über-Container in dem sowohl Listen- und Grid-Layouts, als auch Detail Views eingebettet werden können. Durch eine Swipe-Bewegung, oder durch Klick auf das jeweilige Element ist es dem User dabei möglich zwischen den Tabs umzuschalten und die verschiedenen Unter-Views zu aktivieren. Ein List-Layout besteht aus einer einfachen, vertikalen Auflistung in der ein Element je eine Zeile der Liste bildet. Dabei ist allerdings zu beachten, dass unter Zeile der klickbare Bereich innerhalb der Liste verstanden wird. Es ist also durchaus möglich, dass ein Listenelement individuell gestaltet wird und zum Beispiel mit einem Bild versehen wird und mehr als eine Zeile Text enthält. Das Grid-Layout ist eine Tabelle, deren Spaltenanzahl frei gewählt werden kann. Die Anzahl der Zeilen wird automatisch durch den Inhalt bestimmt, der dem Container zugewiesen wird. [3]

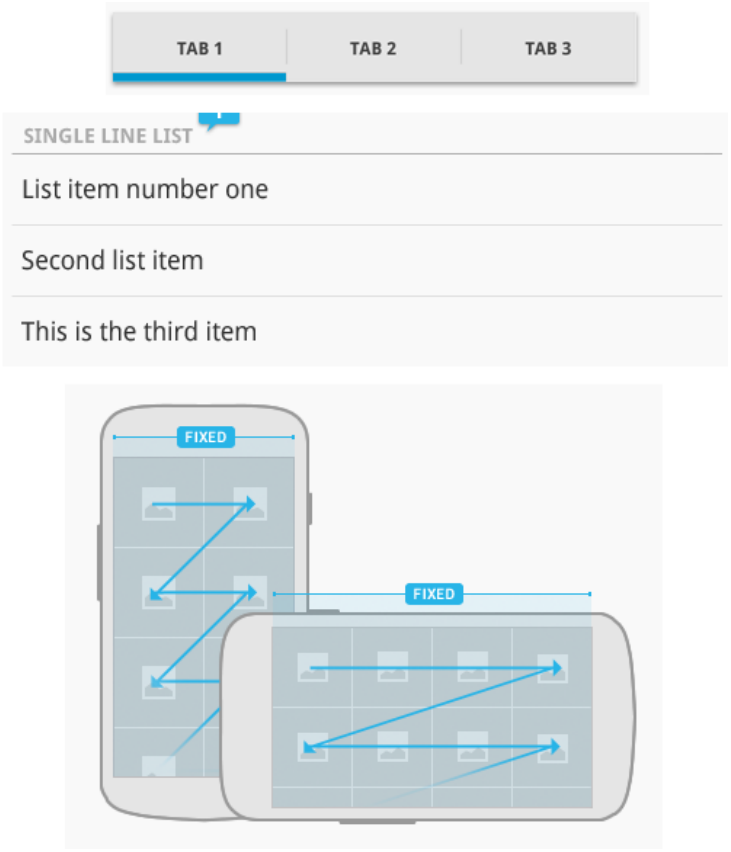


Abbildung 7: Tab-, List- und Grid-Layouts in Android [3]

Eine weitere wesentliche Designkomponente der Layout-Gestaltung für mobile Geräte ist die context-sensitve Layoutanpassung. Darunter versteht man die Berücksichtigung von unterschiedlichen Displaygrößen. Da z.B. Tablets deutlich größere Displays als Smartphones besitzen, haben sie auch andere Möglichkeiten in der Anwendungsdarstellung und die Entwickler sollten diesen Umstand im Design der Anwendung berücksichtigen. Dies ist in den nativen Applikationen durch verschiedene Layouttechniken und für Web-Anwendungen durch Responsive Design möglich.

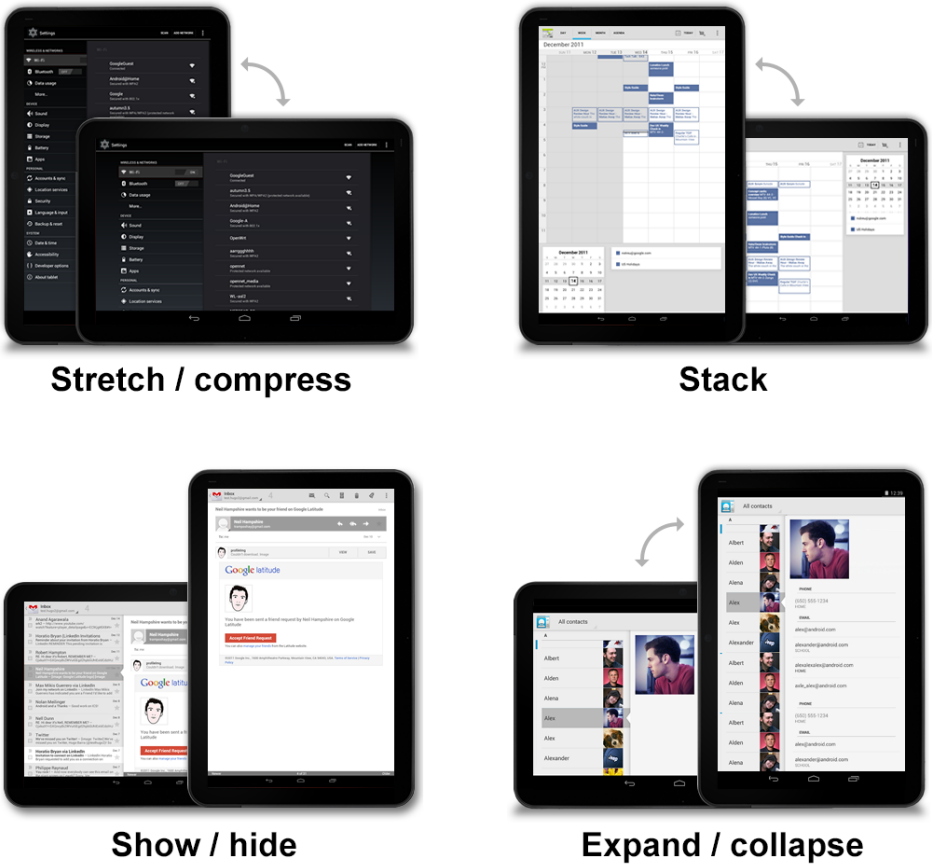


Abbildung 8: Multi-pane Layouts in Android [3]

Die diesbezüglich möglichen Anpassungsszenarien werden in Abbildung 8 gezeigt. Die Technik Stretch / Compress nimmt lediglich die vorhandenen Layout-Strukturen und passt die Breite der existierenden Container an die Breite des Geräts an. So werden vorhandene Spalten entweder verbreitert (stretch) oder verschmälert (compress). Die Stack Methodik ordnet die vorhandenen Layout-Strukturen je nach vorhandenem Platz um. So können Elemente entweder horizontal oder vertikal angeordnet werden (bleiben aber stets sichtbar). Die Show/Hide Methode ist die komplexeste Methode die zur Verfügung steht. Je nach verfügbaren Platz werden bei diesem Ansatz Elemente ein- oder ausgeblendet. Der Grund warum diese Methode die kom-

plexeste ist, ist dass die Methode als einzige in den Navigationsverlauf der Anwendung eingreift. So ist es mit diesem Ansatz zum Beispiel möglich eine Applikation zu entwickeln die auf Tablets links eine Liste an Elementen anzeigt und rechts gleich den ausgewählten Detail View darstellt. Auf Smartphones würde die selbe Navigation allerdings einen sequentiellen Navigationsfluss verwenden, der zuerst die Liste anzeigt und nach einem Klick den Detail View seperat rendert. So ist es nötig verschiedene Nutzerszenarien je nach Layout programmatisch abzudecken. Der letzte mögliche Ansatz ist ein Expand / Collapse Ansatz. Dieser verhält sich prinzipiell ähnlich wie die Show / Hide Methode, allerdings wird in diesem Fall nicht in die Navigationsstruktur eingegriffen. Stattdessen werden nur zusätzliche Informationen des Detail View ein- oder ausgeblendet.

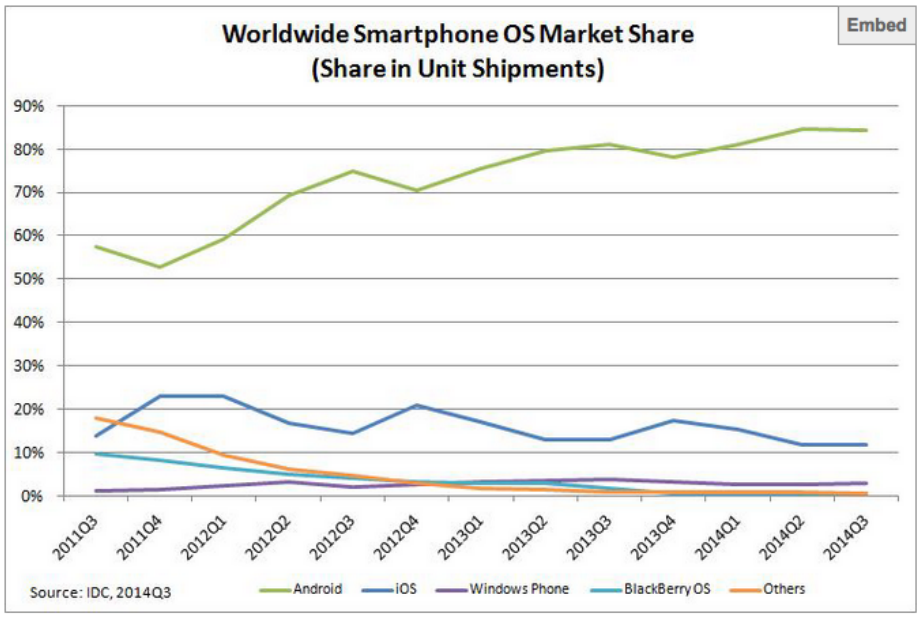
5 Adaption von Medienangeboten für Mobile Geräte

Um ein existierendes Medienangebot zu adaptieren gibt es unterschiedliche Methoden, die sich hinsichtlich ihrer Vor- und Nachteile unterscheiden. Diese Vor- und Nachteile können etwas abweichen wenn es nicht notwendig ist auf ein bestehendes Angebot Rücksicht zu nehmen und stattdessen ein neues Angebot erstellt wird. Im Folgenden soll auf die unterschiedlichen Möglichkeiten eingegangen werden. Dabei wird der Fokus auf die Adaption eines existierenden Portals gelegt, da dies der Aufgabenstellung dieser Arbeit entspricht, es soll aber zumindestens rudimentär auf die Ausgangslage eingegangen werden, wenn kein Legacy-System beachtet werden muss. Um das Szenario weiter einzuschränken, wird sich die Kategorisierung vornehmlich auf Web-Services beschränken, da bei einer reinen Offline-Anwendung viele Einschränkungen und Überlegungen nicht anwendbar sind.

5.1 Native Applikationen

Native Applikationen werden für eine Ziel-Plattform entwickelt und sind nur auf dieser lauffähig. Laut aktuellen Zahlen besitzt Android einen weltweiten Marktanteil von 84,4%, iOS von 11,7% und Windows Phone von 2,9% (Vgl. Abbildung 9). Das bedeutet, dass für eine umfassende Marktabdeckung von 99% zumindestens drei Applikation erstellt werden müssen. [4] Abhängig von der konkreten Zielgruppe, länderspezifischen Änderungen in der Systemverteilung, oder der Bereitschaft auf einen Teil der Zielgruppe zu verzichten,

steht es den Entwicklern zwar frei auf eine, oder mehrere der Systeme bewusst zu verzichten, aber da dieser Verzicht zwangsläufig mit einer Einschränkung der Zielgruppe einher geht, ist es keinesfalls als optimales Szenario zu betrachten, sondern macht lediglich für spezielle Anwendungen Sinn (als Beispiel sei eine App erwähnt, die als Ausgabenüberwachung für iTunes dient, und die dementsprechend auch nur auf einer Plattform entwickelt werden muss).



Period	Android	iOS	Windows Phone	BlackBerry OS	Others
Q3 2014	84.4%	11.7%	2.9%	0.5%	0.6%
Q3 2013	81.2%	12.8%	3.6%	1.7%	0.6%
Q3 2012	74.9%	14.4%	2.0%	4.1%	4.5%
Q3 2011	57.4%	13.8%	1.2%	9.6%	18.0%

Source: IDC, 2014 Q3

Abbildung 9: Weltweite Marktanteile an Smartphone Betriebssystemen [4]

Durch die Anforderung drei spezifische Apps entwickeln zu müssen, ergeben sich bereits die großen Nachteile dieses Ansatzes. Da jede der drei Plattformen auf gravierend unterschiedliche Programmiersprachen, Frameworks und Entwicklungsumgebungen setzen, ist auch bei der Besetzung der entsprechenden Programmierer auf eine dementsprechende Heterogenität zu achten. So erfordert die Programmierung für iOS Geräte eine Mac-Arbeitsumgebung, die Programmiersprache Objective-C und einen Entwickler der sowohl mit der IDE XCode, als auch mit dem iOS Framework vertraut ist. [20] Für die Android-Entwicklung benötigt der Entwickler eine Mac-, Windows-, oder Linux-Arbeitsumgebung, Java-Kenntnisse und Erfahrung mit der IDE Android Studio und dem Android Framework. [21] Entwicklung für Windows Phone erfordert einen Windows PC, C# oder C++ Kenntnisse und Erfahrung mit dem Windows Phone Framework, sowie der IDE Visual Studio. [22]

Um nun eine Anwendung zu erstellen, die auf den drei marktstärksten Betriebssystemen lauffähig ist, muss dafür gesorgt werden, dass verantwortliche Entwickler vertraut mit allen drei Programmiersprachen und den dazugehörigen Frameworks sind und außerdem die Arbeitsumgebungen und Lizenzen vorhanden sind um eine Entwicklung für alle Betriebssysteme zu ermöglichen. Dies bewirkt aus Sicht des Projektmanagements mehrere Hürden. Da es durch die drei verschiedenen Umgebungen keine Überschneidungen gibt, müssen entweder die dreifache Menge an Entwickler engagiert werden oder die dreifache Zeit eingeplant werden um alle Umgebungen zu versorgen. Hinzu kommen Ausgaben und Wartungskosten für die verschiedenen Betriebssysteme und die entsprechenden Administrationsaufwände (um eine Entwicklung für alle drei Betriebssysteme zu ermöglichen sind zumindestens Windows und Mac OS-Systeme erforderlich). Die administrativen Probleme

der nativen App-Entwicklung enden allerdings nicht mit der Fertigstellung der App. Die selben Probleme gelten auch für die Weiterentwicklung und da die APIs der großen Hersteller einer stetigen Weiterentwicklung unterworfen sind, ist dies auch kein einmaliger, sondern ein sich stetig wiederholender Prozess. All diese Punkte führen dazu, dass die Entwicklung von nativen Anwendungen in der Regel die aufwändigste und teuerste Form der mobilen Entwicklung darstellt. Dass dieses Problem nicht nur ein Faktor für kleine Unternehmen ist, zeigt die Tatsache, dass selbst Google für die mobile Strategie des Konzerns nicht ausschließlich auf rein native Anwendungen setzen konnte, weil es finanziell nicht rentabel ist. [23]

Allerdings bedeutet dies nicht, dass native Anwendungen keine Berechtigung mehr haben. Trotz Technologien wie HTML 5 ist der Offline-Support von nativen Anwendungen immer noch einfacher und stabiler zu erreichen als mit Web-Anwendungen. Der wahrscheinlich größte Vorteil von nativen Anwendungen ist der Performance-Vorsprung gegenüber Web-Anwendungen. Dieser entsteht auf Grund der Tatsache, dass Web-Anwendungen während der Laufzeit interpretiert werden müssen, während native Anwendungen kompiliert werden. Zwar haben es sich die Browserhersteller zur Aufgabe gemacht die Interpretationsgewschwindigkeit von JavaScript innerhalb der Browser, und in weiterer Folge auch innerhalb von WebView Bereichen in Anwendungen zu verbessern, allerdings gibt es zum aktuellen Zeitpunkt noch große Unterschiede bei rechenintensiven Aufgaben. Zwar ist dies im Kontext vieler Anwendungen vernachlässigbar, allerdings werden komplexe 3D-Berechnungen wie sie z.B. bei Spielen oder sehr aufwändigen Visualisierungen vorgenommen werden, in vielen Fällen immer noch nicht umsetzbar sein im Rahmen einer Web-Anwendung. [23]

Auch aus wirtschaftlicher Sicht gibt es (neben dem Nachteil durch höhere Entwicklungskosten) einige Vorteile, die es vereinfachen können Geld mit der Anwendung zu verdienen. Eine dieser Vorteile ist die zusätzliche Werbefläche die native Apps in Form des App Stores (iOS), Play Stores (Android) und Windows Stores (Windows Phone) bekommen. Diese zentralen Download-Umschlagsplätze bieten für die Benutzer der verschiedenen Systeme eine zentrale Anlaufstelle. Entsprechend ist es durch die Download-Charts, auf Apps optimierte Suchfunktionen, Genre-Listen, und ähnliche Empfehlungssysteme für die Benutzer einfacher passende und neue Apps zu finden. Weiters bieten die Betriebssysteme auch die Möglichkeit direkt Geld von den Benutzern zu verlangen. Dies kann entweder direkt als Kaufpreis für die Anwendung erfolgen oder weiterführend durch In-App-Verkäufe, die es ermöglichen für bestimmte Inhalte Geld vom User zu verlangen. Zwar ist es auch möglich auf Webseiten Geld zu verlangen, allerdings sind die dort vorhanden Dienste deutlich fragmentierter und weniger einfach zu benutzen, während es für native Anwendungen standardisierte Möglichkeiten gibt, die für alle Anwendungen gleich sind und direkt über den Betriebssystem-spezifischen User-Account abgebucht werden, der auch mit Gutscheinkarten aufgeladen werden kann. Hinzu kommt, dass Bezahlsysteme im Allgemeinen für Web-Anwendungen unüblicher und weniger akzeptiert sind, als für native Anwendungen. [24]

Allerdings gibt es auch für den an sich positiven Punkt des Store-Systems einen dazugehörigen Negativpunkt, der in vielen Szenarien überwiegen kann. Denn Web-Anwendungen können von der gesamten Vernetztheit und offenen Zugänglichkeit des Internets profitieren, während das Store-System dazu

geführt hat, dass die eigenständigen Betriebssysteme verriegelte Container bilden, aus denen die Benutzer nicht ausbrechen können. So können Websiten sich untereinander verlinken, jeder hat die Möglichkeit eigene Web-Anwendungen zu erstellen und diese sind auch für jeden über Suchmaschinen auffindbar. Für native Anwendungen gibt es kein solches System. Zwar erlaubt Android über Umwege die Installation am Store vorbei, aber in der Regel ist der Benutzer an das Angebot des Stores gebunden, externe Suchmaschinen sind nur schwer realisierbar (besonders wenn eine Indexierung über die App an sich hinausgehen soll und z.B. spezifische Unterseiten gefunden werden wollen) und eine Verlinkung ist nur sehr limitiert möglich, da eine App nicht automatisch davon ausgehen kann, dass ein Benutzer auch eine andere App installiert hat auf die man möglicherweise verlinken möchte, während in Web-Anwendungen eine Verlinkung auf alle anderen Web-Anwendungen stets möglich ist. [24]

Vorteile besitzen native Applikationen hingegen in der Konsistenz des User-Interface-Designs. Sowohl iOS, als auch Android und Windows Phone besitzen UI-Guidlines und Frameworks, die teilweise deutlich unterschiedlich sind. Daraus folgt, dass eine einheitlich gestaltete Web-App unmöglich den gesamten Guidlines und Best Practices aller drei Systeme entsprechen kann, weil sie teilweise radikal unterschiedlichen Look & Feel Modelle folgen was das konkrete Design von Elementen wie Buttons betrifft. Im Rahmen einer Web-Anwendungen ist es aber nur sehr schwierig, unzuverlässig und unter stark erhöhtem Merhaufwand möglich ein System zu designen, das sich an das System anpasst und die jeweiligen Guidlines befolgt (dies würde außerdem dem Prinzip einer Web-Anwendungen entgegenlaufen und zahlreiche Vorteile des Ansatzes zunichte machen). Dementsprechend muss man sich beim Ge-

stalten einer Web-Anwendung auf eine Design-Sprache einigen, die entweder den Guidlines eines Anbieters folgen kann, oder eine vollständig eigenstädige Methode wählt. In beiden Fällen würden sich aber deutliche Design-Brüche im Vergleich zu den systeminternen Anwendungen und anderen nativen Anwendungen ergeben, die dazu führen, dass die Anwendung sich weniger konsistent in das System einfügt. Programmiert man eine native Applikation hat man dieses Problem nicht, da die IDEs der einzelnen Systeme den Entwickler bei der Auswahl der zu verwendenden User-Interface-Elemente unterstützt und für alle Anwendugsszenarien Bausteine verfügbar sind, die gewährleisten, dass die App sich nahtlos in das Ökosystem des jeweiligen Anbieters einfügt und ein einheitliches Bild mit anderen Anwendungen entsteht. [23]

Weiterführend besitzt auch die gesamte User-Experience von nativen Anwendungen potentielle Vorteile gegenüber Web-Anwendungen. Besonders relevant ist dies wenn die Anwendung durch gewisse Ereignisse regelmäßig die Aufmerksamkeit des Benutzers anfordern möchte. Als Beispiel sei ein E-Mail Client erwähnt, der bei jeder eingehenden Mail den Benutzer benachrichtigen möchte, dass die Anwendung seine Aufmerksamkeit benötigt. Im Rahmen einer nativen Anwendung lassen sich solche Push-Benachrichtigungen mehr oder weniger einfach mittels des jeweiligen Frameworks realisieren, während es für Web-Anwendungen keine Möglichkeit gibt dem Benutzer solche Push-Benachrichtungen (ohne Umwege wie z.B. Mails, die dann über die Push-Funktion des Mail-Clients eine Benachrichtigung auslösen) direkt auf das mobile Gerät zu senden. [23]

Ein weiterer Faktor, der die User-Experience von nativen Applikationen potentiell verbessert ist, dass die nativen Anwendungen über die Betriebssystem-Frameworks einen abstrahierten, vollständigen Zugriff auf alle Sensorendaten und besonderen Hardwarefunktionen der Geräte verfügen. Dazu zählen unter anderem GPS-Daten, Beschleunigungssensor-Daten, Kamerafunktionen, Kompass, und je nach Gerätegeneration noch zahlreiche weitere Features. Will man als Anwendungs-Entwickler vollständig und zuverlässig auf diese Funktionen zugreifen, so kommt man um die Entwicklung einer nativen App kaum herum. Zwar wird daran gearbeitet gewisse Funktionen auch über das Web abzufragen (z.B. ist der Zugriff auf GPS Daten bereits möglich), allerdings ist dies noch nicht umfassend möglich, sodass dies aktuell noch ein klarer Vorteil für native Applikationen ist. [23]

Vorteile von nativen Applikationen

- Performance
- UI-Design
- User-Experience
- Direkter Zugriff auf Gerätefunktionen
- Vermarktbarkeit und Monetisierbarkeit

Nachteile von nativen Applikationen

- Entwicklungs-, Wartungs-, und Servicekosten
- Keine Code-Weiterverwendung
- Vorteile nicht für alle Szenarien relevant

5.2 Web Applikationen

Web Applikationen underscheiden sich von nativen Applikationen dadurch, dass sie nicht in den nativen Programmiersprachen der Betriebssysteme programmiert werden und nicht über die Stores downloadbar sind. Stattdessen werden sie in Form einer Webseite auf einem Server angeboten und sind über den Browser allgemein zugänglich. Das bedeutet auch, dass die Entwickler nicht an die Betriebssysteme, Programmiersprachen und Frameworks der jeweiligen mobilen Betriebssysteme gebunden sind, sondern frei wählen können welche Technologien zum Einsatz kommen sollen.

Abgesehen von den typischen Frontend-Technologien JavaScript und HTML, steht es somit völlig frei welche Technologien verwendet werden sollen. So ist es möglich das Backend mittels Apache und PHP, oder mittels Django und Python zu betreiben, oder auch jede beliebige, andere Kombination an Technologien zu verwenden. Für die Browser ist es nur relevant, dass letzten Endes HTML und JavaScript Elemente ausgegeben werden, wie dieser Code allerdings vom Backendsystem erzeugt wird, ist für die Browser (genau wie bei jeder anderen Webseite), egal. Bedeutung hat dies für die Entwickler vor allem deshalb, weil eine höhere Flexibilität bei der Technologiewahl unter anderem bedeutet, dass vorhandene Entwickler effizienter genutzt werden können und eine erhöhte Spezialisierung auf Kerntechnologien möglich ist, ohne dass es notwendig wäre auf fragmentierte Technologien von drei verschiedenen Betriebssystemen Rücksicht zu nehmen.

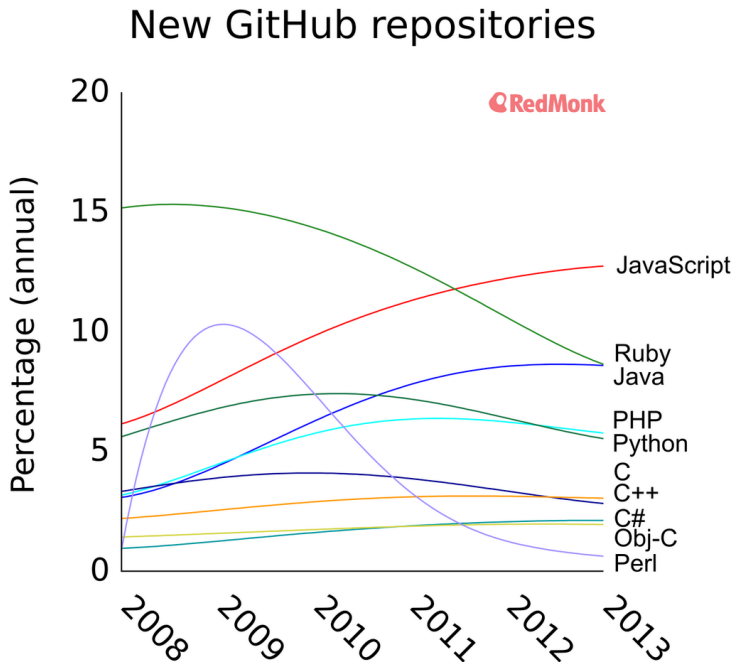


Abbildung 10: Neue Github Repositories nach verwendeter Programmiersprache [5]

Abbildung 10 zeigt die Programmiersprachen die bei neuen Repositories auf der Website Github zum Einsatz kamen. Relevant ist dies, da die Grafik veranschaulicht welche Programmiersprachen bei Entwicklern besonders beliebt sind und für welche Programmiersprachen es viele verfügbare, fertige Libraries gibt. Zwar bedeutet dies nicht zwangsläufig, dass es für die konkreten Projekte der Entwickler fertige Libraries gibt, allerdings zeigt die Grafik doch an wie hoch die Wahrscheinlichkeit ist für eine gegebene Programmiersprache eine fertige Library zu finden. Außerdem zeigt die Grafik wieviele aktive Programmierer und Projekte es für eine bestimmte Programmiersprache gibt. Dies ist relevant wenn man auf der Suche nach Entwicklern ist um sie für ein Projekt zu engagieren. Die Grafik hat allerdings auch die weiterführende Aussagekraft, dass die meist verwendeten Programmiersprachen, die in Github Repositories zum Einsatz kamen, eben gerade nicht die Programmierspra-

chen sind, die für native Applikationen verwendet werden müssen. Von den Top 5 Programmiersprachen befindet sich lediglich das für Android benötigte Java auf Rang 3, die anderen Plätze werden von JavaScript, Ruby, PHP und Python belegt - Programmiersprachen die nicht für native Applikationen verwendet werden können, allerdings sehr wohl für Web-Applikationen zum Einsatz kommen können.

Ein weiterer Vorteil ist, dass Web-Applikationen nicht für jede Plattform einzeln gestaltet und optimiert werden müssen, sondern allgemeinen Standards folgen, die von den Browsern beachtet werden und dementsprechend universell gültig sind. Natürlich kann es hinsichtlich der Unterstützung von einzelnen HTML und CSS Elementen Schwankungen zwischen den Browsern geben, aber die Unterstützung der allgemeinen Sprache ist auf allen Betriebssystemen gegeben. Das führt dazu, dass der Entwicklungsaufwand im Vergleich zu nativen Applikationen um 2/3 gesenkt wird, was entsprechend auch sowohl Entwicklungs- als auch Wartungskosten senkt.

Allerdings ergibt sich der Nachteil, dass die Performance der Anwendung nicht mit der Performance von nativen Applikationen mithalten kann. Die interpretierten Sprachen die für Web-Applikationen verwendet werden, können rechenintensive Aufgaben nur langsamer lösen, als die kompilierten Sprachen, die bei nativen Anwendungen zum Einsatz kommen. Wie bereits im vorigen Kapitel erwähnt, ist es auch schwieriger eine konsistente User Experience und ein entsprechendes User-Interface Design zu bieten. Zwar gibt es UI-Frameworks, die beim Design für mobile Geräte unterstützen, allerdings wird ein Unterschied zu den systemspezifischen Frameworks stets erkennbar sein. [24]

Auch ist es Web-Applikationen nicht möglich die Promotion- und Monetarisierungseffekte der anbieterspezifischen Stores auszunutzen. Umgekehrt steht es Web-Anwendungen allerdings offen die verlinkte und ungefilterte Struktur des Internets auszunutzen. So besitzen alle Stores Regeln und Aufnahmekriterien, die mehr oder weniger streng sind, während für Web-Anwendungen ausschließlich gültige Gesetze als Kriterium gelten ob etwas erlaubt ist, oder nicht. Entwickler begeben sich also nicht in Abhängigkeit gegenüber einem anderen Konzern. Das Risiko wegen verstößen gegen die Regeln aus dem Store entfernt zu werden ist somit nicht gegeben und dementsprechend gibt es auch keine diesbezüglichen Limitierung bei der Gestaltung der Anwendung.

Vorteile von Web-Applikationen

- Entwicklungs-, Wartungs-, und Servicekosten
- Code-Weiterverwendung
- Freie Wahl von Programmiersprache, Framework und Arbeitsumgebung
- Suchmaschinenoptimierung
- Einfache Verlinkung
- Keine Store-Richtlinien zu beachten

Nachteile von Web-Applikationen

- Performance
- Keine Promotion- und Monetarisierungsvorteile durch Stores
- UI und UX Konsistenz
- Wenige Device-APIs verwendbar

Während die Umsetzung bei nativen Anwendungen einer konkreten Struktur folgt, die an die Programmiersprachen, IDEs und Frameworks der jeweiligen Systeme gebunden ist, gilt es bei Web-Anwendungen zahleiche Entscheidungen zu treffen, wie die genaue Umsetzung der Anwendung erfolgen soll. Entscheidungen bezüglich des Backends, der zu verwendenden Programmiersprachen und ähnliches soll nicht Teil dieser Arbeit sein, da sie nicht direkten Bezug auf die Eigenschaften der Anwendung auf mobilen Geräten haben, sondern andere Paramter beeinflussen. Stattdessen sollen an dieser Stelle die zwei grundsätzlichen Möglichkeiten zum Design von mobilen Web-Applikationen beschrieben, sowie die Vor- und Nachteile des jeweiligen Ansatzes aufgezeigt werden.

5.2.1 Umsetzung durch getrennte Angebote

Bei diesem Ansatz wird das Web-Angebot für mobile Geräte und das Angebot für Desktop-User strikt getrennt. Realisiert wird dies z.B. durch zwei verschiedene Domainaufrufe. So kann die Standard-Domain unter www.domain.com erreichbar sein, während die mobile Domain unter m.domain.com erreichbar

ist. Dieser Ansatz bewirkt den Vorteil, dass für keinen Angebotstypus Kompromisse eingegangen werden müssen, stattdessen können die Stärken aller Systeme voll und ganz ausgespielt werden, da jeder Gerätetypus ein passendes Angebot bekommt. [18]

Der Nachteil an diesem Ansatz ist allerdings, dass zwei getrennte Web-Angebote betreut und gewartet werden müssen. Das bedeutet auch einen höheren Zeit- und Kostenaufwand. Hinzu kann es auch zu Problemen hinsichtlich der Suchmaschinenoptimierung führen, da Suchmaschinen wie Google eindeutige URLs für inhaltlichen Content bevorzugen und es im Falle einer dezidierten mobilen Website jeden inhaltlichen Content unter zwei verschiedenen URLs gibt, bei denen sich lediglich das Design, aber nicht der Inhalt verändert. [18]

Ein weiterer Nachteil dieses Ansatzes ist, dass durch die Trennung über die Domain harte Schnitte zwischen den Gerätekategorien gemacht werden. Häufig findet man eine Seite für Desktopnutzer und eine Seite für Smartphonenutzer. Das Problem daran ist allerdings, dass die tatsächliche Größe der Displays sehr stark variieren kann. So können Smartphones von 3.5 - 6 Zoll jede Größe haben und zwischen den Smartphones und Desktop-Rechnern haben sich auch noch Tablets etabliert, die ebenfalls als mobile Geräte gelten und deren Größe zwischen 6 und 12 Zoll schwanken kann. Gemäß dieser Verteilung ist es nicht möglich eine einzelne mobile Seite zu erstellen, die für alle Größen eine optimierte User Experience bietet. Außerdem ist es notwendig festzustellen mit welchem Gerät der User auf das Angebot zugreift und ihn entsprechend auf das jeweilige Portal umzuleiten. Da es stetig Weiterentwicklungen gibt und es für einen Entwickler so gut wie unmöglich ist ein Angebot

mit allen verfügbaren Smartphone- und Tabletmodellen zu testen, ergibt sich in dieser Funktion ein Unsicherheitsfaktor, sowie ein möglicherweise stetiger Wartungsaufwand, da es notwendige ist neue Geräteklassifikationen mit in die Unterscheidungsfunktion aufzunehmen. [18]

Falls allerdings ein sehr komplexes Legacy-Angebot existiert, kann es jedoch mitunter schwierig sein auf andere Art und Weise in die Struktur der Seite einzugreifen um sie für mobile Geräte verfügbar zu machen. In diesem Fall kann es für die Entwickler mitunter einfacher sein ein vollständig getrenntes mobiles Angebot zu entwickeln.

5.2.2 Umsetzung durch Responsive Design

Mit der wachsenden Diversifikation an Gerätekategorien, Auflösungen und Bildschirmgrößen von denen aus auf Web-Angebote zugegriffen wird, wird es auch immer schwieriger optimierten Content anzubieten. Diese Tatsache und der Umstand, dass Web-Browser die benötigten Technologien immer besser und umfassender unterstützen, hat dazu geführt, dass sich der Adaptions-Ansatz mittels Responsive Design immer mehr verbreitet hat und als der wohl vielversprechendste Ansatz für die Generierung von mobilen Anwendungen in sehr vielen Szenarien gilt. [25]

Traditionell entwickelte Websites weisen den einzelnen Modulen fixe Größen in Pixel zu. Zum Beispiel kann der äußere Container 1200px weit sein und sich aus zwei Spalten zusammensetzen, die 900px und 300px breit sind. Wird diese Website nun von einem Desktop-PC aufgerufen, der eine maximale Auflösung von 1600px in der Breite hat, so werden 400px der Website als Rand ohne Inhalt dargestellt. Wird die selbe Website von einem mobilen Gerät mit 600px

maximaler Breite aufgerufen, so wird die Website verkleinert, so dass sich der gesamte Inhalt darstellen lässt. Der Benutzer hat nun die Möglichkeit in einzelne Bereiche zu zoomen um so navigieren und lesen zu können. Dies führt dazu, dass sich im Prinzip auch nicht optimierte Websites auf einem mobilen Gerät darstellen und benutzen lassen, allerdings mindert die nicht vorhandene Optimierung die User-Experience. [26]

Ein großer Vorteil bei diesem Ansatz ist, dass es dem Designer möglich ist pixelgenau zu bestimmen welches Element wie groß an welcher Stelle erscheint. Beim Ansatz des Responsive Design wird darauf verzichtet pixelgenaue Entscheidungen zu treffen. Stattdessen wird der vorhandene Raum in Blöcke eingeteilt, welche relativ auf den vorhandenen Raum aufgeteilt werden. So entsteht ein flexibles Gitter am Bildschirm, das sich z.B. aus 12 Spalten und so vielen Zeilen wie möglich aufteilt. Nun ist es durch den Einsatz von CSS Media Queries möglich zu bestimmen wie viele Spalten ein gewisser Block bei welcher Displaygröße einnehmen soll. Ein einfaches Beispiel ist ein Media Query, der unterscheidet ob ein Display größer als 600px ist, oder nicht. Falls das Display größer als 600px ist, nimmt Block A eine Größe von 8 Spalten an, Block B eine Größe von 4 Spalten und die beiden Blöcke werden nebeneinander angezeigt. Ist das Display kleiner gleich 600px, sind beide Blöcke 12 Spalten groß und werden untereinander angezeigt. Der Designansatz des Responsive Designs ermöglicht es also fixe Größenangaben und Positionierungen zu vermeiden, stattdessen wird dem Designer so die Möglichkeit gegeben sich dynamisch am verfügbaren Platz zu orientieren und Elemente je nach Displaygröße unterschiedlich anzuordnen (oder gar auszublenden). [27]

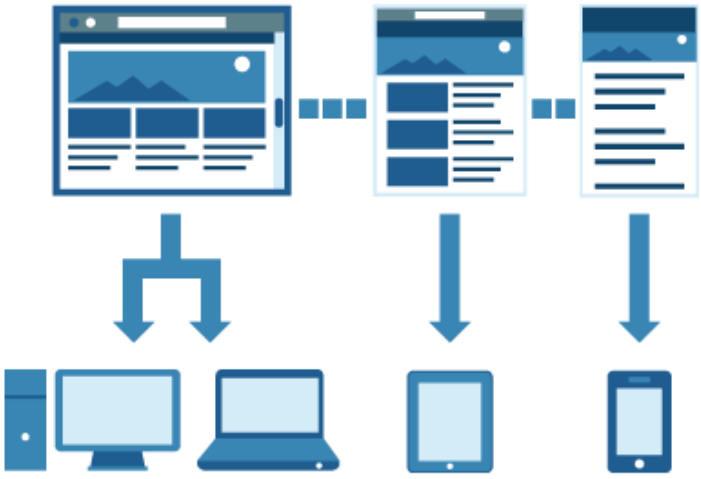


Abbildung 11: Visualisierung des Responsive Design Ansatzes [6]

Abbildung 11 veranschaulicht das zuvor beschriebenen Ansatz des Responsive Designs. In diesem Beispiel wird das Layout für Desktop PCs und Notebooks so ausgelegt, dass mehrere Spalten für den Inhalt gebildet werden, während auf Tablets und Smartphones die Spalten wegfallen und der Inhalt stattdessen untereinander angezeigt wird. Auf Smartphones werden außerdem die Bilder ausgeblendet, sodass das Layout zusätzlich vereinfacht wird und die instabilere Internetsituation von Smartphones ebenfalls beachtet wird.

Doch Responsive Design betrifft nicht nur die groben Container, aus denen sich die Struktur der Seite ergibt (im Falle des Responsive Designs wird diese Anordnung auch als Raster bezeichnet), sondern auch alle Subelemente, die diese Container befüllen müssen flexibel gestaltet werden. So ist es möglich die Größen diverser Elemente wie Buttons je nach Displaygröße zu adaptieren, besonders relevant ist aber unter anderem die Anpassung von Bildern. Während diese in vielen statischen Website-Designs mit fixen Breiten

angegeben werden, ist dies bei Responsive Design Seiten nicht möglich, da die Bilder ansonsten die dynamischen Container sprengen würden. Dementsprechend ist es auch notwendig die Bilder (und alle anderen Subelemente) ebenso dynamisch zu gestalten wie den Hauptraster der Seite. Gelöst werden kann dies entweder mit einem Subraster, der in einem Rasterfeld angelegt wird, oder durch prozentuale Größenangaben. [27]

Zwar gibt es noch zahlreiche weitere Feinheiten im Umgang mit Responsive Design, allerdings sollten die bislang erwähnten Grundkonzepte ausreichend sein um zu verstehen, dass der Grundansatz von Responsive Design mitunter etwas von traditionellen Web-Design Praktiken abweichen kann. Außerdem ist es während des Designprozesse notwendig ständig die alternativen Auflösungen zu beachten in denen die Seite noch renderbar ist. Das führt dazu, dass der Entwicklungsprozess einer Responsive Design Seite aufwändiger ist, als der von traditionellen Seiten. Allerdings steht dem gegenüber, dass mit Responsive Design dauerhaft alle Auflösungsvarianten abgedeckt sind und es nicht nötig ist noch zusätzliche mobile Seiten zu erstellen. Um die Arbeit für Entwickler zu erleichtern und gewisse Standardaufgaben (z.B. Buttondesign, Grid-Layout, Popovers, etc.) beim Erstellen eines Responsive Designs zu vereinfachen existieren zahlreiche Frameworks wie z.B. Bootstrap, die den Entwicklungsprozess beschleunigen und dabei helfen einheitliche Designs innerhalb der Applikation umzusetzen, die auf vielen Geräten getestet und für verschiedene Auflösungen optimiert wurden.

Nachdem lange Zeit Anwendungen primär für Desktop-PCs entwickelt wurden und so angepasst wurden, dass sie anschließend auch auf mobilen Geräten mit etwas limiteriten Umfang lauffähig waren, sorgte die Entwicklung der

Marktanteile (siehe Abbildung 1) für ein Umdenken von vielen Partein zu einem Mobile-first Ansatz. Unter anderem das beliebte Framework Bootstrap, welches bis Version 2 mit einer Desktop-first Philosophie arbeitete, wechselte ab Version 3 zu einem Mobile-first Ansatz. Dies bedeutet, dass Anwendungen primär für mobile Geräte designed werden und das Layout erst anschließend für größere Auflösungen angepasst und angereichert wird. [27] [28] [26]

Responsive Design ist auch hinsichtlich der Suchmaschinenoptimierung zu bevorzugen, da der Ansatz eher dem Dokumentmodell des Internets folgt in dem jedes Dokument eine eindeutige URL bekommt. Das Design wird im Fall von Responsive Design dynamisch angepasst und die Inhalte bleiben stets an eine eindeutige Adresse gebunden. Tatsächlich ist Responsive Design auch jener Ansatz der vom Suchmaschinenbetreiber Google empfohlen wird, insofern es nicht gravierende Gründe gibt, die gegen den Ansatz sprechen würden. Dementsprechend ist für sehr viele Webprojekte Responsive Design in Zukunft auch als Standardlösung heranzuziehen. [29]

5.3 Hybride Applikationen

Hybride Angebote vereinen die Ansätze von nativen Applikationen und von Web-Applikationen und können je nach Design und Implementierung zahlreiche Nachteile beider Ansätze aufheben. Konzeptuell bestehen Hypride Applikationen aus einem nativen Container in den Web-Content geladen wird. Dies ist allerdings eine sehr lose Definition, da die konkrete Art der Implmentierung sehr stark variieren kann.

Im simpelsten Falle stellt die native Applikation einen reinen, funktionslosen Container dar, der nur einen WebView beinhaltet in dem der eigentliche Inhalt von einem Server geladen wird. Die Vorteile gegenber einer reinen Website (die nach wie vor vorhanden ist - der native Container stellt lediglich ein Zusatzangebot dar) ist, dass die Anwendung über die Promotionmöglichkeiten der jeweiligen Stores verfügt und einfach von den Usern auf den HomeScreen ihrer mobilen Geräte installiert werden kann. [7]

Von diesem Minimalstatus ausgehend ist es möglich die native Anwendung nach und nach um Funktionen zu erweitern. Zum Beispiel ist es möglich die Hauptnavigation mittels des nativen Frameworks zu programmmieren und die Inhaltsdarstellung auf einen WebView auszulagern, oder es ist möglich gewisse Sensordaten abzufragen, an den Web-Server zu schicken und entsprechende an die Daten angepasste Inhalte zu bekommen. Vorteil in diesem Fall ist, dass nur die Datengewinnung für jedes Betriebssystem programmiert werden muss, während die Ergebnisdarstellung mit Web-Standards programmiert werden kann und so für alle Betriebssystem übernommen werden kann. Hierbei gibt es keine Grenzen und keine Schablonen, sondern es steht dem Entwickler völlig frei welchen Teil der Anwendung er nativ und welchen er plattformübergreifend programmieren möchte. Dementsprechend können auch die Vor- und Nachteile stärker auf der Seite der nativen Applikationen oder der Web-Applikationen liegen, je nachdem welchen individuellen Ansatz die Entwickler gewählt haben.

Abbilding 12 zeigt die Unterschiede zwischen einem nativen Ansatz, hybriden Ansatz mittels Framework und dem Web Ansatz. Unter einem hybriden Ansatz mittels Framework versteht man, dass eine Middleware wie z.b. das

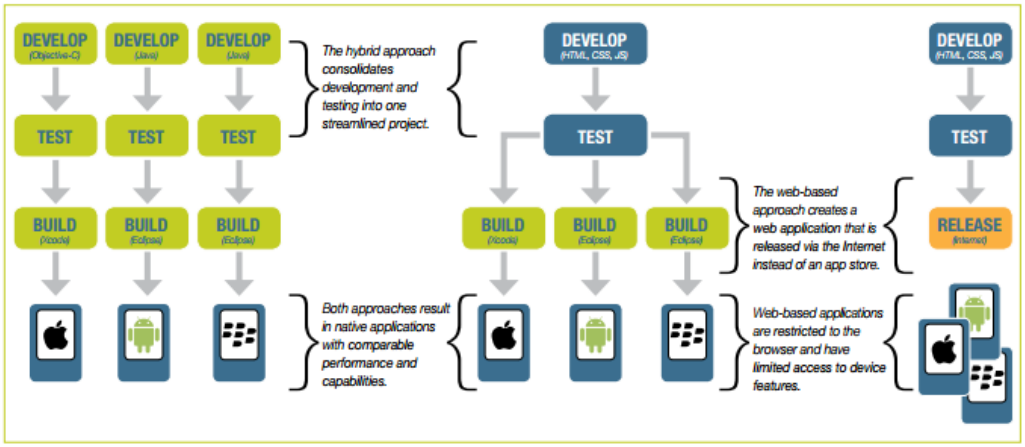


Abbildung 12: Hybrider Ansatz mittels Framework [7]

beliebte Framework Phonegap verwendet wird, mit dessen Hilfe native Applikationen erstellt werden. Der Unterschied zwischen dem Ansatz von rein nativen Applikationen ist allerdings, dass die Anwendung nicht mit verschiedenen Programmiersprachen und Frameworks programmiert werden muss, sondern nur mehr mit einem Framework und der dazugehörigen Programmiersprache. Ähnlich wie Web-Anwendungen werden Phonegap Anwendungen z.B. mittels HTML, CSS und JavaScript geschrieben und das Framework kümmert sich darum, dass die verwendeten Methoden nativ anwendbar sind. [7]

Vorteile von Hybrid-Applikationen

- Native APIs nutzbar
- Anteil an nativem Code und Web-Code liegt im Ermessen der Entwickler
- Promotion- und Monetarisierungseffekte der Stores können genutzt werden

Nachteile von Hybrid-Applikationen

- Expertise in den nativen Entwicklungsumgebungen oder eines Third-Party Frameworks wie Phonegap wird benötigt
- Bei komplexen Applikationen können die Abstraktionsebenen des Frameworks Performanceeinbußen bewirken

6 Anwendungsprojekt

Ausgehend von den zuvor beschriebenen Konzepten, sowie den generellen Best-Practices für mobile Applikationen, soll in diesem Kapitel die Arbeit an einem konkreten Prototypen beschrieben werden, der im Rahmen dieser Diplomarbeit realisiert wurde.

6.1 Aufgabenstellung

Ziel des Anwendungsprojekts war es, eine mobile Schnittstelle für die webLyzard Web Intelligence Plattform [30] zu entwickelt, die speziell auf die Bedürfnisse von Smartphone- und Tablet-Benutzern eingeht aber gleichzeitig auch für Desktop-Benutzer eine lineare und weniger komplexe Bedienung des Systems erlaubt - im Gegensatz zu den komplexeren Synchronisationsmechanismen [31] des regulären Web Intelligence Dashboards.

Abbildung 13 zeigt das webLyzard Portal-Framework im Einsatz. Aufgabe des Frameworks ist es Kunden die Möglichkeit zu geben zu beobachten und zu analysieren wie zuvor definierte Medien über gewisse Themen berichten. Die Media Watch on Climate Change (MWCC) ist eine unter `www.ecoresearch.net/climate` öffentliche zugängliche Anwendung der webLyzard Plattform, welche die Berichterstattung in Nachrichtenmedien und die Diskussionen auf sozialen Plattformen rund um das Theme Klimawandel verfolgt. Das umfangreiche MWCC Dokument-Archiv war die ideale Datenbasis für die im Rahmen dieser Diplomarbeitet entwickelten Applikation.

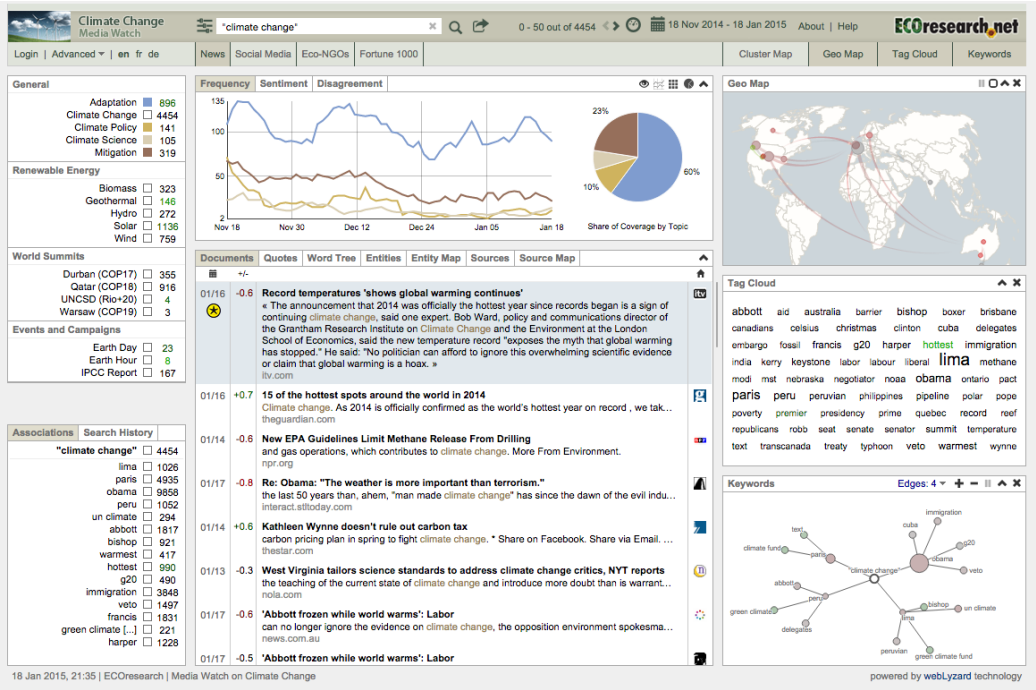


Abbildung 13: Das webLyzard Portal-Framework [8]

Eine Besonderheit des Frameworks ist die dynamische Struktur der Informationsvisualisierung. Sobald der User einen Button klickt, eine Einstellung ändert, oder mit einer Visualisierung interagiert, reagiert das gesamte Portal dynamisch auf diese Interaktion und alle Visualisierungen werden in Echtzeit angepasst. [31]

6.2 Ansatz

Die Backend-APIs des webLyzard-Portalframeworks sind großteils in Java implementiert, und exportieren die relevanten Visualisierungen und Einstellung in Form von JavaScript Elementen. Außerdem ist die Struktur sehr auf eine serverseitige Erzeugung des HTML-Outputs ausgelegt. Das bedeutet,

dass es nur unter großem Mehraufwand möglich gewesen wäre die Struktur zu ändern, und zum Beispiel sämtliche relevanten Bestandteile in JSON-Output zu ändern um sie von einer clientseitigen, nativen Applikation verwerten zu lassen.

Hinzu kommt, dass die meisten verwendeten Visualisierungen in JavaScript erzeugt wurden, und der Look & Feel der Seite sich zu stark geändert hätte, wenn diese Visualisierungen grundlegend neu gestaltet worden wären. Der Performance-relevante Teil der Anwendungen (die Visualisierungen), wäre also auch im Falle des Programmierens einer nativen Applikation weiterhin in JavaScript geschrieben worden und mittels eines WebViews eingebaut worden, was bedeutet, dass die Anwendung keinen nennenswerten Performance-Vorteil aus einer nativen Applikation gezogen hätte.

Weiters war es von Anfang an ein Kernziel der Anwendung, dass es auch möglich ist die vereinfachte, mobile Version des Portals auf Desktoprechnern einzusetzen, um eine simplere Variante des Portalframeworks anbieten zu können. Im Falle einer nativen Applikation wäre dies nur über den Umweg eines Emulators möglich, was für ein produktives Anwendungsszenario wenig reizvoll ist. Durch diese Faktoren und die Tatsache, dass die Erstellung einer nativen Applikation weder vom Erstellungsaufwand, der benötigten Zeit, noch von den Folgekosten Sinn für diese Form der Anwendung ergibt, wurde beschlossen das mobile Portal in Form einer Web-Applikation umzusetzen.

Auf Grund der Komplexität des vorhandenen Portals war es nicht möglich das gesamte Portal in eine einheitliche, responsive Seite umzugestalten. Deshalb wurde ein Mischansatz zwischen einer eigenständigen Applikation und

einem Responsive Design Ansatz gewählt. Es wurde zwar beschlossen eine seperate, mobile Website für das Portal zu erstellen, allerdings wurde diese Website mittels Responsive Design erstellt. Dieser Ansatz birgt den Vorteil, dass keine umfangreichen Änderungsarbeiten am existierenden Portal vorgenommen werden müssen, aber dennoch die Vorteile von Responsive Design genutzt werden können. Im konkreten Fall dieses Projekts, war es sogar ein Vorteil die beiden Angeboten zu splitten um das Portal auch am Desktop in einer vereinfachten Version benutzen zu können.

Als Framework zur Realisierung des Responsive Designs wurde Twitters Bootstrap Framework und das JavaScript Framework jQuery gewählt. Grund für die Auswahl dieser Frameworks waren vorherige, positive Erfahrungen mit den Frameworks beim Erstellen von responsiven Websites, der ausgeprägte Support im Internet, da beide Frameworks zu den beliebtesten JavaScript Frameworks zählen und die Verfügbarkeit von Plugins und Erweiterungen, wie z.B. einem DateTime-Picker, der anhand der Bootstrap und jQuery Standards entwickelt wurde und so einfach in das Projekt einfügbar ist.

6.3 Umsetzung

Sämtliche hier besprochenen Ansätze richten sich nach den Best Practices für mobile Geräte, die zuvor ausgearbeitet wurden und deren Theorie in Kapitel 4 erläutert wurde. In Folge soll auf die Detailentscheidungen während des Designs des Anwendungsprojekt eingegangen werden und die konkreten Entscheidungen gezeigt und erläutert werden.

6.3.1 Generelles Konzept

Auf Grund der limitierten Raumverhältnisse auf mobilen Geräten musste mit dem vorhanden Bildschirmplatz sehr sorgfältig umgegangen werden. Durch die Komplexität und Struktur der vorhandenen Applikation, ist es allerdings dennoch nicht möglich die Navigation und die Einstellungen zu sehr in den Hintergrund zu drängen, die diese ein zentrales Elemente der Interaktion mit der Anwendung darstellen.

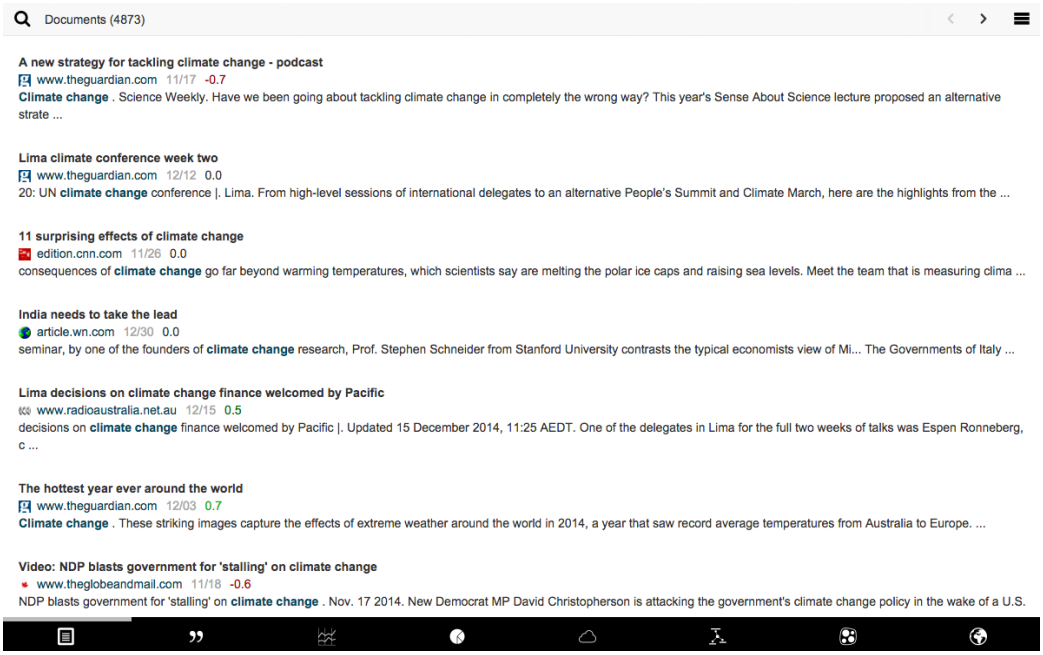


Abbildung 14: Hauptbereich der mobilen Applikation

Abbildung 14 zeigt den gewählten Designansatz für das mobile Portal. Der Bildschirm wurde insgesamt in fünf Hauptteile zerlegt, von denen drei permanent sichtbar sind, und zwei nur bei Bedarf erscheinen. Die beiden optionalen Bildschirmelemente befinden sich links und rechts vom Hauptscreen und werden auf Buttonklick ein- bzw. ausgeblendet. Auf diese Elemente und

das dazugehörige Navigationskonzept wird später noch im Detail eingegangen. Die drei permanent sichtbaren Bildschirmteile befinden sich zentral im Bild. Dies sind die beiden Navigations-Toolbars am oberen und unteren Rand des Bildschirms und der zentrale Inhaltsbereich in dem die Ergebnisse der Interaktionen präsentiert werden.

Auf Grund des beschränkten inhaltlichen Raums auf mobilen Geräten wurde entschieden die inhaltliche Parallelität der Desktop Anwendung in ein anderes Designmodell zu transformieren. Während die Desktop-Applikation die Visualisierungen parallel neben und übereinander anzeigt, beschränkt sich die mobile Version auf eine Visualisierung, die den vollen Bildschirm einnimmt. Um dennoch einen Überblick über die gesamten vorhandenen Visualisierungen zu erlauben wurde in der unteren Toolbar der Anwendung eine Navigation für die Visualisierungen eingebaut. Entweder durch Klick auf ein Icon, oder durch Swipen im Hauptcontent lässt sich die Art der Visualisierung ändern.

Wesentliches Designelement der mobilen Version war es zu versuchen den Charakter der Ausgangsapplikation zu erhalten und diesen in ein mobiles Benutzungsszenario zu konvertieren. Konkret betrifft dies das Interaktionsmodell der Anwendung. In der Desktopversion sorgt ein Klick, eine Änderung der Einstellungen oder eine Suche automatisch für eine dynamische Anpassung sämtlicher zu sehenden Visualisierungen. In der ersten Version der Anwendung wurde versucht diesen Effekt auch für die mobile Version zu übernehmen. Dies führt auf Grund des zu sehr befüllten DOM zu starken Performance-Problemen. Da außerdem ohnehin nicht alle Visualisierungen zur selben Zeit am Bildschirm zu sehen sind, wäre diese eine Verschwen-

dung von Resourcen, die noch dazu nicht den gewünschten dynamischen Effekt hatte, da das gleichzeitige Rendern aller Visualisierungen zu starken Verzögerungen geführt hat. Deshalb wurde entschieden, dass eine Interaktion lediglich die Daten einholt und zwischenspeichert, die zum Rendern der Visualisierungen benötigt werden und lediglich die aktuelle Visualisierung gerendert wird. Erst durch eine Navigation zu einer neuen Visualisierung wird diese gerendert. Dies geschiet auf Grund des Datenzwischenspeichers sehr schnell und sorgt auf Grund des reduzierten Renderaufwands für ein dynamischeres User-Interface das näher am Charakter der Desktopversion ist, als eine genaue Reproduktion des Desktop-Mechanismus es auf mobilen Geräten erlaubt hätte.

6.3.2 Navigationskonzept

Abbildung 15: Obere Navigation der mobilen Applikation

Die obere Navigation der mobilen Applikation hat zwei Hauptanwendungsgebiete. Zum einen hat sie zwei statische Elemente, die in Form von Buttons gestaltet sind und entweder das Suchmodul auf der linken Seite, oder das Einstellungsmodul auf der rechten Seite öffnen. Zum anderen dient die obere Navigation aber auch dazu um spezifische Details zur aktuellen Visualisierung, sowei visualisierungsspezifische Optionen anzuzeigen. Bei allen Visualisierungen wird dabei der Titel der Visualisierung angezeigt (z.B. Do-

cuments), die anderen Elemente der Anzeige variieren je nach Visualisierung und auch nach dem Status der Visualisierung. Beispielsweise wird in der Document-Visualisierung die Anzahl der Dokumente angezeigt, sowei eine Pfeilnavigation, falls mehrere Seiten an Suchergebnisse gefunden werden, während im Frequency Chart die Möglichkeit gegeben ist mit den Buttons die Details der Visualisierung zu verändern.

Abbildung 16: Untere Navigation der mobilen Applikation

Die untere Navigation dient zum Durchschalten der verschiedenen Visualisierungsmethoden. Die Icons wurden so gestaltete, dass sie grafisch die jeweilige Visualisierung eindeutig repräsentieren. Durch einen Klick auf das jeweilige Icon lässt sich die Visualisierung zum aktuellen Suchergebnis wechseln. Um die Dynamik der Interaktion zu verstärken wird zusätzlich zu den statischen Icons eine dynamische graue Leiste angezeigt, die sich in der Standardposition über der aktiven Visualisierung befindet. Beginnt der User allerdings mit einer Swipe-Geste quer über den Bildschirm zu wischen, so bewegt sich der graue Balken gleichermaßen (bis zu einem gewissen Limit) entlang der Bewegung mit. Dies soll den User dabei unterstützen die Navigation besser zu verstehen und die horizontale Anordnung der Visualisierung andeuten, die nacheinander durchgeschaltet werden können. Um das User-Interface konsistent zu gestaltet wird auch bei einem Klick auf ein Icon der graue Indikator nicht einfach umgeschaltet, sondern gleitet in die neue Position um den dynamischen Charakter der Navigation zu veranschaulichen.

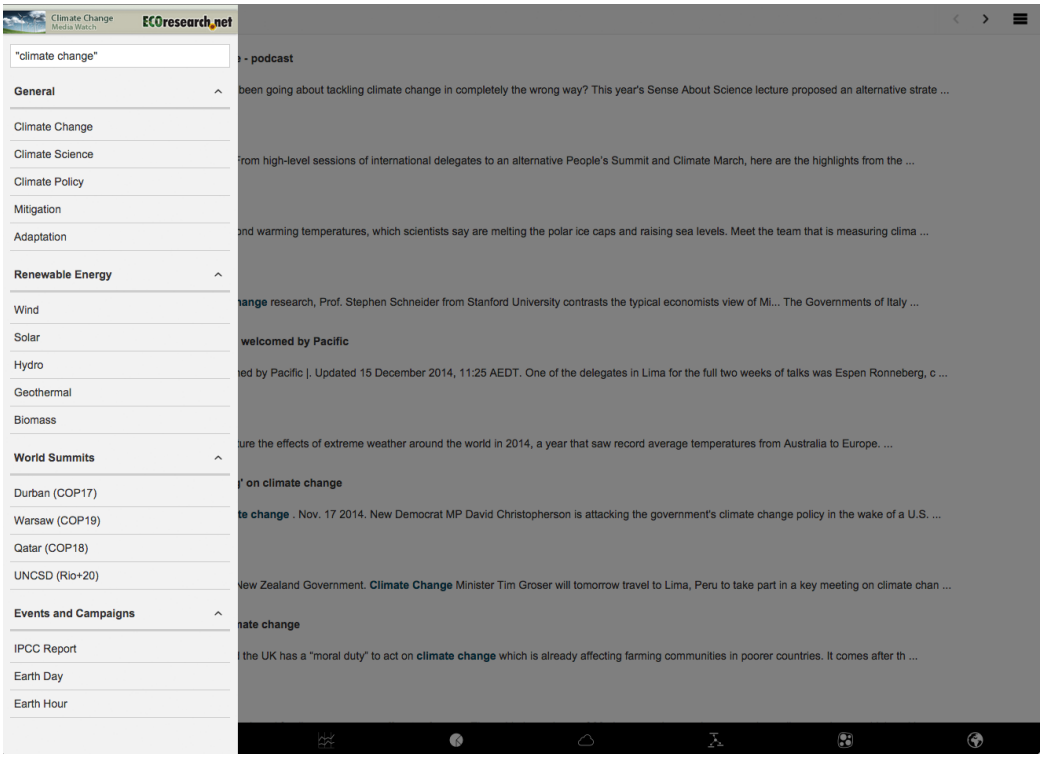


Abbildung 17: Linke Navigation der mobilen Applikation

Abbildung 17 zeigt die linke Navigation der mobilen Applikation, die angezeigt wird, sobald der Benutzer den Suchbutton in der Top-Navigation klickt. Hier befinden sich vorgefertigte Themen, die von den Portal-Administratoren festgelegt werden können. Ziel ist es eine schnelle Anlaufstation für häufig gesuchte Themen zu bieten. Das linke Menü dupliziert dabei die Funktion des linken Bereichs der Desktop-Version, allerdings wurde auch die Suchfunktion in das linke Menü verschoben. Grund dafür ist, dass die Prinzipien der Suche in der Desktopversion auf volle Parallelität ausgelegt ist, während in der mobilen Darstellung auf eine sequentielle Anordnung von Suchanfragen und Ergebnisdarstellungen gesetzt wurde.

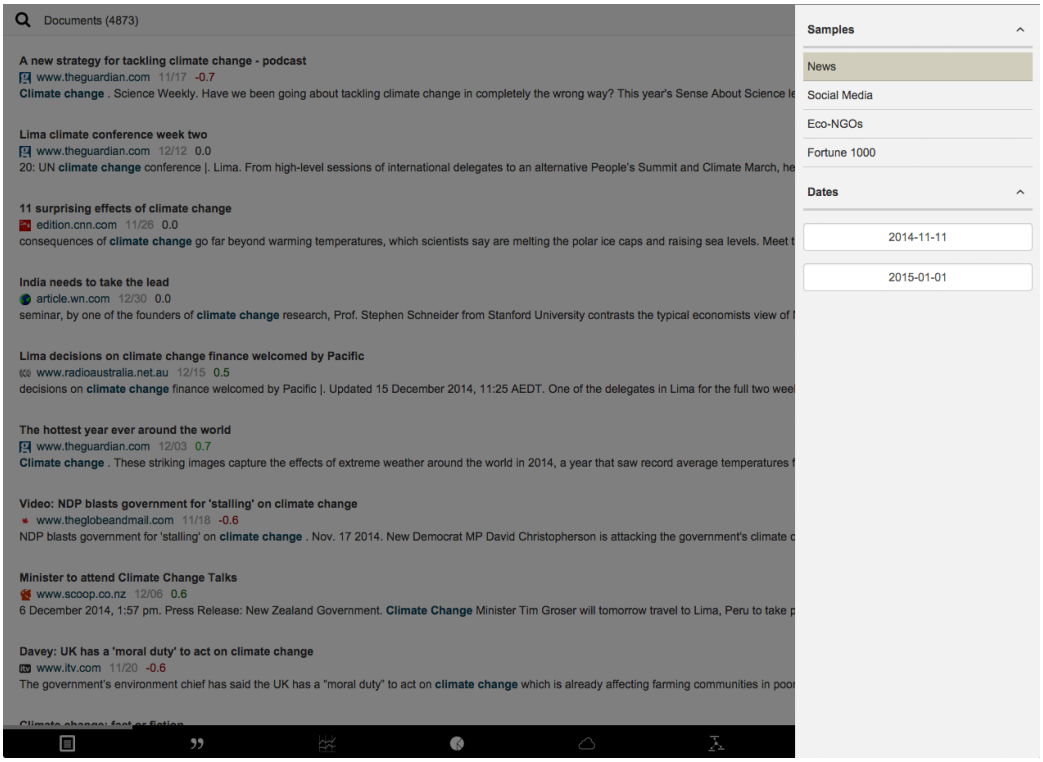


Abbildung 18: Rechte Navigation der mobilen Applikation

Die rechte Navigation (siehe Abbildung 18) wird aktiviert durch Klicken des Icons mit den drei horizontalen Linien, in der Top-Navigation. Im Gegensatz zur linken Navigation, die zur Aktivierung der Suche benötigt wird, ist die rechte Navigation eine reine Einstellungsseite, die in der aktiven Bedienung der Anwendung eine untergeordnete Rolle einnimmt und lediglich verwendet wird um Paramter für die Suche einzustellen. Konkret wird der Art der zu durchsuchenden Dokumente bestimmt und ein Datumsfilter gesetzt um festzulegen zwischen welchen Zeitraum die zurückgelieferten Suchresultate liegen müssen.

Sowohl das linke, als auch das rechte Navigationsmenü gleitet auf Knopfdruck dynamisch vom Bildschirmrand über den Hauptcontent. Die Animation soll

die Menüs hinsichtlich ihrer Position eindeutig verorten und außerdem die dynamische Komponente der Anwendung visualisieren. Beide Navigationen bekommen standardmäßig eine Breite von 80% der Bildschirmbreite zugewiesen, werden aber auf einen maximalen Wert von 300px begränzt. Dieser Mechanismus soll garantieren, dass stets ein rechter Rand erkennbar bleibt. Da beim Aktivieren der Menüs der Hauptcontent abgedunkelt wird, aber dennoch sichtbar bleibt erhält der Benutzer so einen festen Punkt, der die Position der Elemente visuell verankert und eine Verwirrung beim Benutzer vermeidet.

Wie die oben ausgeführten Erläuterungen zeigen, wurde Wert darauf gelegt die Bedienkonzepte der Anwendung so zu gestalten, dass sie den Navigations- und Layoutprinzipien entsprechen, die in Kaptiel 4 erläutert wurden. So befinden sich die zentralen Menüs an der Ober- und Unterkante in Form einer Leiste und der Inhaltsbereich wurde mittels des Viewdesign Pattern gestaltet, das unter anderem in Android und iOS integriert ist. Außerdem wurden Texteingaben möglichst vermieden in dem z.B. beliebte Suchterme vorformuliert werden, die Datumseingabe mittels eines grafischen DatePickers erfolgt und es für die Suche eine Autocomplete-Funktion gibt.

6.3.3 Visualisierungen

Der Hauptinhalt der Applikation sind die im zentralen Fenster dargestellten Visualisierungen. Die Auswahl der Visualisierung erfolgt wie zuvor beschrieben durch die Implementierung eines Tab View Pattern, das eine einfache Navigation erlaubt. In folge werden die einzelnen Detail Views der Visualisierungen vorgestellt und ihre Besonderheiten erläutert.

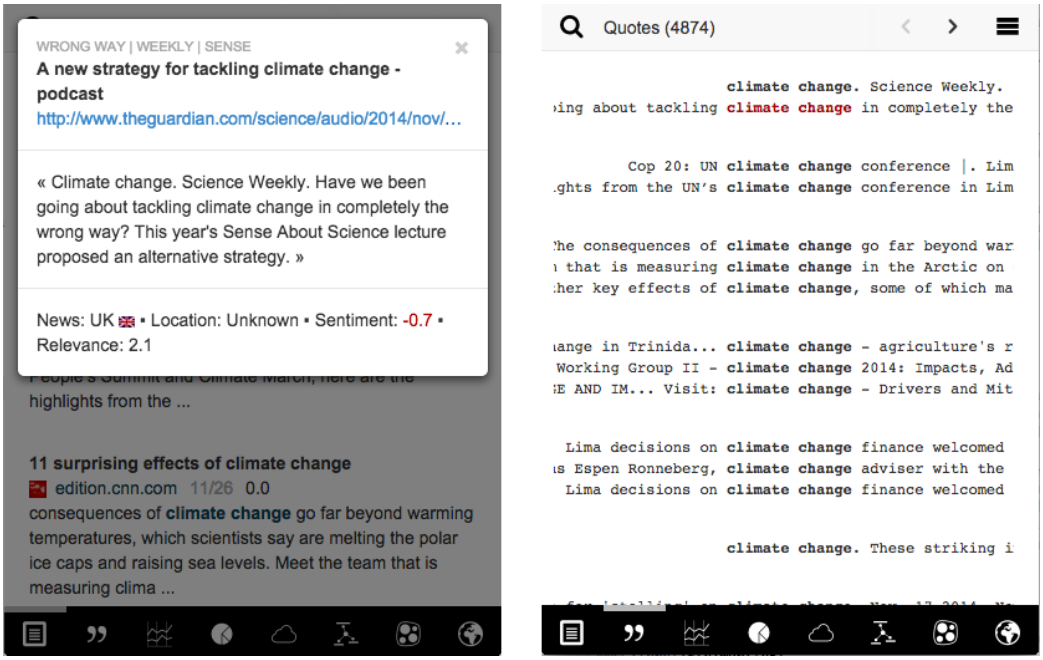


Abbildung 19: Document- und Quotes-View der mobilen Applikation

Abbildung 19 zeigt den Document View und den Quotes View der Applikation. Beide views liefern dabei eine Liste an Dokumenten, die auf unterschiedliche Art visualisiert werden. Der Document View liefert eine Liste ähnlich wie aus einer Suchmaschine, mit einem kurzen Auszug aus dem Dokument, während der Quotes View Zitate aus dem Dokument darstellt, die den verwendeten Suchbegriff beinhalten. Die beiden views folgen dabei dem bereits vorgestelltem Pattern der Listendarstellung, das auf klick zu einer Detailansicht führt.

Abbildung 20 zeigt den Line-Chart- und Pie-Chart-View der Applikation. Für diese beiden views erfolgt eine context-sensitive Anpassung des linken Navigationsmenüns um die Funktionalität der Visualisierung zu ermöglichen. Während die vorgegebenen Terme in den anderen views lediglich anklickbar

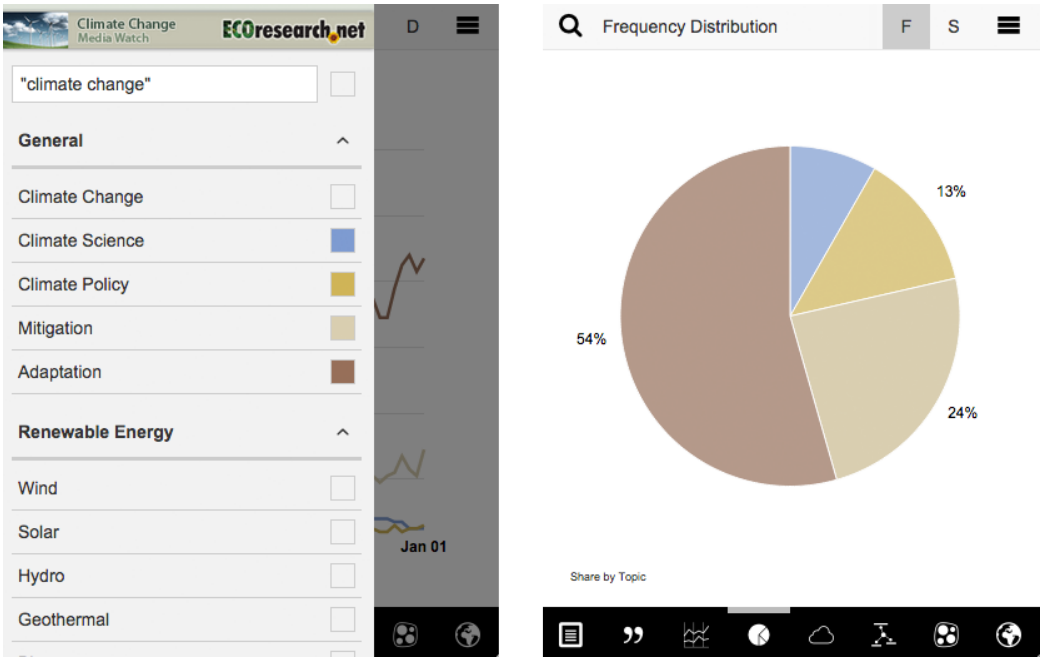


Abbildung 20: Line-Chart- und Pie-Chart-View der mobilen Applikation

sind um eine Suche auszulösen, sind sie in diesen views zusätzlich auch noch markierbar über eine Box am rechten Rand des Menüs. Dies ermöglicht es Terme auszuwählen und mit dem aktuellen Suchbegriff zu vergleichen. Die Visualisierungen an sich zeigen anschließend die Häufigkeit der gewählten Begriffe oder das Sentiment [32] der gefundenen Dokumente an.

Abbildung 21 zeigt mit Tag Cloud, Keyword Graph, Cluster Map und Geo Map die restlichen vier Visualisierungen. Da keine Designanpassungen für mobile Geräte notwendig waren, sondern die Visualiserungen bereits in der Desktop-Anwendung für eine responsive Darstellung in verkleinerbaren Fenstern ausgelegt waren, soll an dieser Stelle nur auf die allgemeine Funktionalität der Visualisierungen eingegangen werden. Die Tag Cloud zeigt verwandte Suchbegriffe an, die hinsichtlich ihres Sentiments grün, rot oder schwarz

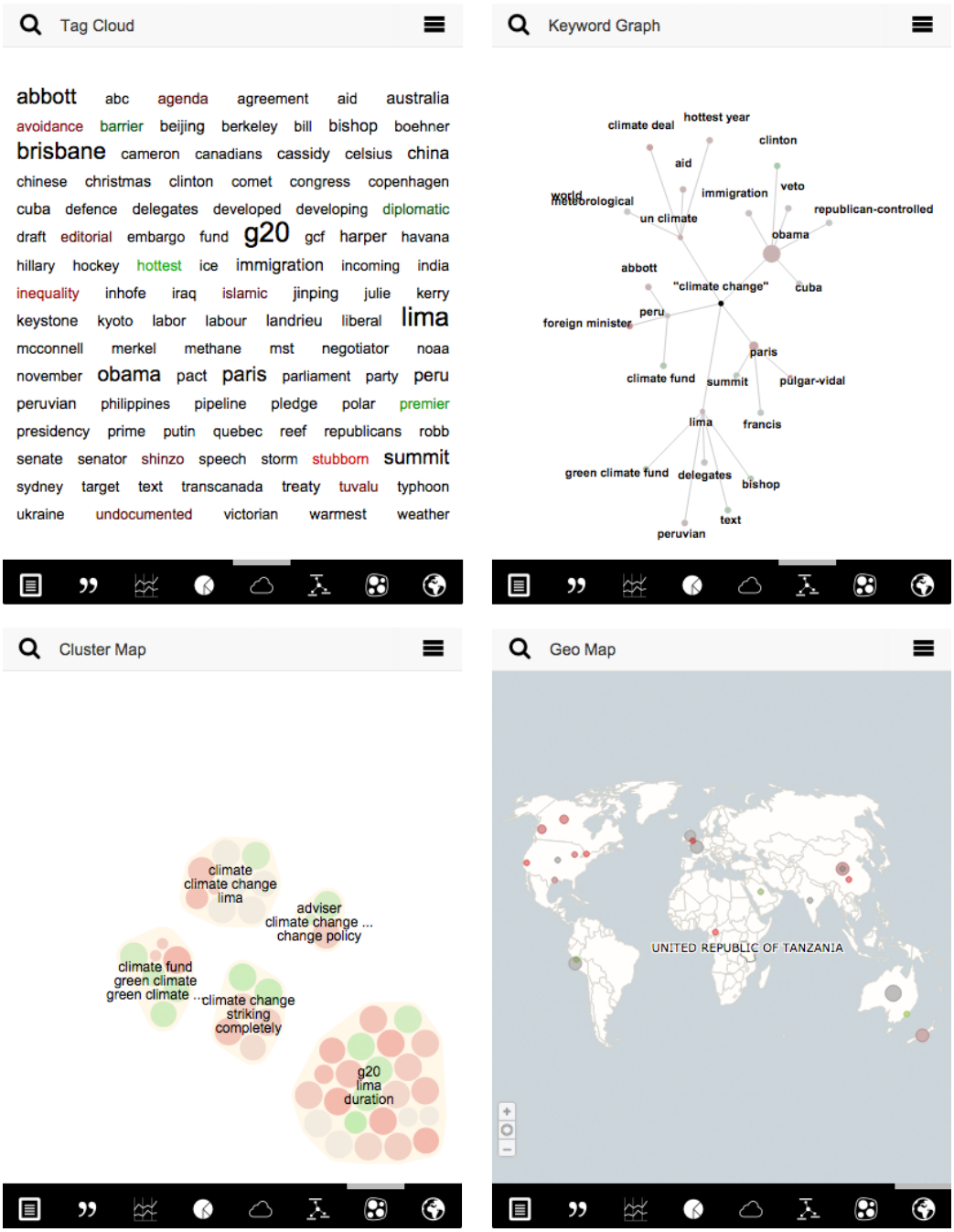


Abbildung 21: Tagcloud-, Keyword-Graph, Clustermap und Geo-Map-View der mobilen Applikation

markiert sind (bzw. in Abstufungen der jeweiligen Farben). Durch einen Klick auf einen der Begriffe wird eine neue Suche ausgelöst und die Tag Cloud passend zum neuen Begriff generiert. Der Keyword Graph zeigt passend zum aktuellen Suchbegriff weitere Begriffe an und visualisiert in welcher Abhängigkeit diese zueinander stehen. Durch einen Klick auf eines der Elemente lassen sich weitere Kinder ein- oder ausblenden. Die letzte Visualisierung ist die Geo Map, die zeigt in welchen Regionen der Erde über den aktuellen Suchbegriff berichtet wurde und welches Sentiment der Bericht hat.

7 Zusammenfassung

Auf Grund der Markttrends hinsichtlich der Verbreitung von mobilen Geräten ist es für Entwickler längst nicht mehr optional eine optimierte, mobile Version ihrer Angebote zur Verfügung zu stellen. In vielen Fällen dürfte dabei der Ansatz des Responsive Designs, wie in Kapitel 5.2.2 besprochen, für Web-Applikationen die beste Methode hinsichtlich des Kosten-Nutzen-Verhältnisses sein. Um zusätzliche, positive Effekte ohne allzu große Hürden zu erzielen bieten sich auch eine zumindest simple Hybrid Applikation (siehe Kapitel 5.3) an, die eine vorhandene Web-Applikation einbettet und es den Entwicklern so ermöglicht die Plattform der nativen Stores zu verwenden.

Da native Applikationen deutlich teurer und aufwändiger zu erstellen und zu warten sind, ihre Vorteile aber nur in wenigen Fällen wirklich Auswirkungen hinsichtlich der Performance oder der Sensorennutzung haben, macht es auf Grund der einfacheren Alternativen nur mehr in Sonderfällen Sinn auf native Applikationen zu setzen. Zu diesen Sonderfällen zählen Spiele, oder sonstige performancealastige Applikationen und auch Anwendungen die sehr stark auf die hardwarenahen Schnittstellen der Geräte setzen.

Auf Grund des explorativen Charakters von Touch-Interaktionen, der sich durch ihren direkten, ungenaueren Charakter deutlich von der indirekten aber präzisen Maussteuerung unterscheidet, ist es notwendig vorhandene Best Practices und Design Pattern zu verwenden, die Entwickler dabei unterstützen Interaktionsmodelle zu finden, die bei den Usern über eine hohe

Akzeptanz verfügen. Dabei ist es besonders hilfreich sich an den nativen Elementen zu orientieren, die von den verbreiteten Betriebssystemen aufgegriffen wurden und dadurch vielen Benutzern vertraut sind.

So ist es auch möglich den aktuellen Designtrends zu flachen Designs zu folgen. Während frühere Designmodelle wie der Skeuomorphismus geholfen haben gewisse Interaktionsmethoden bei den Benutzern zu etablieren, können aktuelle Designs nützlich sein diese Lerneffekte aufzugreifen und für neue Design-Sprachen zu verwenden.

All diese Überlegungen wurden anhand eines konkreten Anwendungsprojekts erprobt. In Kapitel 6 wurde beschrieben wie die Media Watch on Climate Change für mobile Geräte optimiert wurde. Dieses Projekt zeigte die Flexibilität der Best Practices und wie diese sich anwenden und kombinieren lassen um zu einer fertigen Applikation zu gelangen. Die nächsten Schritte zur Erweiterung dieses Projekts wäre das Hinzufügen von weiteren Visualisierungen und die Ergänzung von weiteren Suchmodellen, die zusätzlich zu den existierenden Methoden der Spiegelung von Newsmedien und Social Media Quellen angeboten werden können.

8 Curriculum Vitae

Ausbildung

seit 2012:	Masterstudium „Information Systems“an der WU-Wien
seit 2006:	Diplomstudium „Theater- Film- und Medienwissenschaften“ an der Universität Wien
2006-2012:	Bachelorstudium der „Wirtschafts- und Sozialwissenschaften“ an der WU-Wien, Studienzweig „Wirtschaftsinformatik“
2000-2005:	HTL Waidhofen/Ybbs, Fachrichtung „Automatisierungstechnik“

Berufserfahrung

seit 2011:	WU-Wien - Software Entwickler am Institut für Informationswirtschaft
2013-2014:	consol.Media - Programmierer

Sonstige Qualifikationen

Sprachen:	Deutsch und Englisch in Wort und Schrift
Forschungsgebiete:	Semantic Web, Games With a Purpose, Recommender-Systeme, Mobile Computing, Crowdsourcing
Programmieren:	PHP, Java, Python, JavaScript

Literatur

[1] Edwards, Jim: *Mobile Apps Are Killing The Free Web, Handing A Censored Duopoly to Google And Apple.* Website, 2014. – http://www.businessinsider.com/mobile-web-vs-app-usage-statistics-2014-4?IR=T; Abgerufen: 13.10.2014

[2] DDB: *SKEUOMORPHISM VS FLAT DESIGN.* Website, 2014. – http://dmierzdesignblog.com/post/51586050423/skeuomorphism-vs-flat-design; Abgerufen: 15.10.2014

[3] Google: *Android Design.* Website, 2015. – https://developer.android.com/design/index.html; Abgerufen: 15.10.2014

[4] IDC: *Smartphone OS Market Share, Q3 2014.* Website, 2014. – http://www.idc.com/prodserv/smartphone-os-market-share.jsp; Abgerufen: 15.10.2014

[5] Berkholz, Donnie: *GitHub language trends and the fragmenting landscape.* Website, 2014. – http://redmonk.com/dberkholz/2014/05/02/github-language-trends-and-the-fragmenting-landscape/; Abgerufen: 17.10.2014

[6] Boorder, Elles de: *Responsive Design: Hype or Solution?* Website, 2011. – http://www.paulolyslager.com/responsive-design-hype-solution/; Abgerufen: 15.10.2014

[7] Christ, Adam M.: Bridging the Mobile App Gap. In: *Connectivity and the User Experience* 11 (2011), Nr. 1, S. 27–32

[8] Ecoresearch: *Media Watch on Climate Change*. Website, 2014. – https://www.ecoresearch.net/climate; Abgerufen: 15.12.2014

[9] Heisler, Yoni: *Report: 90% of iOS apps are free; average cost of an iOS app is 19 cents*. Website, 2013. – http://www.tuaw.com/2013/07/18/90-of-ios-apps-are-free-average-cost-of-an-ios-app-is-19-cents/; Abgerufen: 14.10.2014

[10] Weller, Nathan B.: *Web Design Trends To Look Out For In 2015*. Website, 2014. – http://www.elegantthemes.com/blog/resources/web-design-trends-to-look-out-for-in-2015; Abgerufen: 14.10.2014

[11] Manovich, Lev: The Back of Our Devices Looks Better than the Front of Anyone Else's. In: Snickars, P. (Hrsg.) ; Vonderau, P. (Hrsg.): *Moving Data: The IPhone and the Future of Media*. New York City : Columbia University Press, 2012, S. 278–286

[12] Campanelli, Vito ; Bardo, Francesco ; Heber, Nicole: *Web aesthetics: how digital media affect culture and society*. NAi Publishers, 2010

[13] Schneider, Alexandra: The iPhone as an Object of Knowledge. In: Snickars, P. (Hrsg.) ; Vonderau, P. (Hrsg.): *Moving Data: The IPhone and the Future of Media*. New York City : Columbia University Press, 2012, S. 278–286

[14] Hou, Kai-Chun ; Ho, Chun-Heng u. a.: A preliminary study on aesthetic of apps icon design. In: *IASDR 2013* (2013)

[15] Apple: *iOS Human Interface Guidelines: Introduction*. Website, 2015. – https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html; Abgerufen: 15.10.2014

[16] Zahler, Thomas: *Die Wahrheit über Skeuomorphismus vs. Flat Design*. Website, 2013. – http://intuio.at/blog/die-wahrheit-uber-skeuomorphismus-vs-flat-design/; Abgerufen: 15.10.2014

[17] Gibson, James J.: The theory of affordances. In: *Hilldale, USA* (1977)

[18] Bieh, Manuel: *Mobiles Webdesign - Konzeption, Gestaltung, Entwicklung*. Galileo Press, 2008

[19] Rabin, Jo ; McCathieNevile, Charles: *Mobile Web Best Practices 1.0*. Website, 2008. – http://www.w3.org/TR/mobile-bp/; Abgerufen: 15.12.2014

[20] Apple: *Start Developing iOS Apps Today*. Website, 2014. – https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/index.html#//apple_ref/doc/uid/TP40011343; Abgerufen: 15.10.2014

[21] Google: *Android Studio*. Website, 2014. – http://developer.android.com/sdk/index.html; Abgerufen: 15.10.2014

[22] Microsoft: *Getting started with developing for Windows Phone 8*. Website, 2014. – http://msdn.microsoft.com/en-us/library/windows/apps/ff402529%28v=vs.105%29.aspx; Abgerufen: 15.10.2014

[23] Charland, Andre ; Leroux, Brian: Mobile Application Development: Web vs. Native. In: *Commun. ACM* 54 (2011), Mai, Nr. 5, S. 49–53

[24] Fling, B.: *Mobile Design and Development: Practical concepts and techniques for creating mobile sites and web apps.* O'Reilly Media, 2009 (Animal Guide)

[25] Gardner, Brett S.: Responsive Web Design: Enriching the User Experience. In: *Connectivity and the User Experience* 11 (2011), Nr. 1, S. 13–19

[26] Frain, B.: *Responsive Web Design with HTML5 and CSS3.* Packt Publishing, Limited, 2012

[27] Marcotte, E.: *Responsive Web Design.* A Book Apart, 2011

[28] Bootstrap: *Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.* Website, 2014. – `http://getbootstrap.com/`; Abgerufen: 15.10.2014

[29] Google: *Recommendations for building smartphone-optimized websites.* Website, 2012. – `http://googlewebmastercentral.blogspot.co.at/2012/06/recommendations-for-building-smartphone.html`; Abgerufen: 15.10.2014

[30] Scharl, Arno ; Hubmann-Haidvogel, Alexander ; Sabou, Marta ; Weichselbraun, Albert ; Lang, Heinz-Peter: From Web Intelligence to Knowledge Co-Creation: A Platform for Analyzing and Supporting Stakeholder Communication. In: *IEEE Internet Computing* 17 (2013), Nr. 5, S. 21–29. – ISSN 1089–7801

[31] Hubmann-Haidvogel, Alexander ; Scharl, Arno ; Weichselbraun, Albert: Multiple coordinated views for searching and navigating Web content repositories. In: *Information Sciences* 179 (2009), Nr. 12, S. 1813 – 1821

[32] Weichselbraun, A. ; Gindl, S. ; Scharl, A.: Extracting and Grounding Contextualized Sentiment Lexicons. In: *Intelligent Systems, IEEE* 28 (2013), March, Nr. 2, S. 39–46

„Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.“