

Inhaltsverzeichnis

1 Einleitung	3
2 Windows 8	4
2.1 Neues Bedienkonzept	4
2.2 Windows 8 als hybrides System	4
2.3 Windows RT	4
2.4 Das Ökosystem	4
2.4.1 Microsoft Store	4
2.4.2 Windows Phone	4
3 Bildsprache	5
4 Konzeption der App	6
4.1 Ziel der App	6
4.2 Nativ vs. Web	7
4.2.1 Native App	7
4.2.2 Windows 8 App mit Webtechnologien	8
4.3 Navigationskonzept	8
4.3.1 Flaches System	8
4.3.2 Hierarchisches System	9
4.4 Menü und Features	11
5 Umsetzung der App	12
5.1 Rastervorlage	12
5.1.1 HTML Dateien	12
5.1.2 Javascript Dateien	13
5.1.3 CSS Dateien	13
5.2 Datenanbindung	13
5.2.1 ZEIT ONLINE Datenbestand	13
5.2.2 Daten verfügbar machen	15
5.3 Darstellung der Daten	18
5.3.1 Die Hubansicht	18
5.3.2 Die Ressortansicht	20
5.3.3 Die Artikelansicht	20
5.4 Features	22
5.4.1 Artikeltitel ausblenden	22
5.4.2 Schriftgröße verändern	24
5.4.3 Semantic Zoom	24
5.5 Entwicklungschronologie	26

6 Evaluierung	27
6.1 Fokusgruppe	27
6.2 Interview	27
6.2.1 Intuitivität der App	27
6.2.2 Der nur Bilder Modus	27
6.3 Ergebnis	27
7 Fazit	28

1 Einleitung

2 Windows 8

Seit dem 26. Oktober 2012 ist Microsofts aktuellstes Betriebssystem „Windows 8“ für die breite Öffentlichkeit verfügbar. Dieses unterscheidet sich grundlegend von seinem Vorgänger „Windows 7“. Microsoft verfolgt mit Windows 8 den Ansatz, das gleiche Betriebssystem sowohl für Desktop- als auch für Tablet-Computer¹ zu verwenden. Es besitzt nach wie vor den bekannten Desktop von Windows 7. Äußerlich sind hier nur kleine Änderungen vorgenommen worden. So besitzt z.B. der Windows Explorer eine neues, kontextsensitives Ribbon-Menü². Neu ist hingegen das "Modern User Interface"(Modern UI, früher Metro), eine für Touch-Gesten optimierte Oberfläche. Obwohl die neue Oberfläche für Touch-Gesten optimiert ist, kann sie ebenso mit Maus und Tastatur bedient werden. Es können, im Gegensatz zum Desktop, keine herkömmlichen Programme im Modern UI ausgeführt werden. In der Modern UI Oberfläche laufen nur Programme (Apps), die speziell für Windows 8 entwickelt wurden und über den Microsoft Store (siehe Sektion 2.4.1 auf Seite 4) erhältlich sind. Es soll nun ein etwas detaillierterer Blick auf die neue Oberfläche und dessen Eigenheiten geworfen werden.

2.1 Neues Bedienkonzept

Beim Hochfahren von Windows 8, erwartet den User ein Interface, das er so vom Vorgänger Windows 7 nicht kennt. Viele bunte, viereckige Kacheln, in verschiedenen Größen. So sieht die neue Startoberfläche von Windows 8 aus (siehe Abbildung 1). Der Startbildschirm sind die verschiedenen Programme angeordnet. Dies können „normale“ Programme, die auf dem Desktop laufen oder es können Modern UI Apps (Windows-Store) sein, welche nur über den Microsoft Store zu installieren sind.

2.2 Windows 8 als hybrides System

Windows 8 kann sowohl auf herkömmlichen Desktop Computern als auch auf Tablet-Computern eingesetzt werden.

2.3 Windows RT

2.4 Das Ökosystem

2.4.1 Microsoft Store

2.4.2 Windows Phone

¹Tablet-Computer: kleiner, flacher, tragbarer Computer mit einem Touchscreen. Es wird von nun an, der Einfachheit halber die englische Version „Tablet“ benutzt.

²Auch bekannt als Menüband oder Multifunktionsleiste.

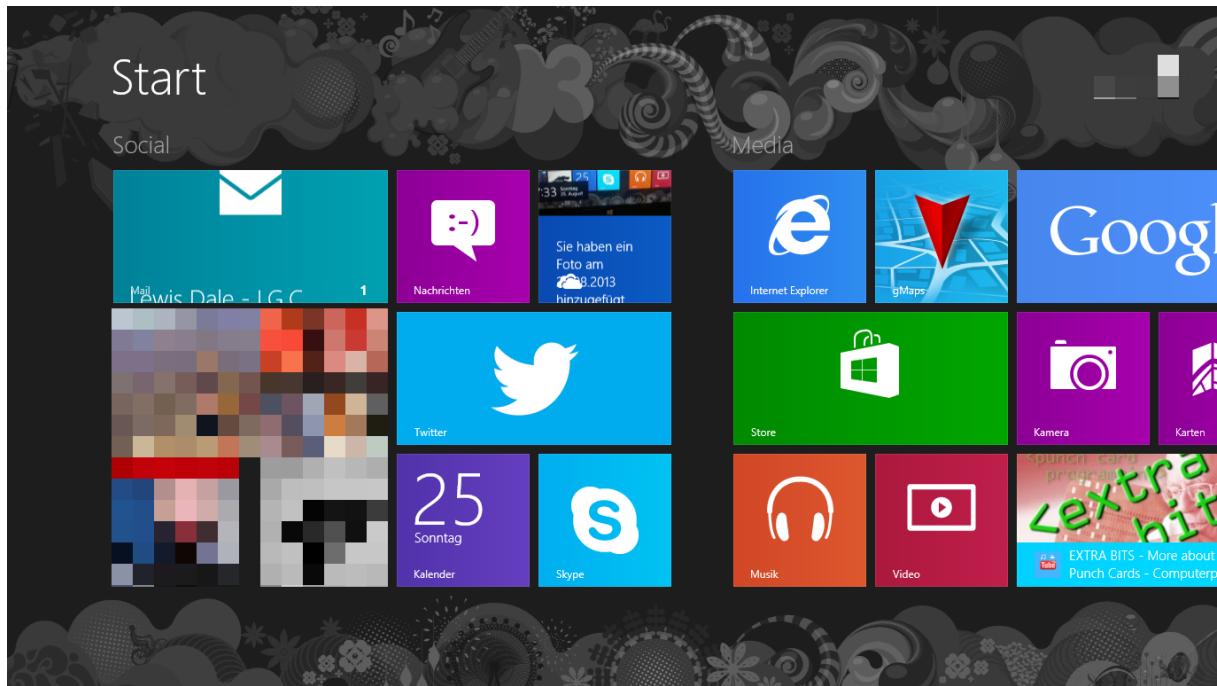


Abbildung 1: Der Startbildschirm von Windows 8.

3 Bildsprache

4 Konzeption der App

In diesem Kapitel geht es darum, die Konzeption für eine Windows 8 Nachrichten App darzustellen und zu erläutern. Dabei gilt es darzulegen, warum und mit welchem Hintergrund Entscheidungen zu Gunsten der einen oder der anderen Möglichkeit ausfallen. Dazu müssen zunächst die Ziele der App ausgeführt werden. Anschließend muss sich entschieden werden mit welcher Technologie bzw. Programmiersprache gearbeitet werden soll. Zuletzt wird noch ein detaillierterer Blick auf das Herzstück der App geworfen. Das Navigationskonzept. Hierbei ist eine der wichtigsten Fragen: „Wie kann ich auch bei ggf. großen Datenmengen eine übersichtliche, intuitive Struktur schaffen die sich in das Gesamtkonzept von Windows 8 eingliedert?“

4.1 Ziel der App

Das Ziel der zu erstellenden App ist, den Inhalt der ZEIT ONLINE Website auf ansprechende Art und Weise in einer Windows 8 Applikation darzustellen. Das Layout und die Darstellung der Inhalte soll sich an das sogenannte „Look and Feel“ von Windows 8 anpassen und sich daran orientieren. Der Fokus bei den Inhalten liegt auf den Artikeln selbst und den jeweiligen Aufmacher bzw. Teaser Bildern. Das heißt andere Inhalte wie z.B. Bildergalerien, Infografiken, Blogartikel oder Quizze werden von der App nicht erfasst und nicht dargestellt. Hintergrund ist, die App möglichst einfach zu halten da es in der Fragestellung um die Relation zwischen den Aufmacherbildern und dem dahinter liegenden Artikeltext geht. Die App erhebt in dieser Hinsicht keinen Anspruch darauf, die gesamten redaktionellen Inhalte von ZEIT ONLINE darzustellen, sondern versteht sich eher als explorative Applikation im Sinne der Fragestellung.

Der User soll die Möglichkeit haben die standardmäßig vorhandenen Artikeltitel auszublenden um so, wenn gewünscht, einen rein visuellen Eindruck der Artikelbilder zu bekommen. Hierzu soll einen Schalter in der Menüleiste am unteren Bildrand geben. Wenn der User sich im Artikel befindet soll er die Möglichkeit haben die Schriftgröße in gewissen Maß selbst zu bestimmen, da die App eventuell auf Monitoren mit verschiedenen großen Auflösungen ausgeführt werden wird oder die Sehkraft des Users nicht mehr ausreicht um eine normal große Schrift zu erkennen und zu lesen. So wird in den Artikeln ein gewisses Maß an Barrierefreiheit gewährleistet.

Das übergeordnete Ziel der App ist es, eine rudimentäre Nachrichten Applikation für Windows 8 zu erstellen. Gleichzeitig soll eine Umgebung geschaffen werden, die es erlaubt Untersuchungen anzustellen, in wie weit es möglich ist allein durch das Betrachten der Aufmacherbilder auf den Inhalt der jeweiligen Artikel zu schließen (siehe Sektion 6.2.2 auf Seite 27). Außerdem soll die App dazu dienen, zu erforschen, wie eine Nachrichten Applikation in der Modern UI Oberfläche von Windows 8 erstellt wird und welche design- als auch funktionstechnischen Vorgaben von Microsofts vorhan-

den sind. Sprich, wie eine App mit Nachrichteninhalten nach Microsofts Sichtweise auszusehen hat.

4.2 Nativ vs. Web

Vor dem Entwickeln einer Windows 8 App muss sich entschieden werden mit welcher Technologie bzw. mit welcher Programmiersprache entwickelt werden soll. Die Rede ist hier von einer App die in der Modern UI Oberfläche von Windows 8 läuft und für diese konzipiert ist. Das heißt es handelt sich nicht um eine klassische .NET oder WIN32 Anwendung für Windows. Um eine Modern UI App zu entwickeln, müssen zwei Dinge zwingend vorhanden sein. Zum einen Windows 8 selbst und zum anderen wird die neueste Version von Visual Studio, Visual Studio 11³, benötigt. Visual Studio steht in der Express Version für Windows 8 kostenlos zur Verfügung. Des weiteren muss sich zwischen der nativen Umsetzung und der Implementierung mit Webtechnologien entschieden werden. Es soll zunächst erläutert werden was die beiden Begriffe bedeuten und in welcher Weise und welchem Zusammenhang sie bei der Entwicklung einer Windows 8 Modern UI App üblicherweise gebraucht werden.

4.2.1 Native App

Der heutige Begriff „native App“ unterscheidet sich in einigen Punkten von der früheren oder ursprünglichen Verwendung des Begriffs. Früher sprach man von einer nativen App wenn direkt auf die Ressourcen der Maschine zugegriffen wurde, wie z.B. Maschinencode der direkt von der CPU ausgeführt wird. Heute wird eine App oftmals schon als nativ deklariert, wenn es sich nicht um eine Webapp handelt. Eine sinnige Definition liegt irgendwo zwischen diesen beiden Varianten. Eine App ist dann nativ, wenn sie Geräte-, Betriebssystem- oder Laufzeitumgebungsabhängig ist. Das heißt, sie ist für ein spezielles Gerät entwickelt und kann nur auf diesem ausgeführt werden. Sie kann dabei alle Geräte- oder Betriebssystemspezifischen Funktionen nutzen, es ist jedoch egal wie nah an der Hardware tatsächlich programmiert wurde (O'Brian, 2013).

In Visual Studio können native Windows 8 Apps u.a. mit den Programmiersprachen C++, C# oder Visual Basic erstellt werden. Für das Design bzw. das Aussehen der App wird die Markupsprache - Extensible Application Markup Language (XAML) - verwendet. Es gibt zwei Möglichkeiten wie das XAML erstellt werden kann. Es kann entweder von Hand geschrieben werden oder man lässt es sich automatisch generieren. Zum automatischen Generieren lassen sich per Drag & Drop Elemente wie z.B. Buttons und andere Schaltflächen aus einer Werkzeugpalette direkt auf die „App-Leinwand“ ziehen, sowie verschieben oder nach Belieben anordnen.

³Visual Studio 2011 war die aktuellste Version beim Erstellen dieser Arbeit.

4.2.2 Windows 8 App mit Webtechnologien

4.3 Navigationskonzept

Damit der User ein für ihn angenehmes und flüssiges Nutzungserlebnis hat, ist es notwendig sich über das Navigationskonzept Gedanken zu machen, gerade wenn es sich um eine App handelt, in der es viele verschiedene Inhaltsbereiche gibt. Der User soll sich möglichst intuitiv durch die Inhalte bewegen können. Außerdem muss dem User auf der Einstiegsebene der ein guter Überblick über die vorhandenen Inhalte gegeben werden, damit er von dort aus zielgerichtet weiter navigieren kann, ohne lange suchen zu müssen. Microsoft nennt in seinen Richtlinien für Windows Store Apps (Modern UI Apps) grundsätzlich zwei Möglichkeiten wie die Navigation umgesetzt werden kann.

4.3.1 Flaches System

Das flache Navigationssystem wird in vielen Windows Store Apps verwendet, häufig in Spielen, Browern oder in Apps zum Erstellen von Dokumenten. Es zeichnet sich dadurch aus, dass sich die Inhalte auf der gleichen hierachischen Ebene befinden. Dieses System eignet sich wenn ein schneller Wechsel zwischen wenigen Seiten oder Registerkarten möglich sein soll (Microsoft, 2013a).

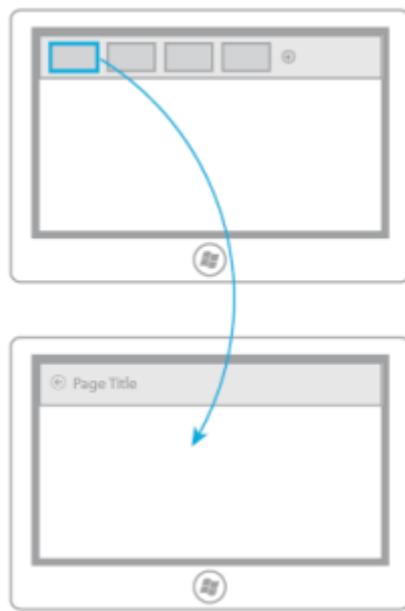


Abbildung 2: Flaches Navigationssystem (Microsoft, 2013a).

Abbildung 2 zeigt wie das flache Navigationssystem funktioniert. Am oberen Rand befindet sich eine, nicht immer sichtbare Navigationsleiste mit den verschiedenen Registerkarten oder Seiten. Die Leiste wird angezeigt, wenn der User vom unteren oder oberen Bildrand streift (siehe Sektion 2.1). Wenn der User die Seite wechseln möchte

geschieht dies direkt über die Navigationsleiste, d.h. es gibt meist keinen permanenten Zurück-Button. Ein flaches Navigationssystem eignet sich nicht unbedingt für eine Nachrichten App wie die, die in dieser Arbeit konzipiert und umgesetzt ist, weil es zu viele Bereiche (Ressorts) gibt, welche sich zudem sehr ähneln. Hier ist es angebracht ein hierarchisches Navigationssystem zu verwenden.

4.3.2 Hierarchisches System

Die meisten Windows Store-App benutzen ein hierarchisches Navigationssystem. Microsoft nennt es Hubnavigationsmuster. Es eignet sich um große Inhaltssammlungen zu ordnen und sie für User benutzerfreundlich auf zu bereiten. Der Schlüssel dazu ist die Unterteilung des Inhalts in verschiedene Detailebenen (Microsoft, 2013a).

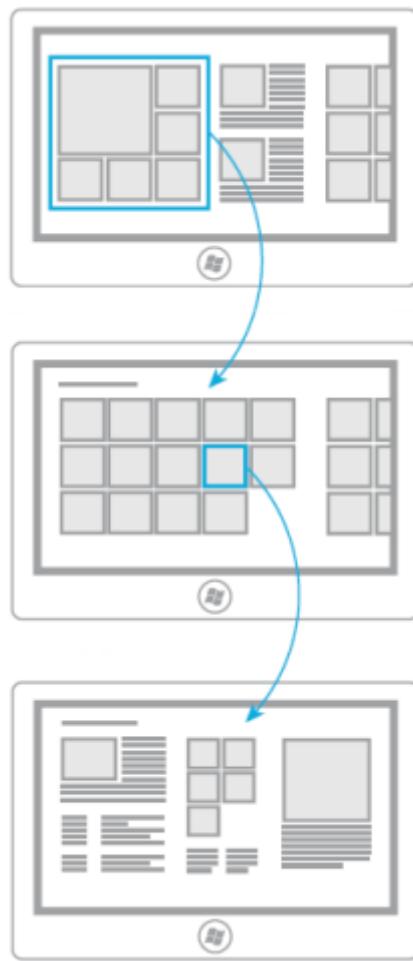


Abbildung 3: Hierarchisches Navigationssystem (Microsoft, 2013a).

Das Schema eines hierarchischen Systems ist in Abbildung 3 dargestellt. Der Einstiegspunkt in die App ist die sogenannte „Hubseite“ hier ganz oben in der Abbildung zu sehen. Auf der Hubebene werden aus den vielen großen Bereichen der App jeweils ein kleiner Teil gesammelt und dargestellt, sodass sich der User vorstellen kann was ihn im jeweiligen Bereich erwartet. Die Anordnung, in welcher der Aus-

zug der Inhalte dargestellt ist muss nicht für jeden Bereich gleich sein. Es kann das Nutzungserlebnis sogar verbessern und vielfältiger machen wenn unterschiedliche Darstellungen (z.B. in Höhe, Breite oder Anzahl der Objekte) für die Bereiche genutzt werden. Die Hubansicht wird horizontal gescrollt, d.h. weiterer Inhalt befindet sich hinter dem rechten Rand und kann entweder durch das Scrollrad der Maus oder auf einem Tablet, durch das swipen nach links in das Sichtfeld des Users gebracht werden.

Die mittlere Darstellung in Abbildung 3 zeigt die zweite Detailstufe des hierarchischen Systems. Hier werden alle Elemente eines Bereichs dargestellt. In diesem Fall werden hier alle Artikel aus einem Ressort aufgelistet.

Im unteren Bereich der Abbildung ist schließlich die letzte Detailstufe zu sehen. Es handelt sich hierbei um den eigentlichen Inhalt (z.B. einen Artikel). Es ist ebenfalls möglich von der Hubansicht direkt in die Detailansicht eines Elements zu wechseln.

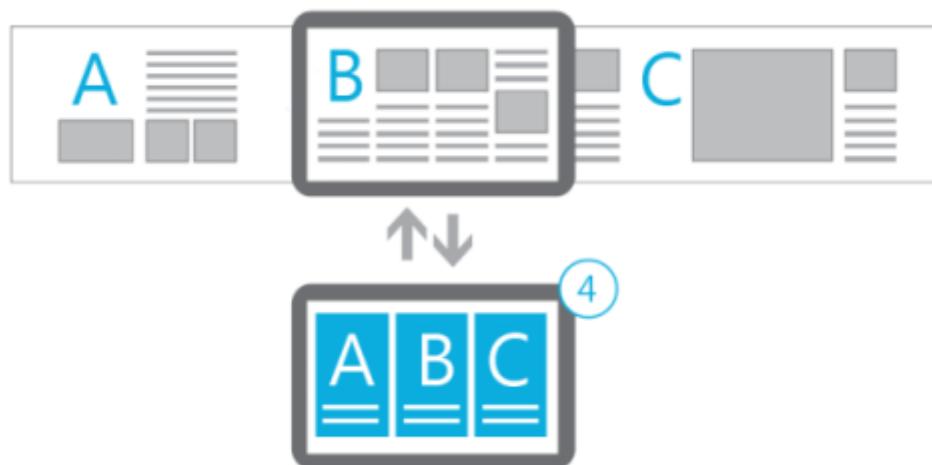


Abbildung 4: Schema des „Semantic Zoom“ (Microsoft, 2013a).

Befinden sich in der Hubansicht, trotz der Zusammenfassung der Bereiche noch zu viele Bereiche und es dauert zu lange bis zum Ende zu scrollen, kann man noch eine weitere Ebene „davor“ legen. Das Konzept nennt Microsoft „Semantic Zoom“ und ist in Abbildung 12 schematisch dargestellt. Oben in der Abbildung ist die Hubansicht dargestellt, unten die zusätzliche semantic Zoom Ebene. In der Praxis kann der User auf diese Weise die Inhalte noch weiter vereinfachen und zusammenfassen lassen. Im Fall dieser App soll es einen semantischen Zoom geben, um ein flüssigeres und schnelleres Navigieren z.B. ganz zum Ende der Hubansicht zu ermöglichen, da es über zehn verschiedene Ressorts geben wird. Diese Ansicht soll nur die Namen der verschiedenen Ressorts, sowie ein zufälliges Bild aus dem jeweiligen Ressort zeigen. Um am Desktop-PC (mit Maus) diese Ansicht aufzurufen, muss der User auf ein kleines Minuszeichen am unteren rechten Rand klicken. Am Tablet wird der semantische Zoom mit der „Pinch to Zoom (out)⁴“ Geste aufgerufen. Klickt der User auf

⁴Pinch to Zoom erklären

ein Element in dieser zusätzlichen Navigationsebene, wird er direkt zum jeweiligen Ausschnitt in die Hubansicht navigiert. Die semantic Zoom Ansicht kann von überall aus der Hubansicht aufgerufen werden.

4.4 Menü und Features

Da die App einen explorativen Charakter hat wird sich bei den Funktionen und Features auf das nötigste beschränkt. Es soll eine untere App-Leiste geben, in welcher, je nach dem in welcher Ansicht sich der User gerade befindet verschiedene Optionen angezeigt werden. Diese Leiste ist nicht dauerhaft zu sehen und kann, wenn man mit der Maus arbeitet durch einen Rechtsklick aufgerufen werden. Auf einem Tablet geschieht dies durch swipen vom oberen oder unteren Bildschirmrand.

In der Startansicht, sowie in der Ressortübersicht soll es am unteren Bildrand die Möglichkeit geben, die Titel aller Elemente aus- bzw. auch wieder einzublenden um so dem User ein rein visuelles Erleben der Artikelbilder zu ermöglichen. In der Artikelansicht soll der User die Möglichkeit haben, die Schriftgröße des Artikels zu vergrößern, zu verkleinern, sowie die Schriftgröße wieder auf ihren Startwert zu setzen.

5 Umsetzung der App

5.1 Rastervorlage⁵

Beim Erstellen einer Windows 8 Modern UI App mit Hypertext Markup Language (HTML) und Javascript bietet Microsoft von Haus aus verschiedene Vorlagen für Visual Studio, welche schon einige Grundfunktionen besitzen und sich so als guter Startpunkt für eine weiterführende App eignen. Die Vorlage die für diese App verwendet wird ist die Raster-App (engl. Grid Application) Vorlage. Sie bietet die Grundaufbau für ein hierarchisches Navigationssystem wie es in Sektion 4.3.2 auf Seite 9 beschrieben ist.

Es wird von Microsoft empfohlen, dass Windows-Store Apps, welche mit HTML und Javascript erstellt werden das single-page Navigationmodell verwenden. Das bedeutet, die Navigation erfolgt nicht über Hyperlinks, sondern die verschiedenen Inhalte werden direkt in die Seite nachgeladen (ähnlich wie ein iframe). Auf diese Weise gibt keine Sichtbare Unterbrechung beim Navigieren (der Bildschirm wird nicht kurz weiß beim Navigieren). Außerdem erzielt man auf diese Weise eine bessere Performance und die App fühlt sich mehr „app-like“ an. Die Rastervorlage verwendet das single-page Navigationsmodell (Microsoft, 2013b). Abbildung 5 zeigt die Dateistruktur des Projektes. Anschließend wird kurz erläutert wofür die einzelnen Dateien da sind.

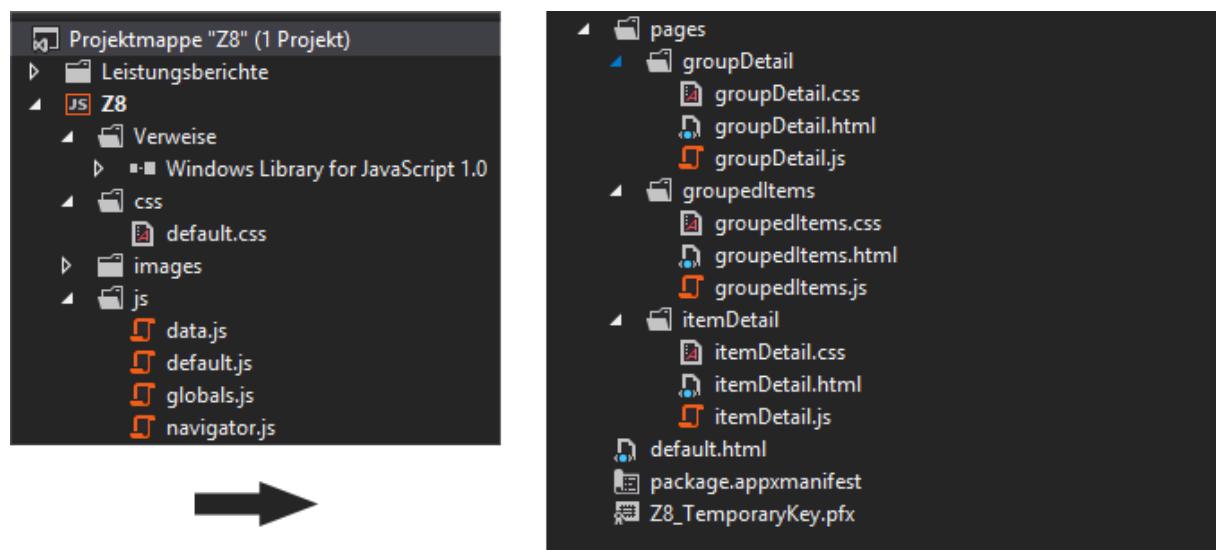


Abbildung 5: Die Projektstruktur in Visual Studio.

5.1.1 HTML Dateien

Folgende HTML Dateien sind bereits in der Raster-App Vorlage enthalten:

⁵Die Namen der verwendeten Dateien in der Rastervorlage wurden mittlerweile von Microsoft geändert.

default.html , wird als erstes geladen und enthält das HTML für den Inhaltshost (dies ist die Seite in welche die anderen Inhalte im Zuge der single-page Navigation herein geladen werden).

groupedItems.html , ist der Einstiegspunkt in die App (die Hubansicht).

groupDetail.html , zeigt alle Elemente eines Bereichs.

itemDetail.html , Einzelansicht eines Elements.

5.1.2 Javascript Dateien

Folgende Javascript Dateien sind bereits in der Raster-App Vorlage enthalten:

default.js , legt fest wie sich die App beim Start verhält.

groupedItems.js | groupDetail.js | itemDetail.js , sind die Javascript Dateien, welche das Verhalten für die gleichnamigen HTML Dateien festlegen.

navigator.js , implementiert das hierarchische Navigationssystem, sowie das single-page Navigationmodell.

data.js , stellt die benötigten Daten für den Rest der App bereit.

5.1.3 CSS Dateien

default.css , enthält Styles für Inhaltshostseite, sowie für die gesamte App.

groupedItems.css | groupDetail.css | itemDetail.css ,

Diese Vorlage wird nachfolgend angepasst und verändert, sodass sie die konzeptuierten Anforderungen erfüllt.

5.2 Datenanbindung

5.2.1 ZEIT ONLINE Datenbestand

Es soll zunächst geklärt werden wie sich der Inhalt der ZEIT ONLINE Website darstellt und wie er generiert wird. Die redaktionellen Inhalte werden fast ausschließlich über ein hauseigenes Content Management System (CMS) erstellt und gepflegt. Das CMS generiert aus den Inhalten eine Extensible Markup Language (XML)-Struktur. Aus dem XML wiederum wird anschließend mit Hilfe von Extensible Stylesheet Language Transformations (XSLT) in einer zweistufigen Transformation das fertige HTML erstellt. Dies ist ein grober Überblick wie sich die Webseite von ZEIT ONLINE zusammensetzt.

Für die Windows 8 App wird direkt das XML verwendet, welches vom CMS generiert

wird. Für die App werden zwei verschiedene Seitentypen der Webseite benötigt um die gewünschten Informationen darzustellen. Zum einen die verschiedenen „Centerpages“ und zum anderen die Artikelansicht. Centerpages sind die Hauptseiten der jeweiligen Ressorts (z.B. die Hauptseite des Politikressorts), sowie die Startseite mit den wichtigsten und aktuellsten Meldungen. Auf den Centerpages befinden sich die Teaserelemente für die verschiedenen Artikel. Die Teaserelemente bestehen meist aus einem Bild und einem kurzen, prägnanten Text, welcher kurz erläutert worum es in dem dahinter liegenden Artikel geht.

```

1 <block href="http://xml.zeit.de/digital/internet/2013-08/fablab-open-
2 hardware" year="2013" issue="34" ressort="Digital" author="Tilman Baumgä-
3 rtel" contenttype="article" publication-date="" expires="" date-last-
4 modified="2013-08-14T12:58:40+00:00" date-first-released="2013-08-14T09
5 :57:43.627551+00:00" date-last-published="2013-08-14T12
6 :59:39.691370+00:00" last-semantic-change="2013-08-14T09
7 :56:40.185797+00:00">
8 <supertitle>Open Hardware</supertitle>
9 <title>Fab Labs, die Maschinen-Bibliotheken</title>
10 <text>
11     3-D-Drucker, CNC-Fräsen oder Lasercutter - mit solchen Maschinen
12         sollen Bastler in Fab Labs experimentieren. Immer mehr solcher
13         Werkstätten entstehen nun in aller Welt.
14     </text>
15     <description>
16         3-D-Drucker, CNC-Fräsen oder Lasercutter - mit solchen Maschinen
17         sollen Bastler in Fab Labs experimentieren. Immer mehr solcher
             Werkstätten entstehen nun in aller Welt.
18     </description>
19     <byline/>
20     <image alt="MakerBot Replicator 2" align="left" title="MakerBot
21         Replicator 2" base-id="http://xml.zeit.de/digital/internet/2013-08/
22         makerbot-cebit-hannover/" type="jpg" publication-date="" expires="">
23         <bu>
24             Ein MakerBot Replicator 2 auf der diesjährigen Cebit in Hannover
25         </bu>
26         <copyright>© REUTERS/Fabrizio Bensch</copyright>
27     </image>
28 </block>
```

Listing 1: Das XML eines Teaserelements

Das XML von einem Teaserelement ist in Listing 1 dargestellt. Es werden jedoch nicht alle Informationen aus dem XML verwendet. Die verwendeten Informationen sind das „date-first-released“ (Zeile 1), der „title“ (Zeile 3), sowie die „description“ (Zeile 7) und das „image“ (Zeile 11). Das XML für die Artikelansicht ist ähnlich aufgebaut,

nur ist hier zusätzlich noch der Artikeltext (Inhalt) in Form von Paragraphen-Tags (<p>) enthalten. Dies ist ein konkreter Einblick wie die rohen Daten aussehen, welche für die App verwendet werden. Wie die Daten im Detail verarbeitet werden wird in der nächsten Sektion erläutert.

5.2.2 Daten verfügbar machen

Die Daten, die in der App dargestellt werden sollen, werden grundsätzlich in zwei Schritten geladen. Wenn die App startet werden zunächst alle benötigten Daten außer den eigentlichen Artikeln geladen. Der Artikelinhalt wird erst geladen, wenn der User ihn lesen will. Dies spart Zeit beim starten der App und der User hat somit ein flüssigeres App-Erlebnis. Der erste Schritt beim Daten Einlesen und Verarbeiten geschieht in der „data.js“ Datei. Es werden alle Teaser Elemente aller Ressorts letztendlich in einer „WinJS.Binding.List()“ gespeichert und für den weiteren Gebrauch verfügbar gemacht. Die Windows Library for JavaScript (WinJS) ist eine Javascript Bibliothek für Windows-Store Apps, die mit Javascript erstellt werden. Sie enthält nützliche Funktionen z.B. für UI Steuerelemente oder sie hilft beim XMLHttpRequest (XHR). Die Binding.List ist ebenfalls ein Teil dieser Bibliothek. Sie stellt Logik für die Datengruppierung bereit, genau in der Art und Weise wie es für diese App sinnvoll ist. Falls mit dynamischen Daten gearbeitet wird stellt sie z.B. Funktionen für die automatische Aktualisierung der Daten bereit. In Listing 2 ist die Initialisierung einer WinJS.Binding.List dargestellt.

```
1 var teaserElements = new WinJS.Binding.List();
```

Listing 2: Initialisierung der Binding-List.

Die Grundinformationen zu den einzelnen Ressorts, wie der Name und die URL, werden zunächst als normales Javascript Array festgelegt. Einen Beispieleintrag aus dem Array zeigt Listing 3.

```
1 ressorts = [
2   {
3     key: "ressort01", url: http://xml.zeit.de/index,
4     title: Top Stories, subtitle: subtitle, updated: tbd,
5     backgroundImage: tbd, articleLink: "tbd",
6     acquireSyndication: acquireSyndication, dataPromise: null
7   }
8 ]
```

Listing 3: Auszug aus dem Ressorts Array.

5 Umsetzung der App

Anschließend wird eine weitere Funktion von WinJS verwendet, die WinJS.Promises. Mit Promises kann mit asynchronen Prozessen und Datenquellen umgegangen werden. Hier werden für alle Ressorts XHRs gestartet, an die im Ressorts-Array angegebene URLs. Alle XHRs werden ebenfalls in einem Array gespeichert und erst wenn alle Promises vorhanden sind (alle URL waren erreichbar), wird die Datenverarbeitung fortgesetzt.

Mit den vorhandenen und validen XHR Responses können anschließend alle Teaserlemente in einer Schleife durchlaufen, das XML geparsst und so die nötigen Informationen für die App verfügbar gemacht werden. Listing 4 zeigt wie der Titel (Zeile 10), der Teasertext (Zeile 13) und der Bildpfad (Zeilen 15-22) aus dem XML gewonnen werden. Am Ende der Funktion werden die Daten in die in Listing 2 erstellte WinJS.Binding.List geschrieben (Zeilen 24-28). Die dargestellte Funktion ist nicht vollständig und wurde aus Übersichts- und Relevanzgesichtspunkten verkürzt dargestellt.

```

1  function getItemsFromXml(ressortXML, teaserElements, ressort) {
2      var teasers = ressortXML.querySelectorAll("region[area=lead] container >
3          block:first-child");
4      // Process each ressort teaser.
5      for (var teaserIndex = 0; teaserIndex < teasers.length; teaserIndex++) {
6          var teaser = teasers[teaserIndex];
7
8          //only articles with an image are allowed
9          if (teaser.getAttribute("contenttype") == "article" && teaser.
10             querySelector("image") != null) {
11              // Get the title
12              var teaserTitle = teaser.querySelector("title").textContent;
13
14              // Process the content so that it displays nicely.
15              var staticContent = toStaticHTML(teaser.querySelector("description
16                  ").textContent);
17
18              //Get and build the image path
19              var teaserImageEl = teaser.querySelector("image");
20              var imagebasePath = teaserImageEl.getAttribute("base-id");
21              var splitImagePath = imagebasePath.split("/");
22              var imageNameSmall = splitImagePath[splitImagePath.length - 2] + "
23                  -220x124.jpg";
24              var imageNameBig = splitImagePath[splitImagePath.length - 2] + "
25                  -540x304.jpg";
26              var imagePathSmall = imagePath + imageNameSmall;
27              var imagePathBig = imagePath + imageNameBig;
28
29              // Store the teaser element info we care about in the array.
30              teaserElements.push({
31                  group: ressort, key: ressort.title, title: teaserTitle,
32                  backgroundImage: imagePathBig, teaserText: staticContent
33              });
34      }
35  }

```

Listing 4: Parsen und speichern der Daten.

Sobald die Daten in der WinJS.Binding.List gespeichert sind können sie vom Rest der App weiterverwendet und grafisch aufbereitet werden. Es wurden bisher noch keine Artikelinhalte geladen, dies geschieht erst wenn der User auf einen Artikel klickt.

5.3 Darstellung der Daten

5.3.1 Die Hubansicht

Wenn die App startet befindet sich der User in der Hubansicht. Hier werden ihm für alle Hauptressorts die ersten sechs Artikel angezeigt, in der Reihenfolge wie sie auch auf der Webseite zu finden sind (siehe Abbildung 6).

Zu der Hubansicht gehören eine HTML Datei, in welcher das Markup festgelegt ist, eine CSS Datei, die das Layout der Seite bestimmt und eine Javascript Datei, in welcher das Verhalten (die Logik) der Hubseite programmiert wird.

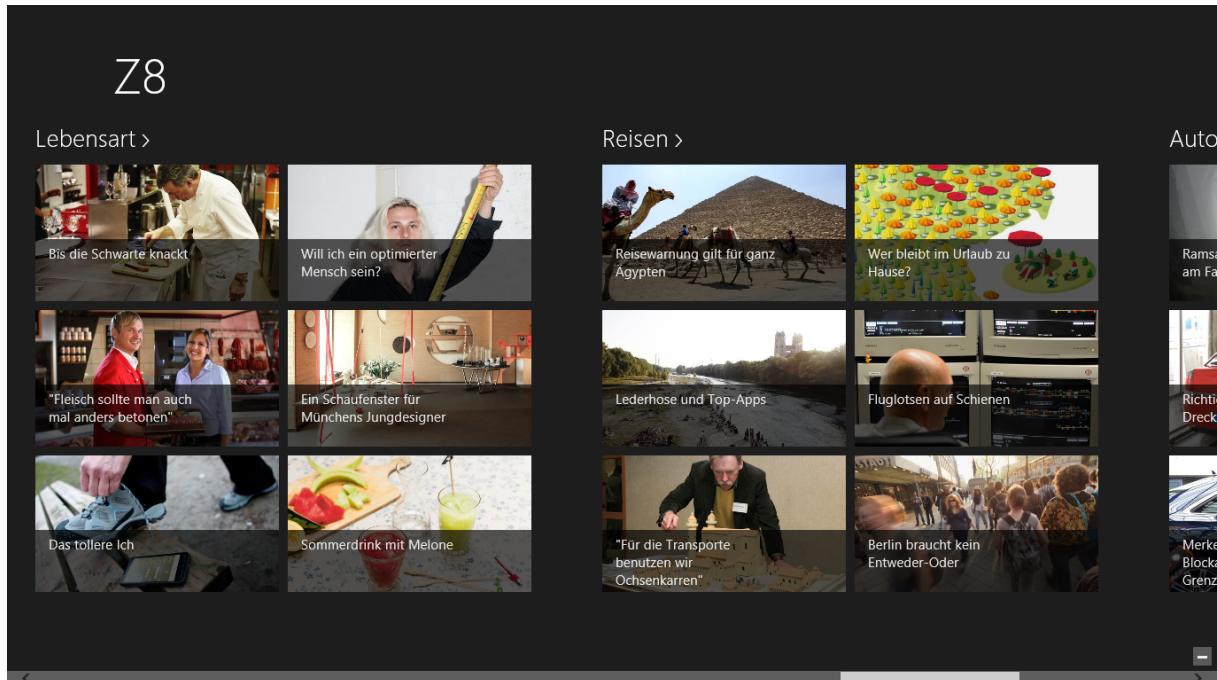


Abbildung 6: Die Hubansicht.

Um die Daten aus der `WinJS.Binding.List` auf der Hubseite anzeigen zu können, sind einige Schritte notwendig. Zunächst soll der Aufbau der HTML Datei erläutert werden. Listing 5 zeigt einen Ausschnitt aus der `groupedItems.html` Datei. Im oberen Teil befindet sich ein Template mit der Klasse `itemtemplate`, welches auf jedes einzelne Element der Hubansicht angewandt wird. Das besondere an diesem Template ist, dass es ein `data-win-control` Attribut mit dem Wert „`WinJS.Binding.Template`“ besitzt. Dies bedeutet, es akzeptiert Daten aus einer `WinJS.Binding.List`, wie sie in der Rohdatenverarbeitung genutzt wurde. Im unteren Teil ab Zeile 10 ist der Container für den eigentlichen Inhalt. Dieses Objekt besitzt ein `data-win-control` Attribut mit dem Wert „`WinJS.ListView`“. Eine `WinJS.ListView` stellt Elemente in anpassbaren Listen oder Rastern dar.

```

1 <div class="itemtemplate" data-win-control="WinJS.Binding.Template">
2   <div class="item">
3     
5     <div class="item-overlay">
6       <h4 class="item-title" data-win-bind="textContent: title"></h4>
7     </div>
8   </div>
9
10 <section aria-label="Main content" role="main">
11   <div class="groupeditemslist win-selectionstylefilled" aria-label="List
12     of groups" data-win-control="WinJS.UI.ListView" data-win-options="{
13       selectionMode: none}"></div>
14 </section>

```

Listing 5: Die wichtigsten Markupelemente der Hubansicht.

Der Rest passiert in der *groupedItems.js* Datei. Die *groupedItems.js* besitzt eine *ready()* Methode. Diese wird jedesmal aufgerufen, wenn der User zur Hubansicht wechselt oder die Hubansicht wiederhergestellt wird. Um die Daten in die *WinJS.ListView* zu bekommen wird als erstes die *WinJS.ListView* in einer Variable gespeichert und anschließend das gewünschte Template bei der ListView registriert (siehe Listing 6).

```

1 var listView = element.querySelector(".groupeditemslist").winControl;
2 listView.itemTemplate = element.querySelector(".itemtemplate");

```

Listing 6: Template bei der Listview registrieren.

Anschließend wird die *initializeLayout* Funktion aufgerufen. Hier wird am Anfang eine entscheidende Operation ausgeführt. In der *WinJS.Binding.List* sind bisher alle Teaserelemente aus allen Ressorts enthalten, für die Hubansicht sind aber nur sechs Elemente pro Ressort gewünscht. Deswegen wird eine Funktion *getClippedList()* in der *data.js* Datei aufgerufen, welche die ersten sechs Elemente jedes Ressorts zurück liefert. Damit die Elemente wie gewünscht angezeigt werden können, muss noch die *dataSource* der *WinJS.ListView* gesetzt werden, ebenso wie das Raster-Layout (siehe Listing 7). Nach diesem Schritt werden die ersten sechs Elemente jedes Ressorts wie in Abbildung 6 angezeigt.

```
1 listView.itemDataSource = groupedItemsHub.dataSource;
2 listView.layout = new ui.GridLayout({ groupHeaderPosition: "top" });
```

Listing 7: Setzen der Datenquelle und des Layouts.

5.3.2 Die Ressortansicht

Die Ressortansicht, die zweite Detailstufe zeigt alle Elemente eines Ressorts (siehe Abbildung 7). Der User gelangt hierhin wenn er in der Hubansicht auf den Ressortnamen im Headerbereich klickt.

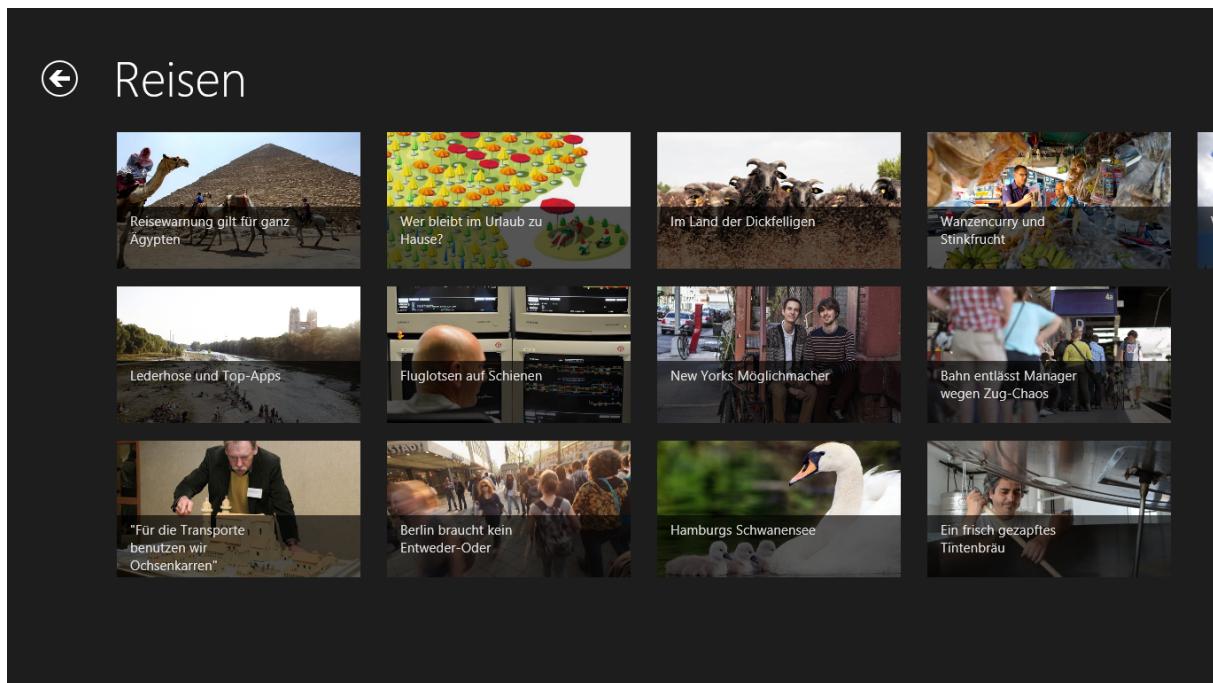


Abbildung 7: Die Einzelressortübersicht (hier das Reisen Ressort).

Die Programmlogik funktioniert analog zur Hubansicht. Es werden die gleichen Mechanismen zur Darstellung des Raster-Musters benutzt. Auch die zu dieser Ansicht gehörende HTML und Javascript Dateien ähneln denen von der Hubansicht sehr. Ein wesentlicher Unterschied ist, dass im Gegensatz zu Hubansicht nicht die Funktion `getClippedList()`, sondern stattdessen die Funktion `getItemsFromGroup()` aufgerufen wird. Sie gibt alle Elemente der übergebenen Gruppe zurück. Diese können nun mit einem `itemTemplate` und einer `WinJS.ListView` dargestellt werden.

5.3.3 Die Artikelansicht

Die Artikelansicht ist die letzte Detailstufe. Sie zeigt den eigentlichen Artikelinhalt (siehe Abbildung 8). Dargestellt wird das Ressort, der Titel, der TeaserText, das Aufmacherbild in groß, die Bildunterschrift, das Copyright für das Bild, der Artikeltext,

die Quelle des Artikels (z.B. ZEIT ONLINE oder dpa), der Autor sowie das Erstellungsdatum. Die Artikeldarstellung funktioniert etwas anders als die Hub- und der Ressortdarstellung. Es wird kein Template verwendet, stattdessen wird das HTML mit Hilfe von Javascript gefüllt. In der *ready()* Funktion der *itemDetail.js* Datei werden zunächst die bereits vorhandenen Informationen in das HTML eingefügt.



Abbildung 8: Die Artikelansicht.

Listing 8 zeigt das Einfügen von Titel, Teaser-Text und dem Bild. Anschließend fehlt noch der Artikeltext, die Quelle und der Autor. Um diese Daten zu bekommen und zu speichern ist ein weiterer XHR nötig. Der XHR wird mit der Artikel-URL aufgerufen, welche in jedem Teaser-Element gespeichert ist. Auf diese Weise bekommt man das XML der Artikel, wie es auch von der Webseite verwendet wird. Beim Parsen des Artikel-XMLs müssen drei Dinge besonders beachtet werden.

```
1 element.querySelector("article .item-title").textContent = item.title;
2 element.querySelector("article .item-subtitle").innerHTML= item.teaserText;
3 element.querySelector("article .item-image").src = item.backgroundImage;
```

Listing 8: Das Einfügen von Titel Teaser-Text und Bild.

Die Absätze des Artikels liegen in Paragraphen-Tags (`<p></p>`) vor. Es kann vorkommen, dass außer dem reinen Text, zusätzlich noch Infoboxen im Artikel vorhanden sind. Ebenso ist es möglich, dass Nachrichten vom Mikroblogging Dienst Twitter eingebunden werden. Diese Abschnitte arbeiten ebenfalls mit Paragraphen-Tags. Da Infoboxen und Twitternachrichten sich inhaltlich nicht direkt in den Artikeltext eingliedern, müssen diese Abschnitte vorher herausgefiltert werden. Des Weiteren sind

in den meisten Artikeln auf der ZEIT ONLINE Webseite Hyperlinks, entweder auf andere Artikel oder auch andere Webseiten, vorhanden. Diese sind genau wie Infoboxen und Twitternachrichten unerwünscht, da sie, wenn sie vom User aufgerufen werden, in einer anderen App (Internet Explorer) geöffnet werden würden. Dies unterbricht und stört das App-Erlebnis. Es muss dementsprechend darauf geachtet werden, dass nur der reine Artikeltext als Artikel gespeichert wird. Listing 9 verdeutlicht wie dies passiert. In Zeile 3 werden die Infoboxen und die Twitternachrichten herausgefiltert und in den Zeilen 8-13 werden die `<a>` Tags mit Hilfe regulärer Ausdrücke gelöscht. Sobald der Artikel komplett geladen ist, wird er analog zu Listing 8 in das HTML Markup eingefügt.

```

1 for (var n = 0; n < paragraphs.length; n++) {
2     //exclude info box and tweets paragraphs
3     if (paragraphs[n].parentNode.parentNode.parentNode.parentNode.nodeName
4         != "infobox" && paragraphs[n].parentNode.getAttribute("class") != "
5             twitter-tweet")){
6         var xmlText = new XMLSerializer().serializeToString(paragraphs[n]);
7         if (paragraphs[n].querySelector("a") != null) {
8             var numberofLinks = paragraphs[n].querySelectorAll("a").length;
9             //remove hyperlinks
10            for (var i = 0; i < numberofLinks; i++) {
11                var rx = new RegExp("<a[^>]+>", "i");
12                xmlText = xmlText.replace(rx, "");
13                rx = new RegExp("</a>", "i");
14                xmlText = xmlText.replace(rx, "");
15            }
16        }
17    }

```

Listing 9: Vermeidung von Infoboxen Twitternachrichten und Hyperlinks in den Artikeln.

5.4 Features

5.4.1 Artikeltitel ausblenden

Ein Hauptfeature der App ist es, dass in der Hubansicht und in der Ressortansicht die Artikeltitel ausgeblendet werden können und der User so nur noch eine Bildergewand vor sich hat. So kann er die Nachrichtenlage rein visuell erleben und erst wenn er auf einen Artikel klickt bekommt er im Artikel den Titel und Artikelinhalt zu sehen. Diese Funktion wird über eine Menüleiste am unteren Bildrand implementiert (siehe Abbildung 9). Um eine solche Menüleiste zu erzeugen genügt ein `<div>` Con-

5 Umsetzung der App

tainer in der *default.html* Datei mit einem *data-win-control* Attribut, das den Wert „WinJS.UI.AppBar“ bekommt. In diesem Element können anschließend *<button>* Elemente angelegt werden. Diese sind später die sichtbaren Schaltflächen wenn die Menüleiste aufgerufen wird.

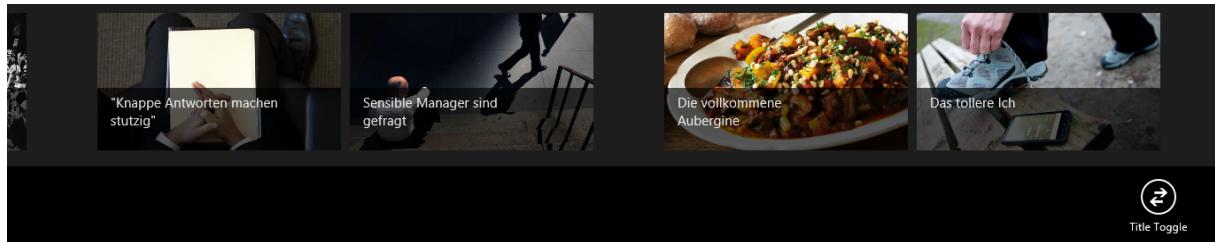


Abbildung 9: Untere Menüleiste mit der Funktion zum Artikeltitel ausschalten.

In der Initialisierungsphase der App wird in der *default.js* Datei der Click-Handler für den Button registriert. D.h. hier wird die Funktion angegeben, welche aufgerufen werden soll wenn der Button geklickt wird. In diesem Fall wird die Funktion *titleToggle()* in der *globals.js* Datei aufgerufen. In der *globals.js* Datei befinden sich Variablen und Funktionen, die von mehreren Ansichten benutzt werden und deswegen global verfügbar sein sollen. Die *titleToggle()* Funktion ist schließlich verantwortlich für das ausblenden der Artikeltitel.

Die Funktion sammelt dazu alle Elemente mit der Klasse *item-overlay* in einem Array und setzt anschließend für alle gefundenen Elemente die CSS Eigenschaft *display* auf „none“. Das Resultat ist in Abbildung 10 zu sehen.

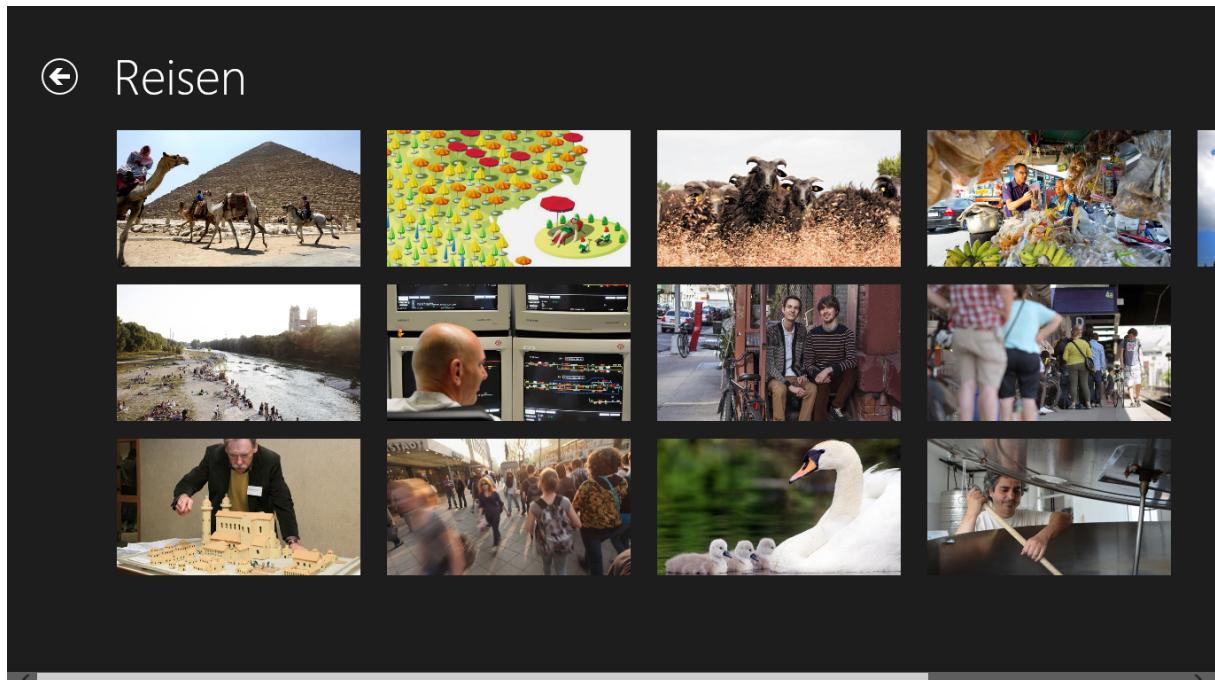


Abbildung 10: Das Reisen Ressort ohne Artikeltitel.

5.4.2 Schriftgröße verändern

In der Artikelansicht hat der User die Möglichkeit, die Schriftgröße anzupassen. Es sind jeweils zwei Stufen größer, als auch zwei Stufen kleiner als die normale Größe möglich. Der Ansatz ist grundsätzlich genau so wie bei der „Titel-ausblenden“ Funktionalität. Zunächst wird ein Button in der unteren Menüleiste angelegt. Es wäre auch möglich zwei Buttons zu erstellen, einen für größer und einen für kleiner. Um die Entwicklungsmöglichkeiten jedoch weiter auszu reizen gibt es in der App nur einen Button in der Menüleiste. Wird dieser geklickt erscheint noch ein weiteres Menü, ein sogenanntes *Flyout-Menü*. In diesem sind schließlich die Schaltflächen für „Schrift größer“, „Schrift kleiner“ und für „Schriftgröße zurücksetzen“. Ist die Schriftgröße z.B. auf der größten Stufe eingestellt, wird im Menü, der größer Button nicht mehr dargestellt. Abbildung 11 zeigt die verschiedenen Stati, die das Schriftgrößenmenü haben kann.

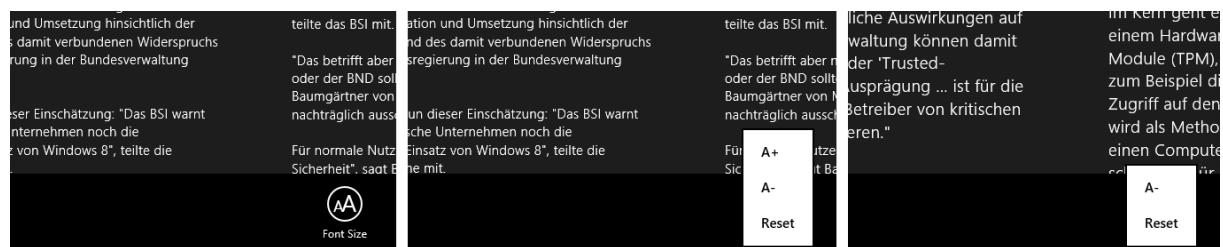


Abbildung 11: links: Schriftgröße Button im unteren Menü | mitte: Das Flyout-Menü mit allen Schaltflächen. | rechts: Das Flyout-Menü beim Aufruf wenn schon die größte Größe eingestellt ist.

5.4.3 Semantic Zoom

Das semantic Zoom Feature wird eingesetzt, wenn es in der Hubansicht sehr viele Bereiche gibt. Es soll dem User ermöglichen, ohne langes scrollen, zu den verschiedenen Bereichen in der Hubansicht zu springen. Beim Aufruf des semantic Zooms wird eine weitere Ebene über die Hubansicht gelegt. Typischer Weise werden in dieser Ebene nur die Verschiedenen Bereiche, ohne inhaltliche Elemente angezeigt. Bei ZEIGHT werden hier alle Ressorts mit dem jeweiligen Ressortnamen und einem zufälligen Bild aus dem Ressort angezeigt (siehe Abbildung 12). Diese Ansicht ist ebenfalls horizontal scrollbar, im direkten Vergleich zur Hubansicht jedoch viel kompakter und übersichtlicher. Klickt der User z.B. auf das Reisen Ressort, wird wieder zurück zur Hubansicht „gezoomt“. Der User hätte dann die Ansicht wie sie Abbildung 6 auf Seite 18 zeigt. Das bedeutet es findet in beide Richtungen ein kontextsensitives Zoomen statt. Je nach dem wo sich der User in der Hubansicht befindet bzw. auf welches Ressort der User in der semantic Zoom Ebene klickt, wird an die entsprechende Stelle raus- bzw. reingezoomt.

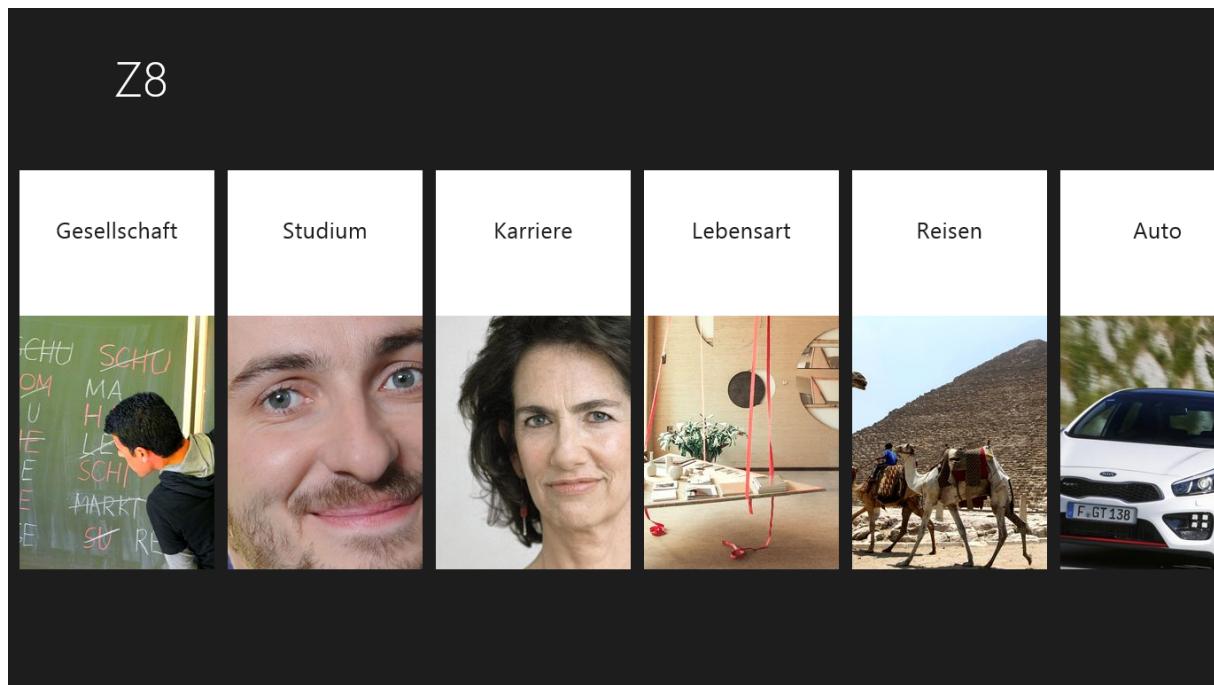


Abbildung 12: Die semantic Zoom Ansicht.

Da der semantic Zoom von der Hubansicht aus aufgerufen wird und mit ihr interagiert, ist er dementsprechend in der `groupedItems.html` und in der `groupedItems.js` implementiert. Genau wie für die Hubansicht gibt es auch für den semantic Zoom ein Template für die einzelnen Elemente. Damit eine Interaktion zwischen der Hubansicht und dem semantic Zoom stattfinden kann, wird ein Wrapper `div` Element um die `WinJS.ListView` der Hubansicht und die `WinJS.ListView` der semantic Zoom Ebene gezogen. Listing 10 zeigt wie der Wrapper eingebunden ist. Das `div` Element besitzt ein `data-win-control` Attribut mit dem Wert „`WinJS.UI.SemanticZoom`“. Des weiteren kann der `zoomFactor` und der initiale Zoomstatus festgelegt werden. Bei der ZEIGHT App ist dieser Zoomstatus initial auf `false`, da der User die Hubansicht als Einstiegsplatz in die App haben soll (nicht die semantic Zoom Ansicht).

```

1 <div class="semanticZoom" data-win-control="WinJS.UI.SemanticZoom" data-win-
2   -options="{ zoomFactor: 0.5, initiallyZoomedOut: false }" style="height:
3     100%">
4   <div class="groupeditemslist win-selectionstylefilled" aria-label="List
5     of groups" data-win-control="WinJS.UI.ListView" data-win-options="{
6       selectionMode: none}"></div>
7   <div class="groupeditemslistZoomOut groupeditemslist" aria-label="List
8     of groups" data-win-control="WinJS.UI.ListView" data-win-options="{
9       selectionMode: none}"></div>
10 </div>
```

Listing 10: Der Semantic Zoom Wrapper.

5.5 Entwicklungschronologie

6 Evaluierung

6.1 Fokusgruppe

6.2 Interview

6.2.1 Intuitivität der App

6.2.2 Der nur Bilder Modus

6.3 Ergebnis

7 Fazit

Abkürzungsverzeichnis

XAML	Extensible Application Markup Language	7
CMS	Content Management System	13
XML	Extensible Markup Language	13
XSLT	Extensible Stylesheet Language Transformations	13
HTML	Hypertext Markup Language	12
XHR	XMLHttpRequest	15
WinJS	Windows Library for JavaScript	15

Literatur

[Microsoft 2013a] MICROSOFT: *Navigationsdesign für Windows Store-Apps.* <http://msdn.microsoft.com/de-de/library/windows/apps/hh761500.aspx>. Version: 2013. – zuletzt geprüft: 17.08.2013

[Microsoft 2013b] MICROSOFT: *Quickstart: Using single-page navigation (Windows Store apps using JavaScript and HTML).* <http://msdn.microsoft.com/en-us/library/windows/apps/hh452768.aspx>. Version: Juni 2013. – zuletzt geprüft: 20.08.2013

[O'Brian 2013] O'BRIAN, Tim: *Web, native, and déjà vu.* <http://channel9.msdn.com/Blogs/Vector/Web-native-and-dj-vu>. Version: Januar 2013. – zuletzt geprüft: 29.07.2013

Abbildungsverzeichnis

1	Der Startbildschirm von Windows 8.	5
2	Flaches Navigationssystem (Microsoft, 2013a).	8
3	Hierarchisches Navigationssystem (Microsoft, 2013a).	9
4	Schema des „Semantic Zoom“ (Microsoft, 2013a).	10
5	Die Projektstruktur in Visual Studio.	12
6	Die Hubansicht.	18
7	Die Einzelressortübersicht (hier das Reisen Ressort).	20
8	Die Artikelansicht.	21
9	Untere Menüleiste mit der Funktion zum Artikeltitel ausschalten.	23
10	Das Reisen Ressort ohne Artikeltitel.	23
11	links: Schriftgröße Button im unteren Menü mitte: Das Flyout-Menü mit allen Schaltflächen. rechts: Das Flyout-Menü beim Aufruf wenn schon die größte Größe eingestellt ist.	24
12	Die semantic Zoom Ansicht.	25

Listings

1	Das XML eines Teaserelements	14
2	Initialisierung der Binding-List.	15
3	Auszug aus dem Ressorts Array.	15
4	Parsen und speichern der Daten.	17
5	Die wichtigsten Markupelemente der Hubansicht.	19
6	Template bei der Listview registrieren.	19
7	Setzen der Datenquelle und des Layouts.	20
8	Das Einfügen von Titel TeaserText und Bild.	21
9	Vermeidung von Infoboxen Twitternachrichten und Hyperlinks in den Artikeln.	22
10	Der Semantic Zoom Wrapper.	25