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 xBox 4
2.4.3 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 Webapp 8
4.2.3 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 12
5.1.3 CSS Dateien 13
5.2 Datenanbindung 13
5.2.1 ZEIT ONLINE Datenbestand 13
5.2.2 Datenverarbeitung 14
5.3 Darstellung der Daten 16
5.4 Wichtiger Code 16

6 Evaluierung **17**

6.1 Fokusgruppe 17
6.2 Interview 17
6.2.1 Intuitivität der App 17
6.2.2 Der nur Bilder Modus 17

6.3 Ergebnis 17

7 Fazit **18**

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-Computer[1] 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-Menü[2]. 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 trotzdem auch 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

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 xBox

2.4.3 Windows Phone

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

[2] Auch bekannt als Menüband oder Multifunktionsleiste.

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 verschieden 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 17). 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 11[3], 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.

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

4.2.2 Webapp

4.2.3 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, Browsern 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).

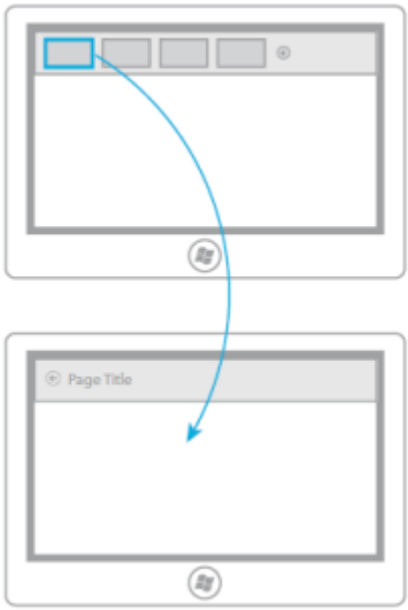

Abbildung 1: Flaches Navigationssystem (Microsoft, 2013a).

Abbildung 1 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).

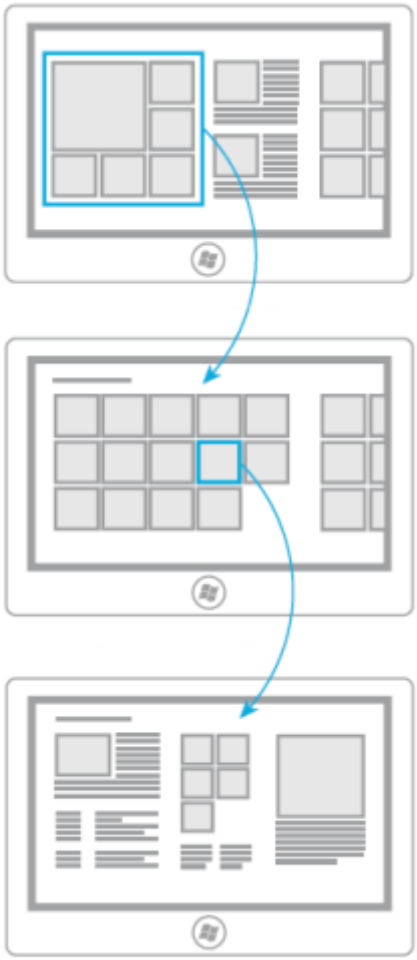

Abbildung 2: Hierarchisches Navigationssystem (Microsoft, 2013a).

Das Schema eines hierarchischen Systems ist in Abbildung 2 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 Auszug 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 2 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.

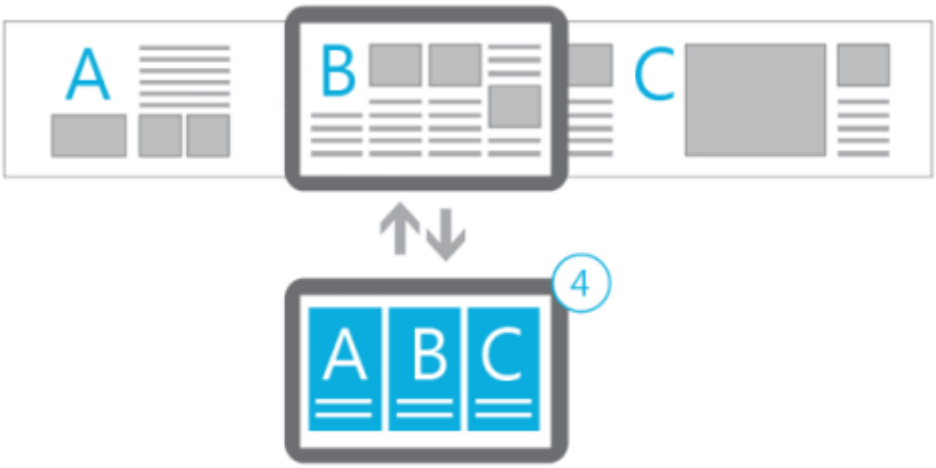

Abbildung 3: Schema des „Semantic Zoom“ (Microsoft, 2013a).

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 3 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 seman-

tische Zoom mit der „Pinch to Zoom (out)[4]“ Geste aufgerufen. Klickt der User auf 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.

[4] Pinch to Zoom erklären

5 Umsetzung der App

5.1 Rastervorlage[5]

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 Grundarchitektur 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).

5.1.1 HTML Dateien

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

default.html , wird als erstes geladen und enthält das HTML für den Inhalthost (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.

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

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 konzeptionierten 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-
     hardware" year="2013" issue="34" ressort="Digital" author="Tilman Baumgä
     rtel" contenttype="article" publication-date="" expires="" date-last-
     modified="2013-08-14T12:58:40+00:00" date-first-released="2013-08-14T09
     :57:43.627551+00:00" date-last-published="2013-08-14T12
     :59:39.691370+00:00" last-semantic-change="2013-08-14T09
     :56:40.185797+00:00">
2    <supertitle>Open Hardware</supertitle>
```

```
3     <title>Fab Labs, die Maschinen-Bibliotheken</title>
4     <text>
5        3-D-Drucker, CNC-Fräsen oder Lasercutter - mit solchen Maschinen
             sollen Bastler in Fab Labs experimentieren. Immer mehr solcher
             Werkstätten entstehen nun in aller Welt.
6     </text>
7     <description>
8        3-D-Drucker, CNC-Fräsen oder Lasercutter - mit solchen Maschinen
             sollen Bastler in Fab Labs experimentieren. Immer mehr solcher
             Werkstätten entstehen nun in aller Welt.
9     </description>
10    <byline/>
11    <image alt="MakerBot Replicator 2" align="left" title="MakerBot
          Replicator 2" base-id="http://xml.zeit.de/digital/internet/2013-08/
          makerbot-cebit-hannover/" type="jpg" publication-date="" expires="">
12       <bu>
13          Ein MakerBot Replicator 2 auf der diesjährigen Cebit in Hannover
14       </bu>
15       <copyright>© REUTERS/Fabrizio Bensch</copyright>
16    </image>
17 </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 Datenverarbeitung

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: "tdb",
6       acquireSyndication: acquireSyndication, dataPromise: null
7    }
8 ]
```

Listing 3: Auszug aus dem Ressorts Array.

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 Teaserelemente in einer Schleife durchlaufen, das XML geparst 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 >
         block:first-child");
3    // Process each ressort teaser.
4    for (var teaserIndex = 0; teaserIndex < teasers.length; teaserIndex++) {
5       var teaser = teasers[teaserIndex];
6
```

```
7       //only articles with an image are alllowed
8       if (teaser.getAttribute("contenttype") == "article" && teaser.
            querySelector("image") !== null) {
9           // Get the title
10          var teaserTitle = teaser.querySelector("title").textContent;
11
12          // Process the content so that it displays nicely.
13          var staticContent = toStaticHTML(teaser.querySelector("description
                ").textContent);
14
15          //Get and build the image path
16          var teaserImageEl = teaser.querySelector("image");
17          var imageBasePath = teaserImageEl.getAttribute("base-id");
18          var splitImagePath = imageBasePath.split(/);
19          var imageNameSmall = splitImagePath[splitImagePath.length - 2] + "
                -220x124.jpg";
20          var imageNameBig = splitImagePath[splitImagePath.length - 2] + "
                -540x304.jpg";
21          var imagePathSmall = imageBasePath + imageNameSmall;
22          var imagePathBig = imageBasePath + imageNameBig;
23
24          // Store the teaser element info we care about in the array.
25          teaserElements.push({
26              group: ressort, key: ressort.title, title: teaserTitle,
27              backgroundImage: imagePathBig, teaserText: staticContent
28          });
29      }
30  }
31  }
```

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

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

5.4 Wichtiger Code

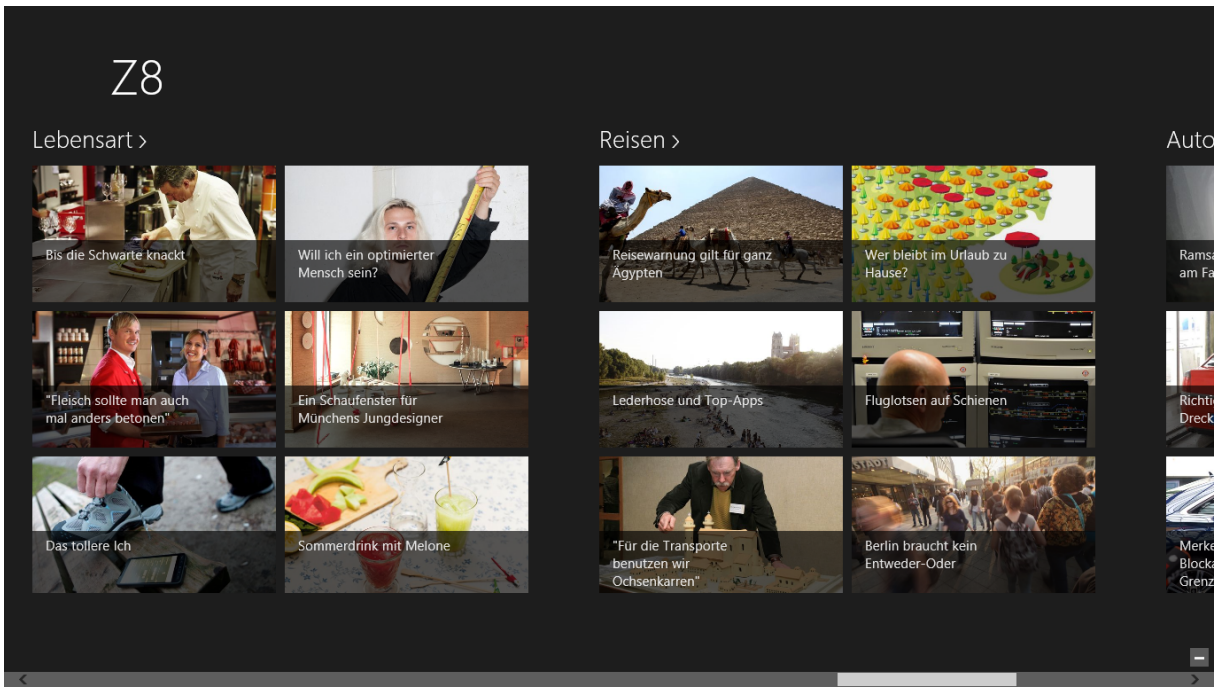

Abbildung 4: Ressortübersicht in Startansicht.

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 14
WinJS Windows Library for JavaScript 14

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 Flaches Navigationssystem (Microsoft, 2013a). 8
2 Hierarchisches Navigationssystem (Microsoft, 2013a). 9
3 Schema des „Semantic Zoom“ (Microsoft, 2013a). 10
4 Ressortübersicht in Startansicht. 17