Unterschiede in der Entwicklung

Windows 8 App – Windows Phone 8 App

Zweite Bachelorarbeit

Bachelor-Studiengang Medientechnik
Fachhochschule St. Pölten

Ausgeführt von:
Marcus Honkisz
mt101042

Betreuer: Dipl.-Ing. (FH) Klaus Temper

Gablitz, 13.08.2013

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Bachelorarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.

- ich dieses Bachelorarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/ einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

- diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

....

Ort, Datum Unterschrift

Abstract

This thesis examines the statement: “How big are the differences between an Windows 8 Desktop and an Windows Phone 8 App for web developers?”

Developer Ben Riga, who works for Microsoft said about the development of an app for both platforms: “*There is also no way for you to build an app that runs on both Windows Phone and Windows 8. That is not a concept and so hopefully you’re not expecting to do so. It’s just not possible at this point”.* Moreover it’s also not possible to create a single project in which an app for both platforms can be developed. Currently Microsoft works on a solution to convergence this both platforms.

The user expects an uniform user experience on all Windows 8 hardware. Because both platforms are based on the same kernel it is possible to use a PLC (=Portable Class Library) to share code between them. Nevertheless developers have to create two different projects, because every platform has different namespaces and controls. On Windows 8 it is possible use Contracts and Extensions whereby under Windows Phone 8 the developer cannot use these concepts, but has to utilize Choosers and Launchers. That’s why for example a camera function must be implemented in different ways when developing an app for distinct platforms.

Although the design of the Windows 8 Modern UI of both platforms looks very similar it is possible to create different Manifest settings and there are some differences concerning programming. Other differences are within the navigation models, lifecycles, storage management, background tasks and the structure of the (API)-Architecture. Some APIs can be used on both platforms whereas others are platform dependent.

In summary it can be said that there are huge differences between the development of an Windows 8 app and an Windows Phone 8 app. If a developer creates apps for only one platform it’s relatively easy. Nevertheless if he wants to build apps for both platforms with code which is suitable for cross-platform use he needs special Knowhow.

Kurzfassung

Diese Arbeit beschäftigt sich mit der Fragestellung wie „Wie groß ist der Unterschied zwischen einer Windows 8 Desktop und einer Windows 8 Phone App für Webentwickler?“

Wie der Entwickler Ben Riga von Microsoft in einem Video über die gemeinsame Erstellung einer App für beide Plattformen sagt: „*There is also no way for you to build an app that runs on both Windows Phone and Windows 8. That is not a concept and so hopefully you’re not expecting to do that. It’s just not possible at this point.“* Es ist also nicht möglich eine App für beide Plattformen mit einem Projekt zu erstellen. Man arbeitet aber daran die beiden Plattformen so gut wie möglich miteinander zu vereinen.

Benutzer erwarten eine einheitliche Bedienung auf ihren Windows 8 Geräten, der gemeinsame Kern ermöglicht es mittels einer PLC (=Portable Class Library) Codeteile auf beiden Plattformen verwenden zu können. Jedoch muss man für die Benutzeroberfläche auf jeden Fall zwei getrennte Projekte anlegen, da diese verschieden aufgebaut sind aufgrund verschiedener Namespaces und Controls. Während es bspw. unter Windows 8 Contracts und Extensions gibt, kommen unter Windows Phone 8 andere Konzepte wie Choosers und Launchers zum Einsatz. Somit ist zum Beispiel die Kamerafunktion unterschiedlich zu integrieren.

Auch das Design beider Plattformen für die Windows Modern UI Kachel sieht zwar ähnlich aus, haben aber unterschiedliche Manifest Einstellungen und Möglichkeiten bezüglich der Programmierung. Weitere Unterschiede liegen in den Navigationsmodellen, im Lebenszyklus, im Speichermanagement, in den Hintergrundaufgaben und der (API)-Architektur. Einzelne APIs stehen beiden Plattformen zur Verfügung, die meisten jedoch sind plattformabhängig.

Zusammengefasst sind die Unterschiede zwischen beiden Plattformen schon nicht zu missachten. Möchte man eine App für eine Plattform erstellen ist dies ziemlich leicht möglich, möchte man diese plattformunabhängig entwickeln, muss man genügend Überlegungen anstellen und sich mit beiden Plattformen sehr gut auskennen, um plattformübergreifende Wiederverwendung von Code zu ermöglichen.

Inhaltsverzeichnis

1 Einleitung

Microsoft hat die Preview von Windows 8.1 veröffentlicht, im Windows Store gibt es mittlerweile schon einige Apps (Applikationen) zum Download. Einige Bücher zur Entwicklung von Windows 8 und Windows Phone 8 Apps für die unterschiedlichen Programmiersprachen sind erschienen und auch die Online Dokumentation von Microsoft ist ordentlich gewachsen und Großteils bereits in einigen Sprachen verfügbar.

Warum gibt es aber unterschiedliche Bücher zu Programmierung einer Applikationen für Windows 8 und für Windows Phone 8? Verwenden sie die gleichen Programmiersprachen? Kann man eine bereits mit HTML5/JS erstellte App für Windows 8 problemlos auch am Windows Phone 8 verwenden?

Die Forschungsfrage dieser Arbeit lautet deswegen ganz einfach: „Wie groß ist der Unterschied zwischen einer Windows 8 Desktop und einer Windows 8 Phone App für Webentwickler?“ und der Titel der Arbeit „Programmierungsunterschiede Windows 8 – Windows Phone 8“.

Da es ja für Entwickler ganz einfach gestaltet worden ist von Microsoft Applikationen zu programmieren, warum sollte dann ein grober Unterschied zwischen Windows 8 und Windows Phone 8 sein? Widerspricht das nicht eigentlich dem Konzept von Microsoft?

Dies sind alles Fragen, mit welchen sich diese Arbeit beschäftigt. Die Motivation dieser Arbeit beruht darauf, dass dies noch immer ein sehr offener und chancenreicher Bereich der Webprogrammierung ist und Erfahrungen und Wissen darin nur von Vorteil sein können.

Als wissenschaftliche Forschungsmethode wurde Recherche angewendet, sowie durch die Programmierung eigener kleiner Beispiele (Apps) die recherchierten Ergebnisse auch praktisch angewendet und getestet.

Diese Arbeit soll die Gemeinsamkeiten und Unterschied der beiden Plattformen erläutern, um Entwicklern zu helfen, einschätzen zu können was möglich ist und was nicht, bevor sie mit der Konzeption und der Entwicklung beginnen.

Im folgenden Kapitel werden kurz die verfügbaren Windows 8 Versionen vorgestellt, unter anderem auf welchen Geräten diese lauffähig sind und mit welchen Versionen Windows 8 oder Windows Phone 8 Apps erstellt werden können.

Im Kapitel darauf werden die Unterschiede und Gemeinsamkeiten einer Windows 8 und Windows Phone 8 App gezeigt. Dazu wurde es in drei Unterkapiteln Aufbau/Struktur, Design und Programmierung unterteilt.

Unter Aufbau/Struktur wird die App selbst einmal vorgestellt, ihr Aufbau, die Struktur, die verschiedenen Navigationsmodelle und Lebenszyklen. Im Bereich Design/Usability werden die allgemeinen Designprinzipien, die Form vorgestellt sowie etwas auf User Experience eingegangen.

Der größte Punkt umfasst die Programmierung einer App unter beiden Plattformen, hier werden Themen wie die (API)-Architektur, Asynchronität, verschiedene Namespaces und Steuerelemente, Speichermanagement, Hintergrundaufgaben und Contracts/Extensions erläutert. Auch die Kacheln werden vorgestellt und die Möglichkeiten, Code für beide Plattformen verwenden zu können.

Im Kapitel 4 wird kurz die gemeinsame Entwicklungsumgebung Visual Studio Professional 2013 Preview und das Designtool Blend vorgestellt. Auch die bürokratischen Aufwände um eine Entwicklerlizenz und einen Entwickler Account zu bekommen werden kurz behandelt.

Anschließend im Kapitel Fazit folgt eine Zusammenfassung der Ergebnisse die mittels dieser Arbeit eruiert und erforscht wurden.

1.1 Begriffserklärungen

Hier werden nun einige Begriffe erklärt, welche in der Arbeit häufig verwendet werden und deren Verständnis beim Lesen von Vorteil ist:

API (= Application Programming Interface), hierbei handelt es sich um eine Programmierschnittstelle, mit welcher ein Programmteil einer Software zur Verfügung gestellt wird. Im weiteren Sinne kann selbst eine Bibliothek (Library) als API gesehen werden.

Namespace, dienen zur Strukturierung von Klassen. Eine Jede Klasse hat immer einen Namespace, welchen man bei der Programmierung vorher einbinden muss. Bspw. in C# erfolgt dies mit dem Schlüsselwort `using`. JavaScript (JS) kennt das Konzept der Namespaces nicht, jedoch kann man hier Abhilfe schaffen, indem man Funktionen in Variablen zusammenfasst.

C#, ist eine Programmiersprache von Microsoft aus dem .NET Framework. Mit Windows 8 ist auch die Version 5.0 erschienen.

.NET, ist eine von Microsoft entwickelte Software Plattform zur Entwicklung und Ausführung von Anwendungen. Sie besteht aus einer Laufzeitumgebung sowie einer Sammlung verschiedener Bibliotheken (WinJS), APIs und Services. Für Windows 8 wurde diese komplett überholt, um die neue Windows Runtime zu integrieren.

JS (= JavaScript), ist eine Skriptsprache mit welcher dynamisch der HTML Inhalt verändern lässt.

HTML5 (= Hypertext Markup Language in Version 5), ist eine Auszeichnungssprache für Webinhalte.

XAML (= Extensible Application Markup Language), ist eine von Microsoft entwickelte Auszeichnungssprache.

App (= Application), ist eine Anwendung, die vorerst für Mobilgeräte entwickelt wurden. Dabei handelt es sich um kleine Programme, die für den Benutzer einen Nutzen haben.

Thread (=Teilprozess), enthält verschiedene Anweisungen, die an den Prozessor übergeben werden. Wenn ein Programm aus mehreren Threads besteht, ist es möglich währenddessen einer ausgeführt wird, in einem anderen zu arbeiten. Die Oberfläche bleibt dabei weiter ansprechbar, was bei einer Anwendung mit nur einem Thread nicht möglich wäre.

2 Windows 8 Versionen

In diesem Kapitel werden kurz die verschiedenen Windows 8 Versionen vorgestellt, welche es am Markt für die verschiedenen Geräte gibt.

In der Praxis wird immer von nur einer Windows 8 Version gesprochen, doch es gibt einige Unterschiede zwischen Windows 8, Windows RT und Windows Phone 8.

2.1 Windows 8

Windows 8 für den Desktop bzw. Tablets mit x86-Architektur (Intel oder AMD Prozessoren) gibt es in 4 verschiedenen Versionen:

- Windows 8 (Standard)
- Windows 8 Professional
- Windows 8 Enterprise (nicht im freien Handel erhältlich – nur für Großkunden verfügbar)

Auf diesen Versionen ist es möglich Windows 8 Apps für Windows 8 und Windows Phone 8 zu entwickeln, da diese noch über den herkömmlichen Windows Desktop verfügen und so noch Win 32/.NET-Anwendungen ausführen können, dazu zählen die Versionen von Visual Studio 2012. Auch der Windows Store ist auf allen diesen Versionen erreichbar. (Bleske, 2013, p. 16)

2.2 Windows Server 2012

Beim Windows Server 2012 handelt es sich ebenso um Windows 8, nur bietet diese erweiterte Serverfunktionen an. Die Programmierung einer App auf diesem System ist selbstverständlich ebenso möglich.

2.3 Windows RT

Die Windows RT Version ist wie Windows 8 Enterprise nicht im freien Handel erhältlich. Es handelt sich um eine Variante, welche für Geräte mit ARM

(= Advanced RISC Machines) - Prozessoren bestimmt ist und nur auf Endgeräten aus dem Handel vorinstalliert ist. (Bleske, 2013, p. 16)

Diese Windows Version verfügt zwar ebenso noch über den herkömmlichen Desktop, jedoch in einer abgespeckten Version und durch die ARM Architektur laufen auch die bisherigen „alten“ Win 32/.NET-Anwendungen nicht mehr bzw. müssten neu übersetzt werden.

Es können nur Apps aus dem Windows Store bezogen werden, somit eignet sich diese Version nicht für die Entwicklung von Windows 8 bzw. Windows Phone 8 Apps.

2.4 Windows Phone 8

Windows Phone 8 basiert auf demselben Windows NT-Kernel wie Windows RT und Windows 8. Es handelt sich also um eine an das Mobiltelefon angepasste Version ohne den herkömmlichen Desktop und es können auch nur Apps aus dem Windows Store bezogen werden.

Da die Windows 8 Tablets mit Intel Prozessoren ausgerüstet sind und die OS-Architektur portierbar ist, ist es laut Greg Sullivan, der bei Microsoft als Senior Produkt Manager für Windows Phone tätig ist, möglich, dass auch das Windows Phone 8 mit Intel Prozessoren ausgerüstet werden können. Mit ZTE, Motorola, Lenovo und Lava bieten bereits einige Hersteller Android-Smartphones mit einem Intel Atom"Medfield" x86-Prozessor an. (Hamblen, 2013) (Quandt, 2013)

Die Entwicklung von Apps ist auf einem Windows Phone ebenso nicht möglich, selbst zum Testen einer App wird es nicht benötigt, da die Entwicklungs-umgebung Visual Studio einen eigenen Windows Phone Simulator integriert hat.

3 Unterschiede/Gemeinsamkeiten

Windows 8 und Windows Phone 8 beruhen auf 2 verschiedenen Plattformen, welche bei Microsoft von zwei verschiedenen getrennten Teams entwickelt wurden, da die Anforderungen einfach zu unterschiedlich sind. Das Windows Phone Team hatte damals mit Windows Phone 7 den Grundstein gelegt. Auf diesen haben die Windows 8 Entwickler aufgebaut. Umgekehrt wurden auch Entwicklungen des Windows 8 Teams im Windows Phone 8 integriert.

Man ist auf dem Weg, beide Plattformen so gut wie möglich zu vereinen, dies ist nur nicht sogleich auf den ersten Blick ersichtlich. Man muss für beide Plattformen eine eigene App/Projekt erstellen. Es ist nicht möglich ein Projekt anzulegen und dieses dann für Windows 8 und für Windows Phone 8 zu exportieren.

Jedoch können mittels verschiedener Möglichkeiten Funktionalitäten für beide Apps durch Portable Klassenbibliotheken (Portable Class Libraries, PCLs), Komponenten für Windows-Runtime (WinRT) und für Windows Phone-Runtime sowie die Visual Studio-Option „Als Link hinzufügen“ zur Verfügung gestellt werden.

Es gibt zwar Übereinstimmungen zwischen beiden Plattformen wie bspw. die Kacheln, jedoch muss separat die Benutzeroberfläche entworfen werden.

Durch Entwurfsmuster wie Model-View-ViewModel (MVVM) oder Model-View-Controller (MVC), welche die Trennung von Design und Code unterstützen, ist die Verwendung gemeinsamer Code einfacher. (Holland, 2013)

Eine für beide Plattformen entwickelte App unterstützt sowohl Geräte mit Intel Prozessoren als auch mit ARM Prozessoren. (Intel Software, 2013)

Diese Arbeit geht auf die gröbsten Unterschiede ein, denn Details zu den einzelnen Punkten findet man in diversen Online Ressourcen von Microsoft:

- http://msdn.microsoft.com
- http://channel9.msdn.com
- http://www.microsoft.com/germany/msdn/academic/default.aspx
- http://developer.windowsphone.com
- http://msdn.microsoft.com/de-de/magazine

3.1 Aufbau/Struktur

3.1.1 Struktur

3.1.1.1 Windows 8

Eine Windows 8 App besteht aus mehreren Elementen als nur dem Hauptinhalt, wie die App-Leiste, diese soll Navigationselemente, Befehle und Tools beinhalten. Sie ist standardmäßig ausgeblendet und kann entweder über einen „Rechtsklick“ oder, wenn der Benutzer den Finger vom oberen oder unteren Bildschirmrand aus über den Bildschirm zieht eingeblendet werden. (Microsoft, 2013a)

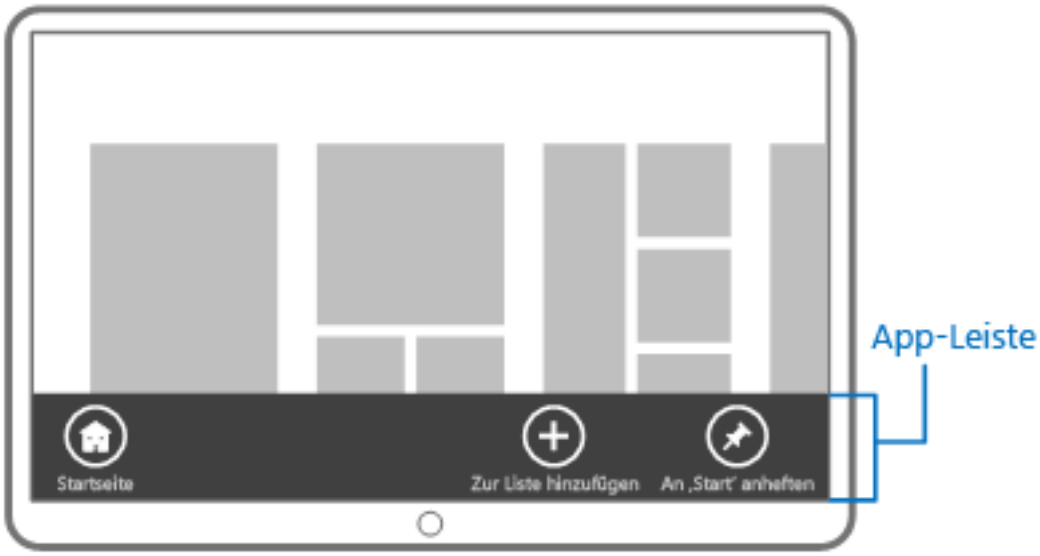

Abbildung 1: App-Leiste (Microsoft, 2013a)

Fährt der Benutzer in die rechte untere Ecke scheinen die Charms auf. Diese sind einheitliche Schaltflächen, die dem Benutzer in jeder App zur Verfügung stehen: "Suche", "Teilen", "Verbinden", "Einstellungen" und "Start". (Microsoft, 2013a)

Weitere Elemente sind dann noch der Hinweis Dialog, Flyouts und das Kontext Menü.

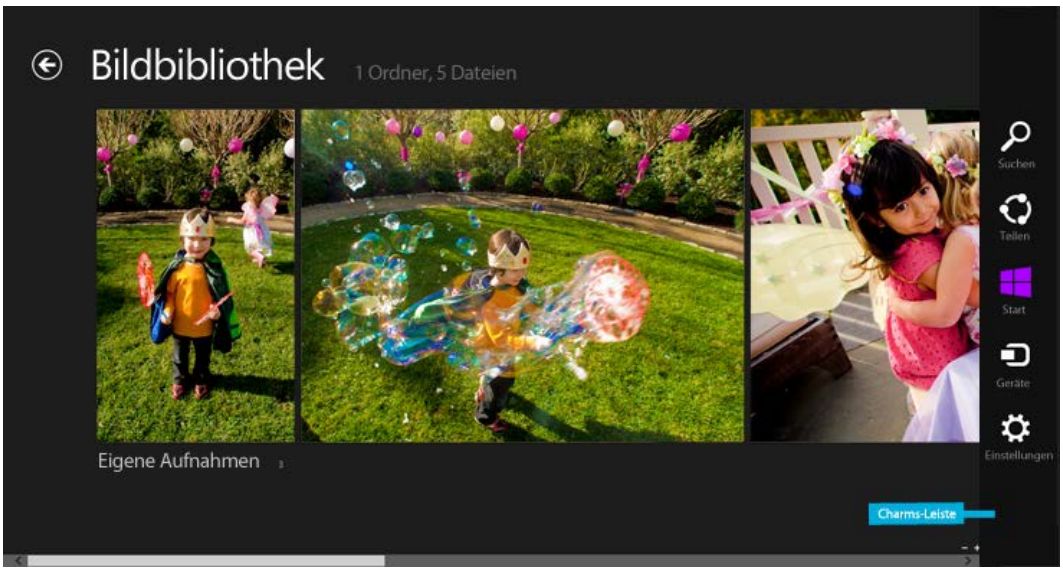

Abbildung 2: Charms (Microsoft, 2013a)

3.1.1.2 Windows Phone 8

Bei einer Windows Phone 8 App ist die Struktur ähnlich einer Windows 8 App - mit dem Unterschied, dass es lediglich keine Charms gibt. Dafür gibt es zusätzlich am oberen Rand die Statusleiste, in welcher bspw. die Uhrzeit, Signalstärke, Batteriestatus, Datenverbindungsrate etc. angezeigt werden. (Microsoft, 2013c)

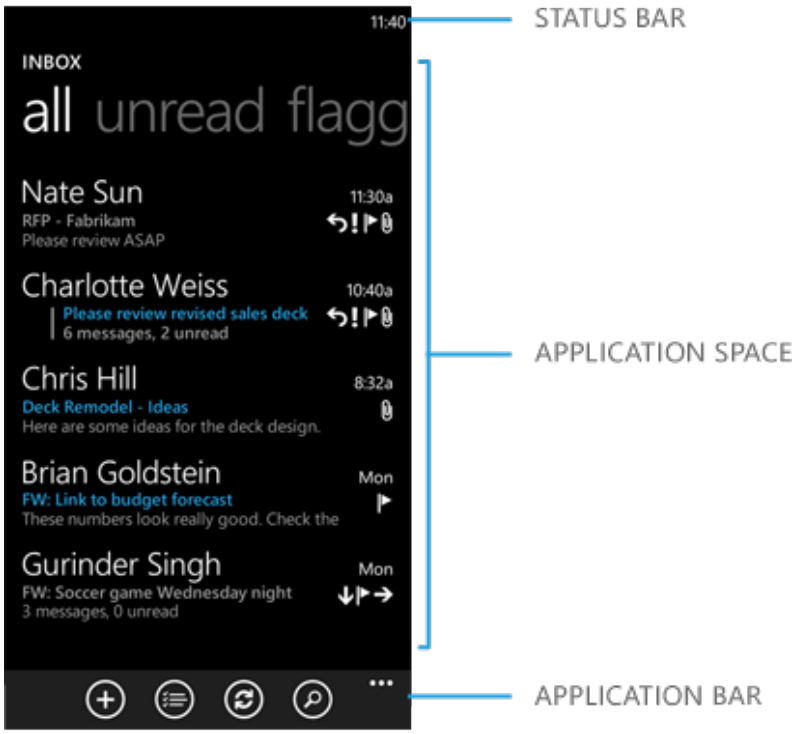

Abbildung 3: Struktur einer Windows Phone 8 Anwendung (Microsoft, 2013c)

Im App-Space wird der eigentliche Inhalt der App angezeigt. Wie ebenso zu erkennen ist, sind in der App-Leiste dieselben Elemente vorhanden wie bei einer Windows 8 App. Zusätzlich kann hier jedoch noch ein Menü dargestellt werden.

3.1.2 Navigationsmodelle

3.1.2.1 Windows 8

Für Windows 8 gibt es zahlreiche Navigationsmodelle, die auch miteinander kombiniert werden können. Grundsätzlich wird zwischen zwei Navigationsmodellen unterschieden:

- Hierarchisches Modell
  - Hub-Seiten (Nabe)
  - Bereichsseiten
  - Detailseiten
- Flaches Modell
  - Navigationsleiste
  - Wechseln

(Scheer, 2012, p. 20)

Das hierarchische Modell ist das am meisten verwendete, da es dem Benutzer bekannt und einfach zu verstehen ist. Eignet sich für Apps mit viel Inhalt und Bereichen. Der Kern des Nabe-Speiche-Designs ist die Unterteilung der Inhalte in verschiedene Bereiche und Detailebenen. (Microsoft, 2013k)

Eine Hub-Seite ist die Einstiegsseite in die App und der Inhalt wird in einer horizontal schwenkbaren Ansicht dargestellt. Die Hub-Seite besteht aus mehreren Kategorien die einzelnen Bereichen der App zugeordnet sind. (Microsoft, 2013k)

Die Bereichsseiten sind die zweite Ebene der App, Inhalte können dem Szenario entsprechend in geeigneter Form dargestellt werden. In einer Bereichsseite werden einzelne Elemente dargestellt, welche wiederum eine eigene Detailseite besitzen. Sie bieten das Panoramalayout und Gruppierungen an. (Microsoft, 2013k)

Detailseiten sind die dritte Ebene der App und hier werden die Details eines Elementes von der Bereichsseite angezeigt. Diese können je nach Inhalt (Text, Bild, Video, etc.) komplett unterschiedlich gestaltet sein. (Microsoft, 2013k)

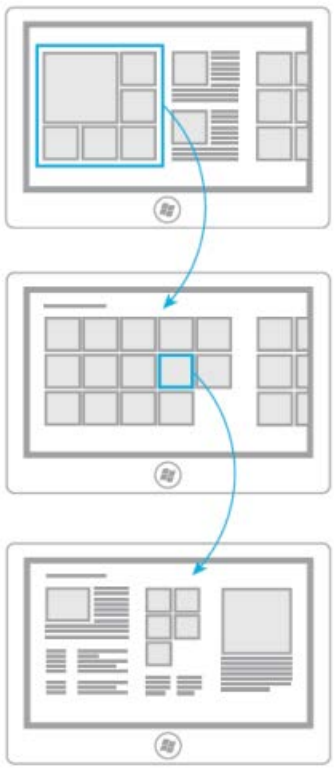

Abbildung 4: Hierarchisches Modell (Microsoft, 2013k)

Das flache Muster ist am ehesten in Spielen, Browsern oder Apps zum Erstellen von Dokumenten vorzufinden. Der Benutzer wechselt zwischen Seiten, Modi oder Registerkarten auf der gleichen Ebene. Dementsprechend darf die App nicht zu umfangreich sein. (Microsoft, 2013k)

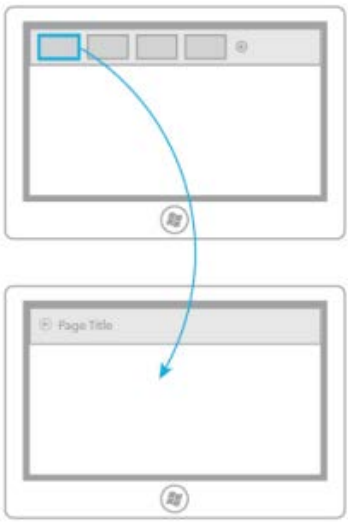

Abbildung 5: Flaches Modell (Microsoft, 2013k)

3.1.2.2 Windows Phone 8

Für Windows Phone 8 gibt es zahlreiche Navigationsmodelle, die auch miteinander kombiniert werden können.

- Panoramalayout
- Tabs (Reiter)
- Liste
- Gleiche Seiten

Das Panoramalayout, eine Technik die es bei Android und iOS nicht gibt. Der virtuelle Bildschirm der App erstreckt sich über mehrere Bildschirmseiten von denen jede einzeln vertikal scrollbar (höher als der Bildschirm) ist. Durch horizontales verschieben kommt man von einer Seite zur nächsten, durch ein einrasten der Seite ist es unmöglich zwischen zwei Seiten hängen zu bleiben. (Immler, 2012, p. 183)

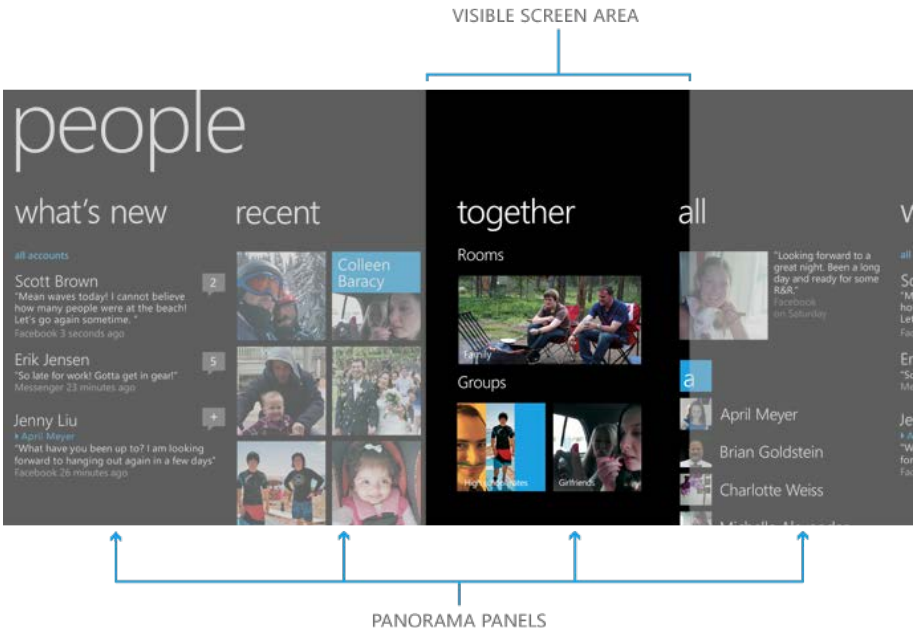

Abbildung 6: Panoramalayout ohne Hauptseite (Microsoft, 2013g)

Um eine App mit mehreren Untermenüpunkten damit zu ermöglichen, gibt es die Möglichkeit auf der Startseite ein Hauptmenü zu verwenden, welches zu weiteren Panoramalayouts verlinkt. (Microsoft, 2013f)

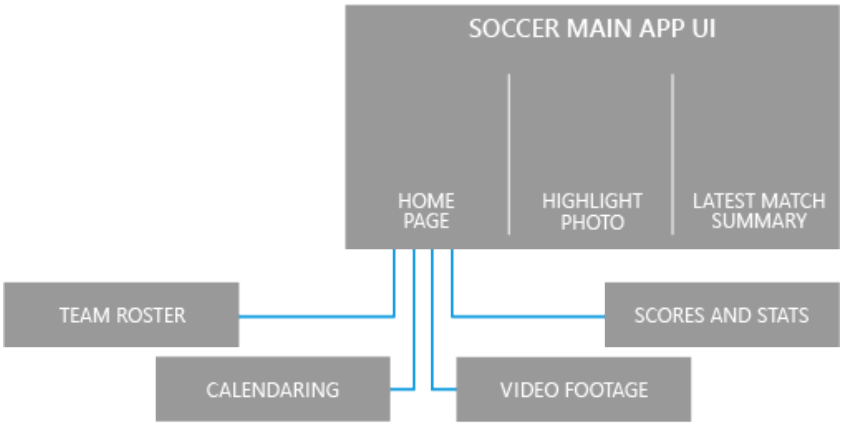

Abbildung 7: Prinzip Panoramalayout mit Hauptseite und Menü (Microsoft, 2013f)

3.1.2.3 Tabs

Das Tab-Prinzip sollte dann verwendet werden, wenn häufig zwischen den Seiten gewechselt werden muss. Dieser Stil ist besonders nützlich, wenn die Anwendung um ein einzelnes Thema wie zum Beispiel Windows 8, Windows Phone 8, etc. basiert. (Microsoft, 2013h)

Die folgende Abbildung zeigt eine Nachrichten-App, welche das Konzept verdeutlicht. Zwischen den drei obigen Tabs kann schnell, einfach umgeschaltet werden.

Abbildung 8: Tabs Prinzip (Microsoft 2013h)

3.1.2.4 Liste

Dieses Konzept ist für Apps geeignet, in welcher eine Liste von Elementen angezeigt werden muss und es eine Detailansicht eines jedes Elementes gibt. Ein Beispiel hierfür wäre eine App mit aktuellen Nachrichten, wo eine Liste mit den Titeln angezeigt wird und wenn man nähere Informationen lesen möchte, kann man tiefer in einen Artikel eintauchen. Auch eine Chatübersicht ist damit einfach darstellbar. (Microsoft, 2013i)

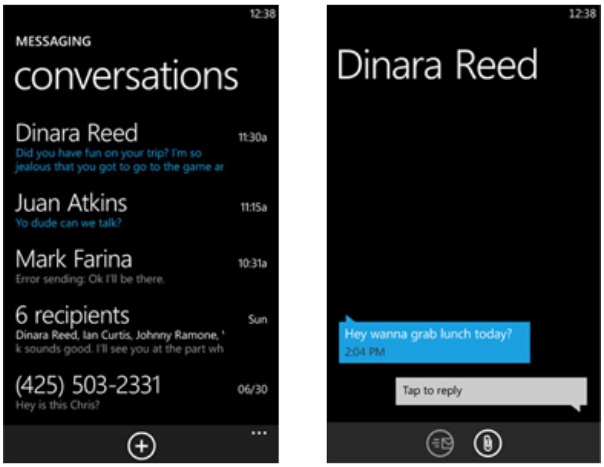

Abbildung 9: List-Navigation (Microsoft, 2013i)

3.1.2.5 Gleiche Seiten

Gleiche Seiten bieten sich dann an, wenn der Seitenaufbau immer gleich ist und sich nur die Inhalte ändern. Um dasselbe Beispiel von oben zu verwenden, wenn ein jeder Artikel gleich aufgebaut ist, könnte man dieses Konzept verwenden und so innerhalb der Artikel auch weiterblättern ohne, dass man zur Übersicht wieder zurückkehren muss. (Microsoft, 2013j)

3.1.3 Lebenszyklus

Da Windows 8 und Windows Phone 8 auf verschiedenen Plattformen basieren, haben deren Apps zwar einen ähnlichen aber doch unterschiedlichen Lebenszyklus.

Den Lebenszyklus einer App sollte man unbedingt kennen, da sonst Daten und Einstellungen des Benutzers verloren gehen können.

Ein Beispiel hierfür wäre folgendes: Eine App ist gestartet welche dem Benutzer Börsenkursen anzeigt. Diese werden alle 30 Sekunden aktualisiert. Wenn die App nun in den Hintergrund kommt und eingefroren wird, werden die Kurse nicht mehr aktualisiert. Wenn der Benutzer dann später zur App zurückkehrt kann es sein, dass er 30 Sekunden lang warten muss, bis die Kurse wieder aktualisiert werden. Und falls die App aufgrund von Speicherressourcen automatisch be-

endet wird und der Benutzer startet diese erneut ist es wie ein Neustart und seine Ticker Daten sind verloren, da die App ja wieder den Startbildschirm zeigt. (Novak, Zoltan, & Fülöp, 2013, p. 404 ff)

Dies widerspricht den Designprinzipien von Microsoft. Deswegen gibt es entsprechende Ereignisse, um auf diese Zustände reagieren zu können und die Daten zu sichern.

3.1.3.1 Windows 8

Der Benutzer kann, wie in früheren Versionen auch, unter Windows 8 mehrere Apps gleichzeitig ausführen, obwohl er wahrscheinlich immer nur mit einer arbeitet.

Jede laufende App benötigt Ressourcen und da unter Windows 8 immer nur eine oder zwei Apps (Landscape, Snapped, Filled) im Vordergrund sind und um Ressourcen zu sparen, haben die Entwickler von Microsoft ein spezielles Ressourcen Management geschaffen. Die aktiven Apps (im Vordergrund) erhalten alle benötigten Ressourcen, um die beste Bedienung der App für den Benutzer zu ermöglichen. (Novak, Zoltan, & Fülöp, 2013, p. 404 ff)

Alle anderen Apps, welche sich im Hintergrund befinden, werden eingefroren und bis auf den belegten Speicher verbrauchen diese dann keine CPU- oder andere Systemressourcen mehr und die Batterie wird ebenfalls geschont. Dadurch wird gewährleistet, dass die aktive App des Benutzers alle Ressourcen bekommt die sie auch benötigt. Wird mehr Speicher benötigt beendet Windows 8 einfach eine eingefrorene App, welche im Hintergrund liegt. Ausgenommen sind Hintergrundaufgaben, da diese vom Betriebssystem verwaltet werden. (Novak, Zoltan, & Fülöp, 2013, p. 404 ff)

Wird eine App in den Hintergrund verschoben, da eine andere in den Vordergrund rückt, dauert es ca. 5 Sekunden bis die App eingefroren wird um bei einem unbeabsichtigten Wechsel der App dem Benutzer keine Wartezeit zuzumuten. Dieses Modell gibt dem Benutzer das Gefühl das seine App immer aktiv und flüssig ist. (Novak, Zoltan, & Fülöp, 2013, p. 404 ff)

Eine Windows 8 App kennt die folgenden Zustände und Ereignisse (in Klammer):

- Aktiviert (`OnLaunched`)
- Wird angehalten (`Suspending`)
- Wird fortgesetzt (`Resuming`)
- Wird beendet

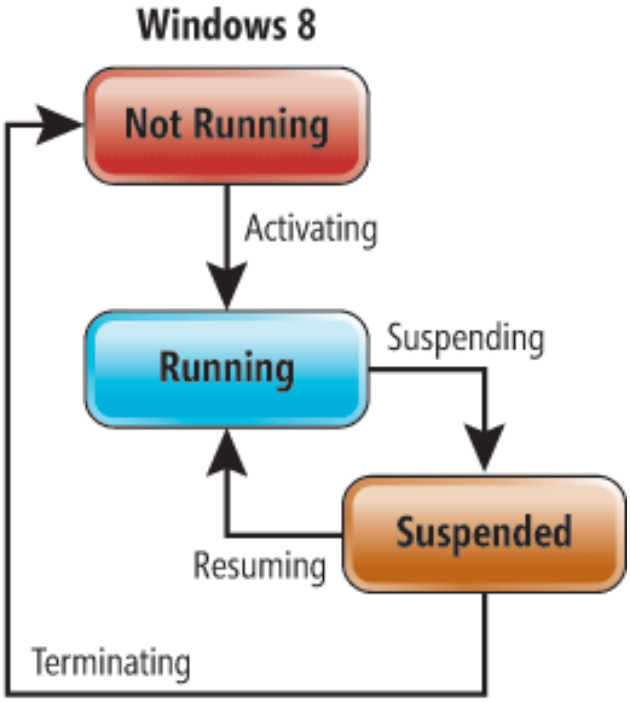

Abbildung 10: Windows 8 Lebenszyklus einer Anwendung (Reyes, 2013)

Aktiviert (`OnLaunched`): Die App wurde vom Benutzer selbst oder von einer anderen App aus gestartet. Zuerst wird mittels `PrevoiusExecutionState` überprüft, ob die App neu gestartet wurde, oder ob der Benutzer nur wieder zurückkehrt. Zusätzlich empfängt der Ereignishändler ein Argument vom Typ `LaunchActivatedEventArgs`. Anhand der Kind-Eigenschaft des Argumentes kann festgestellt werden, ob die App vom System (`Terminated`) oder vom User (`ClosedByUser`) beendet wurde. (Novak, Zoltan, & Fülöp, 2013, p. 404 ff)

Angehalten (`Suspending`): Wenn der Benutzer zu einer anderen App wechselt, hält das System die vorher aktive App an. Hier empfängt der Ereignishandler ein `SuspendingEventArgs` Argument. Hier kann der aktuelle Zustand der App gespeichert werden, um ihn später wiederherstellen zu können, falls das System die App beendet. Auch nicht benötigte Ressourcen können an das System zurückgegeben werden. Es sollte keine Operation starten, die die Ausführung des Schließvorganges stören könnte, wie bspw. Benutzerangaben in einem Popup Feld zu erwarten. (Novak, Zoltan, & Fülöp, 2013, p. 404 ff)

Fortgesetzt (`Resuming`): Wechselt der Benutzer zu einer eingefrorenen App zurück und gespeicherte Ressourcen können wieder freigeben werden. (Novak, Zoltan, & Fülöp, 2013, p. 404 ff)

Beendet: Wenn das System die App beendet gibt es keinen Ereignishandler, um darauf zu reagieren, deswegen müssen schon beim Anhalten der App die Daten gespeichert werden. (Novak, Zoltan, & Fülöp, 2013, p. 404 ff)

Beim Anlegen einer neuen App ist der Lebenszyklus dieser schon mit dem entsprechenden Code hinterlegt.

Listing 1: Windows 8 Lebenszyklus Code

```
public App()
{
   this.InitializeComponent();
   this.Suspending += OnSuspending;
}

// Wird aufgerufen, wenn die Anwendung durch den Endbenutzer
// normal gestartet wird.
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
   Frame rootFrame = Window.Current.Content as Frame;

   // App-Initialisierung nicht wiederholen, wenn das Fenster
   // bereits Inhalte enthält.
   // Nur sicherstellen, dass das Fenster aktiv ist.
   if (rootFrame == null)
   {
      // Einen Rahmen erstellen, der als Navigationskontext fungiert
      // und zum Parameter der ersten Seite navigieren
      rootFrame = new Frame();

      if (e.PreviousExecutionState ==
      ApplicationExecutionState.Terminated)
      {
      // TODO: Zustand von zuvor angehaltener Anwendung laden
      }

      // Den Rahmen im aktuellen Fenster platzieren
      Window.Current.Content = rootFrame;
   }

   // Wird aufgerufen, wenn die Ausführung der Anwendung angehalten
   // wird. Der Anwendungszustand wird gespeichert, ohne zu wissen,
   // ob die Anwendung beendet oder fortgesetzt wird und die
   // Speicherinhalte dabei unbeschädigt bleiben.
   private void OnSuspending(object sender, SuspendingEventArgs e)
   {
      var deferral = e.SuspendingOperation.GetDeferral();
      // TODO: Anwendungszustand speichern und alle
      // Hintergrundaktivitäten beenden
      deferral.Complete();
   }
```

3.1.3.2 Windows Phone 8

Der Lebenszyklus einer App unter Windows Phone 8 ist ähnlich dem von Windows 8. Mit dem Unterschied, dass eine App bevor sie deaktiviert wird noch

in einen `Tomstoned` (Prozesse beendet) Status versetzt wird, aus welchem sie aber wieder aktiviert werden kann. Eine App muss nicht in diesen Zustand vom System gesetzt werden, aber wenn sie in diesen Zustand gesetzt wird, ist es der letzte Zustand bevor sie komplett beendet wird. (Reyes, 2013)

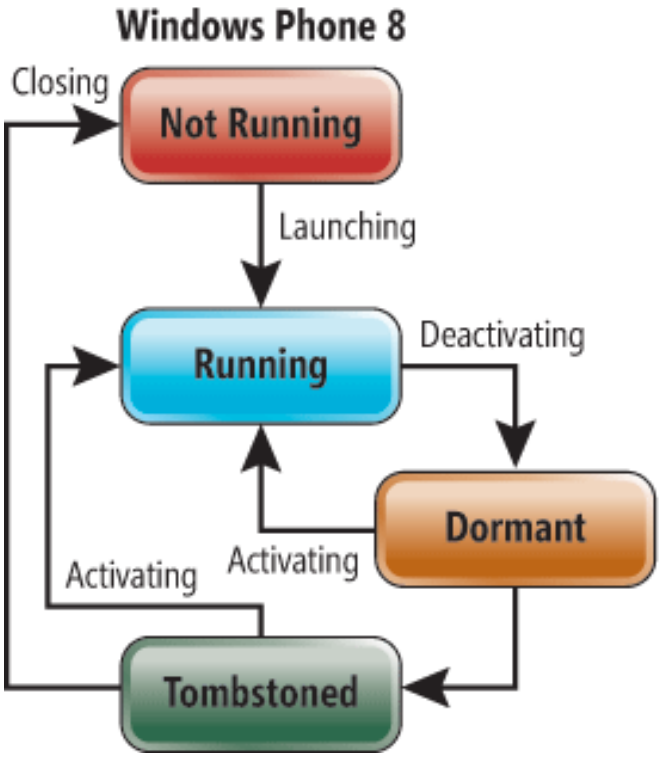

Abbildung 11: Windows Phone 8 Lebenszyklus einer Anwendung (Reyes, 2013)

Listing 2: Windows Phone 8 Lebenszyklus Code

```
// Code, der beim Starten der Anwendung ausgeführt werden soll
// (z. B. über "Start")
// Dieser Code wird beim Reaktivieren der Anwendung nicht ausgeführt
private void Application_Launching(object sender,
LaunchingEventArgs e){}

// Code, der ausgeführt werden soll, wenn die Anwendung aktiviert wird
// (in den Vordergrund gebracht wird)
// Dieser Code wird beim ersten Starten der Anwendung nicht ausgeführt
private void Application_Activated(object sender,
ActivatedEventArgs e) {}

// Code, der ausgeführt werden soll, wenn die Anwendung deaktiviert
// wird (in den Hintergrund gebracht wird)
// Dieser Code wird beim Schließen der Anwendung nicht ausgeführt
private void Application_Deactivated(object sender,
DeactivatedEventArgs e) {}

// Code, der beim Schließen der Anwendung ausgeführt wird (z. B.
```

```
// wenn der Benutzer auf "Zurück" klickt)
// Dieser Code wird beim Deaktivieren der Anwendung nicht ausgeführt
private void Application_Closing(object sender, ClosingEventArgs e) {}
```

3.2 Design/Usability

Die neuen Apps von Windows 8 und Windows Phone 8 haben vieles gemeinsam. Sie werden alle im Vollbildschirm (ohne etwaiger Leisten) ausgeführt. Das Layout einer App weist zum Beispiel keinen „Schließen“ Button auf. Dieses Kapitel geht nun auf das Design der Benutzeroberfläche und deren Bedienung ein.

Bezüglich der Benutzeroberfläche sind sorgfältige Entscheidungen notwendig, um entsprechend den Möglichkeiten des jeweiligen Geräts ein optimales Benutzererlebnis sicherzustellen.

3.2.1 Design Prinzipien

Microsoft hat diesbezüglich Design Prinzipien festgelegt, diese sind ein Set von User Experience Design Prinzipien, welche bei einer jeden Anwendung von Microsoft, egal für welches Produkt (Windows 8, Windows Phone 8, Xbox), eingehalten werden sollen.

Diese fünf Prinzipien stellen die Grundlage für das Erstellen großartiger Windows Store-Apps dar. Alle Entwurfs- und Entwicklungsentscheidungen sollen an diesen Prinzipien ausgerichtet werden.

3.2.1.1 Seien Sie stolz auf Ihre Arbeit

Die App soll in jeder Phase vollständig durchdacht und perfekt sein. Dazu ist es empfehlenswert, viel Wert auf kleine Dinge zu legen, die von vielen Benutzern häufig gesehen werden. Folgende Punkte sind zu beachten:

- Sorgfältig an den Details arbeiten.
- Die Nutzung der App muss sicher und zuverlässig sein.
- Balance, Symmetrie und Hierarchie beachten.
- Anpassen des Layouts an das Raster.
- Die App ebenso zugänglich machen für Personen mit Beeinträchtigungen oder Behinderungen.

(Microsoft, 2013e) (Walther, 2012, p. 6)

3.2.1.2 Schnell und dynamisch

Die App soll die direkte Interaktion des Inhaltes durch den Benutzer ermöglichen. Auf Aktionen muss schnell reagiert werden und die Benutzeroberfläche muss ein

Gefühl von Kontinuität vermitteln. Animationen sollen sinnvoll eingesetzt werden. Folgende Punkte sind zu beachten:

- Schnelle Reaktion auf Benutzerinteraktionen.
- Design muss für die Fingereingabe und die direkte Interaktion konzipiert sein.
- Den Benutzer durch sinnvolle Animationen begeistern.
- Für einen reibungslosen Übergang zwischen Abläufen muss gesorgt sein.

(Microsoft, 2013e) (Walther, 2012, p. 6)

3.2.1.3 Wirklich digital

Alle Vorteile des digitalen Mediums sollen genutzt werden und physische Grenzen beseitigt. Mit dem Design aus Farben und Bildern sollen die Grenzen der wirklichen Welt überwunden werden. Folgende Punkte sind zu beachten:

- Dynamisch und lebendig durch Kommunikation.
- Auf eine schöne Typografie achten.
- Kräftige, lebendige Farben verwenden.
- Verbindung mit der Cloud herstellen.

(Microsoft, 2013e) (Walther, 2012, p. 6)

3.2.1.4 Weniger ist oft mehr

Das Design soll auf das Wesentliche reduziert sein und so eine klare und sinnvolle Umgebung für die relevanten Elemente bieten, damit der Benutzer in den Inhalt der App eintauchen kann. Folgende Punkte sind zu beachten:

- Inhalt ist wichtiger als Spielereien.
- Visuell und direkt fokussiert.
- Vertrauen des Benutzers wecken.
- Vermeiden von Redundanz des UI

(Microsoft, 2013e) (Walther, 2012, p. 6)

3.2.1.5 Gemeinsam gewinnen

Andere Geräte, Systeme und Apps einbinden um bessere Szenarien für den Benutzer zu ermöglichen. Zum Beispiel die Inhalte einer App für eine andere freigeben. Bereits bekanntes für den Benutzer nutzen wie z.B. die standardmäßigen Fingerbewegungen und Charms.

Es überzeugt von Vertrautheit, Kontrolle und Sicherheit. Folgende Punkte sind zu beachten:

- Nutzen des UI-Modells.

- Contracts verwenden.
- Verwenden von Tools und Vorlagen von Microsoft um Konsistenz zu fördern.

(Microsoft, 2013e) (Walther, 2012, p. 6)

3.2.2 Form

3.2.2.1 Windows 8

Windows 8 Modern UI ist konzipiert im Landscape Modus dargestellt zu werden, deswegen wird hier auch horizontal statt vertikal gescrollt.

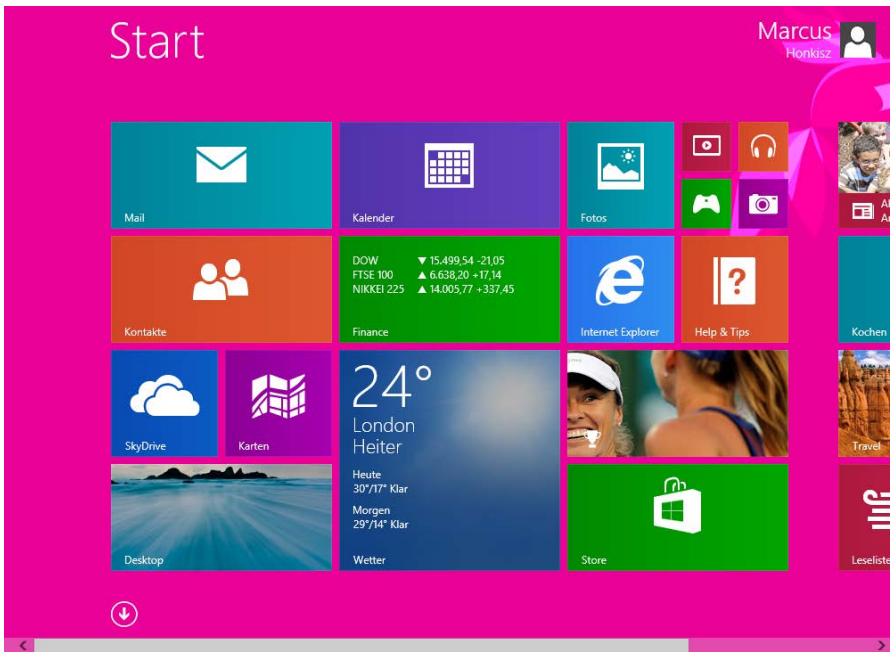

Abbildung 12: Windows 8 Modern UI

Deswegen startet eine Windows 8 App zuerst im Landscape Modus. Andere Ansichten wie Portrait, Snapped und Filled sollten als weitere Möglichkeiten darstellbar sein. (Riga, 2013, Min. 15:35)

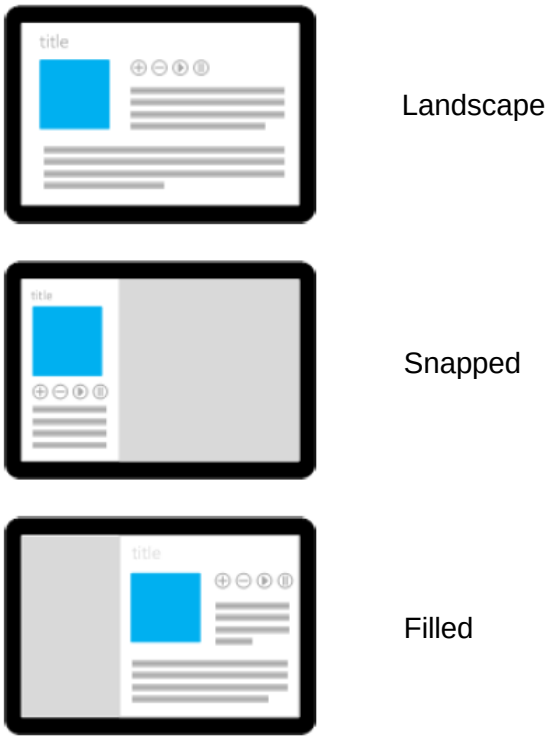

Abbildung 13: Landscape, Snapped, Filled (Microsoft, 2012a, p. 9)

Die Ansicht Portrait und Landscape werden im Manifest festgelegt und die Snapped und Filled Ansicht müssen durch den Entwickler implementiert werden. Jedoch muss die Bildschirmauflösung von mindestens 1633 x 768px gegeben sein. Falls diese Ansicht keinen Sinn macht - zum Beispiel bei Spielen - ist es sinnvoll, den Benutzer mittels eines aussagekräftigen Textes im angedockten Bereich informieren. (Scheer, 2012, p. 21)

Auch ist eine App auf eine Auflösung von mindestens 1024x768px ausgelegt. (Riga, 2013, Min. 15:35)

3.2.2.2 Windows Phone 8

Auf einem Windows Phone 8 ist die Oberfläche so gestaltet, dass nicht horizontal sondern vertikal gescrollt wird. Eine App soll deswegen zuerst im Portrait Modus starten. Eine weitere Ansicht, welche ebenso darstellbar sein soll ist wie bei einer Windows 8 App der Landscape Modus. Andere Ansichten wie Snapped und Filled gibt es hier nicht. (Riga, 2013, Min. 15:35)

Abbildung 14: Windows Phone 8 Modern UI (Microsoft, 2013b)

Bei der Konzeptionierung einer App sind die üblichen Auflösungen 800x480px, 1280x720px und 1280x768px zu beachten. (Riga, 2013, Min. 15:35)

3.2.3 User Experience

3.2.3.1 Windows 8

Windows 8 ist auf eine ganz andere Bedienung ausgelegt als Windows Phone 8, folgende Punkte sind unbedingt zu beachten:

- Bedienung mittels ein oder zwei Händen bzw. Maus
- Ob eine spezifische Hardware (Kamera) vorhanden ist wird erst während der Laufzeit von einer App gecheckt.
- Für mehr Inhalt wird horizontal gescrollt.
- Genug Platz in der AppBar.
- On-Screen Zurück Button
- Semantischer Zoom

(Riga, 2013, Min. 16:29)

3.2.3.2 Windows Phone 8

Windows Phone 8 ist auf eine ganz andere Bedienung ausgelegt als Windows 8, folgende Punkte sind unbedingt zu beachten:

- Bedienung mittels einer Hand.
- Spezifische Hardware (Kamera) immer vorhanden.
- Für mehr Inhalt wird vertikal gescrollt.
- Limitierter Platz in der AppBar.

- Hardware Zurück Button
- Kein semantischer Zoom

(Riga, 2013, Min. 16:29)

3.3 Programmierung

3.3.1 (API-)Architektur

3.3.1.1 Windows 8

Mit der neuen Windows 8 Modern Oberfläche führte Microsoft die komplett neue WinRT API ein, die bisherige steht parallel zur neuen API für den herkömmlichen Desktop zur Verfügung, sodass je nach Version die altbewährten Win 32/.NET-Anwendungen und/oder die neuen Apps ausgeführt werden können.

Der Browser als Laufzeitumgebung für HTML/JS oder XAML/Silverlight ist möglich. Das bedeutet es gibt unterschiedliche Schnittstellen (API = application programming interface) bei Windows 8 mit denen Apps entwickelt werden können. (Bleske, 2013, p. 33)

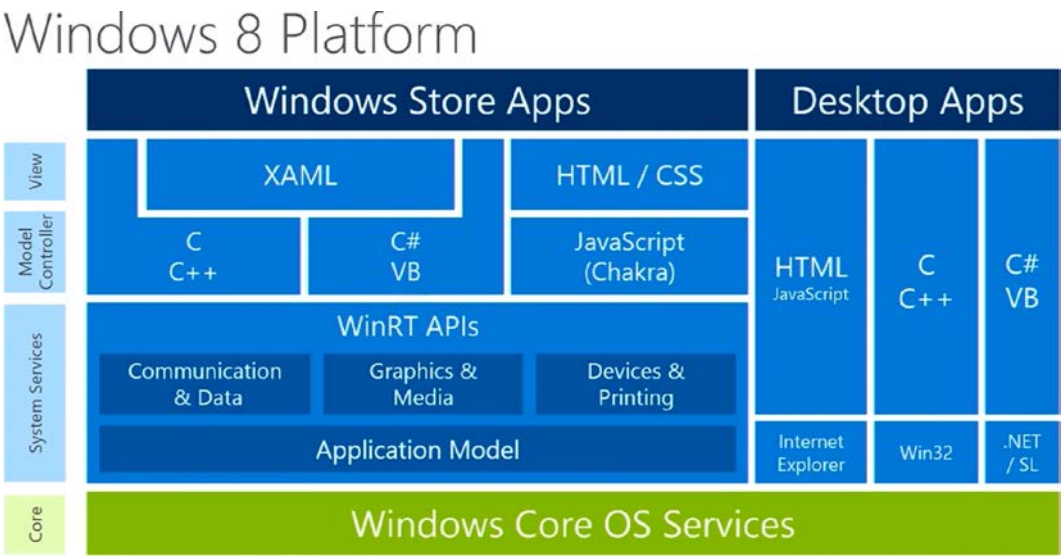

Abbildung 15: Windows 8 Plattform (Riga, 2013, Min. 07:00)

Auf der rechten Seite werden die Programmiersprachen und Laufzeitumgebungen für den herkömmlichen Desktop dargestellt. Hier gibt es für jede Sprache eine andere Laufzeitumgebung.

Und genau hier liegt der Unterschied in Bezug auf das neue Windows 8 Modern UI, ersichtlich auf der linken Seite. Hier verwenden alle Sprachen dieselbe Laufzeitumgebung WinRT (= Windows Runtime).

Die WinRT beinhaltet eine Klassenbibliothek, welche die Grundlage ist für alle Windows Store Apps, egal mit welcher Programmiersprache die App erstellt wird.

„The Windows Runtime (WinRT) contains a class library which you can use in your Windows Store apps. These classes are projected directly into JavaScript so they appear to built-in JavaScript objects.“ (Walther, 2012, p. 22)

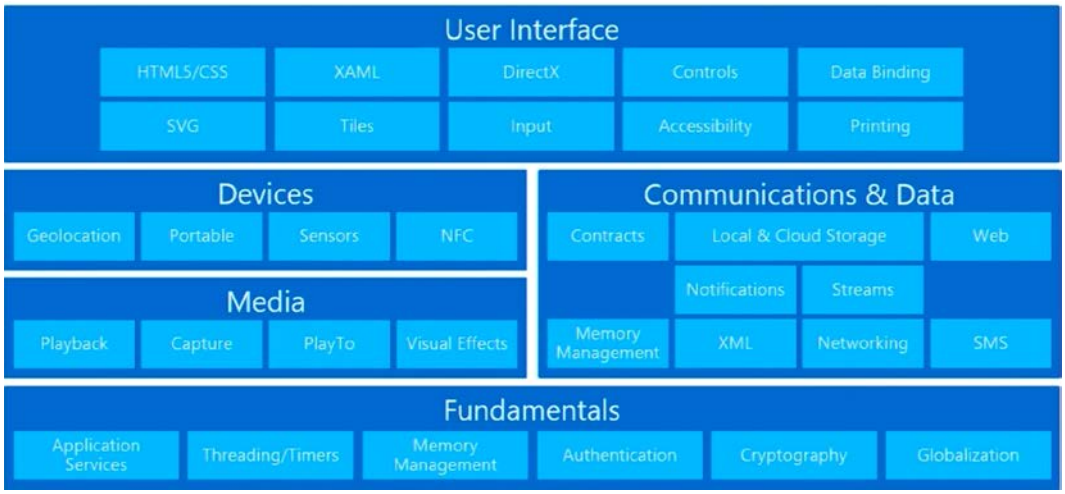

Abbildung 16: WinRT APIs (Riga, 2013, Min. 07:45)

3.3.1.2 Windows Phone 8

Für das Windows Phone 8 gibt es die Windows Phone API, welche sich aus .NET, Windows Phone Runtime und Win32/COM zusammensetzt.

Umgesetzt können Apps mit allen Programmiersprachen wie für Windows 8, ausgenommen ist HTML5/JS, wie auf der folgenden Grafik zu erkennen ist.

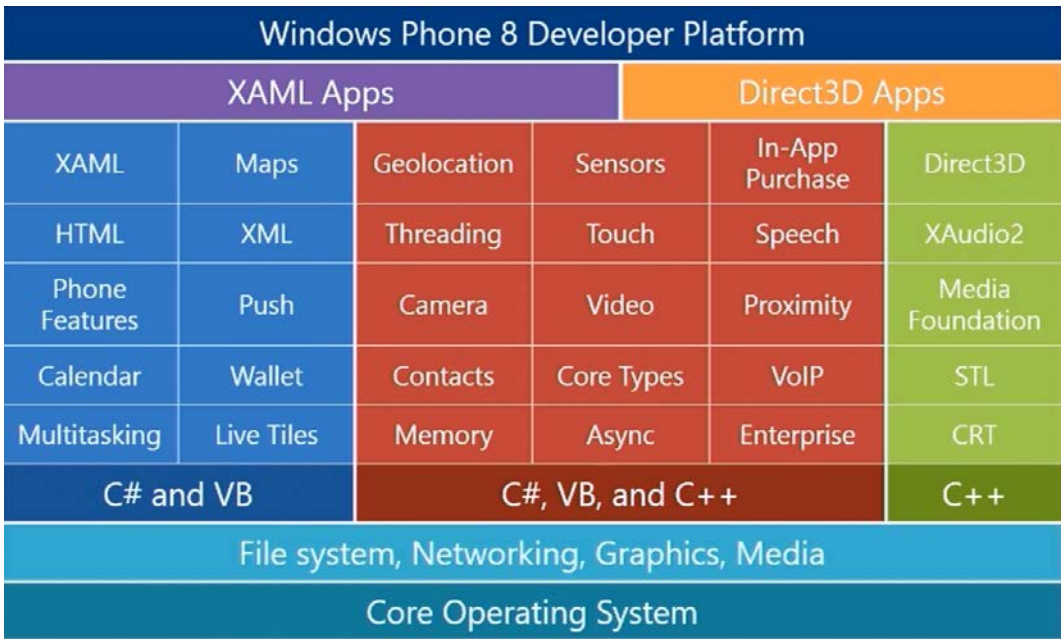

Abbildung 17: Windows Phone 8 Plattform (Riga, 2013, Min. 08:52)

Möchte man eine App für beide Systeme empfiehlt es sich also nicht mit HTML5/JS zu starten.

Es ist zwar möglich eine HTML5/JS App in eine XAML App zu integrieren, wobei das WebBrowser-Steuerelement HTML5-basierte Webseiten hostet. Auch können keine APIs verwendet werden. Eignet sich also bspw. für ein Spiel. (Shapiro & Burtoft, 2013)

Bei einer Windows Phone 8 App handelt es sich also entweder um eine XAML oder Direct3D (= Programmierschnittstelle von Microsoft für 3D Computergrafik, Bestandteil von DirectX) App. (Microsoft, 2013l)

3.3.1.3 Gemeinsame APIs

Das folgende Diagramm zeigt die Beziehung zwischen der Windows Phone Runtime und der Windows Runtime.

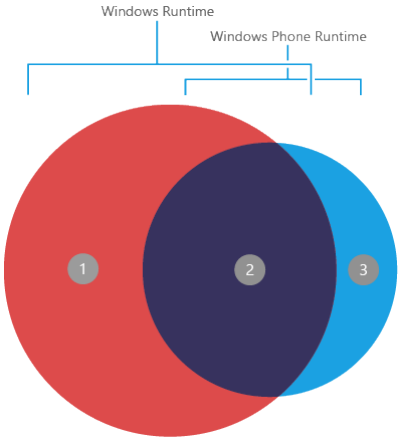

Abbildung 18: Windows Phone Runtime und WinRT (Microsoft, 2013l)

Die Windows Phone Runtime hat von der WinRT einige APIs adoptiert (dunkelblauer Bereich). Der rote Bereich beinhaltet APIs die nur für die WinRT zur Verfügung stehen, während der hellblaue Bereich diejenigen APIs enthält, die nur für das Windows Phone verwendet werden können. (Microsoft, 2013l)

Folgende APIs wurden von der WinRT für Windows Phone Runtime adoptiert:

- Networking
- Sensors
- Proximity
- Storage
- Location
- Threading
- File System

3.3.2 Asynchronität

Eine Neuheit des neuen Windows 8 Modern UI ist, das Apps nicht mehr synchron laufen müssen sondern auch asynchron arbeiten können. Dies bedeutet, dass das GUI nicht mehr von Prozessen aus dem Hintergrund behindert wird, falls diese länger brauchen. Es bleibt weiterhin erreichbar und bedienbar und wirkt somit für den Benutzer immer flüssig und schnell.

Die Regel hierfür lautet, dass wenn eine API länger als 50 Millisekunden läuft, ist sie asynchron. Die meisten APIs bieten standardmäßig nur asynchrone Methoden an. (Novak, Zoltan, & Fülöp, 2013, p. 207)

In C# 5.0 hat Microsoft die asynchrone Programmierung mittels zwei Schlüsselwörter vereinfacht. Diese sind der Modifizierer `async` und der Operator `await`. Mit denen können synchrone Methoden in asynchrone umgewandelt und in den Compiler verlagert werden. (Novak, Zoltan, & Fülöp, 2013, p. 191)

Wenn eine Methode mit `Async` endet, benötigt sie das Schlüsselwort `await` vor ihrem Aufruf und das Schlüsselwort `async` muss zur Signatur der Funktion hinzugefügt werden.

Asynchrone Methoden dienen zur Verwaltung und nicht zur Erstellung der Asynchronität. Sie starten keine neuen Threads sondern ersparen nur das Erstellen von Callbacks, welche im Hintergrund verwendet werden. (Novak, Zoltan, & Fülöp, 2013, p. 192)

Folgend nun ein kurzes Beispiel wie eine synchrone Methode in eine asynchrone geändert werden kann, für den Aufruf `WebClient.DownloadString`, dessen Ausführung länger dauern kann.

Listing 3: Synchrone – Asynchrone Methode

```
// Synchrone Methode
private void DownloadSync(string url)
{
   WebClient wc = new WebClient();
   string result = wc.DownloadString(url);
   this.txtStatus.Text = result;
}

// Asynchrone Methode
private async void DownloadAsync(string url)
{
   WebClient wc = new WebClient();
   string result = await wc.DownloadStringTaskAsync(url);
   this.txtStatus.Text = result;
}
```

Da ja auch für Windows 8 Apps in HTML5/JS erstellt werden können, wurde hier die Asynchronität über Promises geschaffen.

JavaScript ist bekanntlich eine single-threaded Sprache, das heißt jede langlaufende oder wartende Operation blockiert diesen Thread und somit die ganze Anwendung. Die Windows Runtime implementiert die Common JS Promises/A-Proposal, um dies Problem zu lösen. Eine Promise ist ein JS-Objekt, das ähnlich wie ein Task in C# zu einem späteren Zeitpunkt einen Wert zurückgeben kann. Somit wurden alle asynchronen APIs in Promise-Objekte gehüllt und bleiben so einfach zu verwenden in JS. (Novak, Zoltan, & Fülöp, 2013, p. 211f)

3.3.3 Namespaces

In beiden Plattformen kann XAML für die Benutzeroberfläche verwendet werden, dies ist jedoch aufgrund verschiedener Namespaces nicht plattformübergreifend.

Große Unterschiede gibt es bezüglich Seitenlayouts, Seitenausrichtung und bei der Verwendung der XAML-Steuerelemente. (Reyes, 2013)

Viele Steuerelemente sind nämlich in beiden Plattformen gleich, jedoch liegen sie in verschiedenen Namespaces.

Auch die Ereignisse des Lebenszyklus einer App sind auf beiden Plattformen in unterschiedlichen Namespaces.

In Windows 8 sind die Steuerelemente im Namespace `System.Windows.Controls` enthalten, unter Windows Phone 8 in `Windows.UIXaml.Controls`, `Microsoft.Phone.Controls` und `Microsoft.Phone.Shell`. (Riga, 2013, Min. 20:15)

Glücklicherweise reagiert der XAML-Editor mit einer Warnung, wenn man versucht, ein nicht unterstütztes Steuerelement hinzuzufügen. Allerdings gibt es, da XAML die bedingte Kompilierung nicht unterstützt, für keine Plattform eine einfache Möglichkeit zum Einfügen von Namespaces zur Laufzeit. (Reyes, 2013)

3.3.4 Steuerelemente

Bei den Controls (Steuerelementen) gilt dasselbe zu beachten wie schon bei den Namespaces. Jedes Steuerelement ist für die eigene Plattform optimiert und es ist höchst empfehlenswert die Benutzersteuerelemente für jede Plattform getrennt zu entwerfen. (Reyes, 2013)

In der folgenden Liste sind einige Beispiele für Standardsteuerelemente aufgeführt:

Tabelle 1: Standardsteuerelemente (Riga, 2013, Min. 21:38)

Windows Phone 8	***Windows 8***
`PhoneApplicationPage` *ist he root page element*	`Page` *ist he root element*
Use the ***`LongListSelector`*** *to show vertically scrolling content*	*Use the* `ListView` *to show vertically scrolling content*
Use the ***`Pivot`*** *control for paging content horizontally*	*Use the* `FlipView` *control for paging content horizontally*
Use the ***`ApplicationBar`*** *control*	*Use the* `AppBar` *Control*
	Use `GridView` *to group content in a grid*

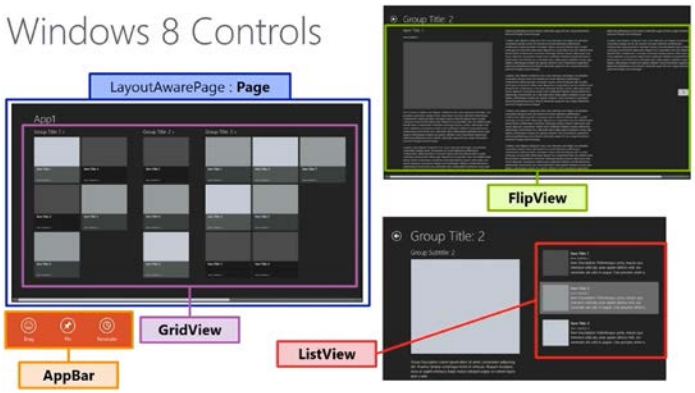

Abbildung 19: Windows 8 Steuerelemente (Riga, 2013, Min. 22:08)

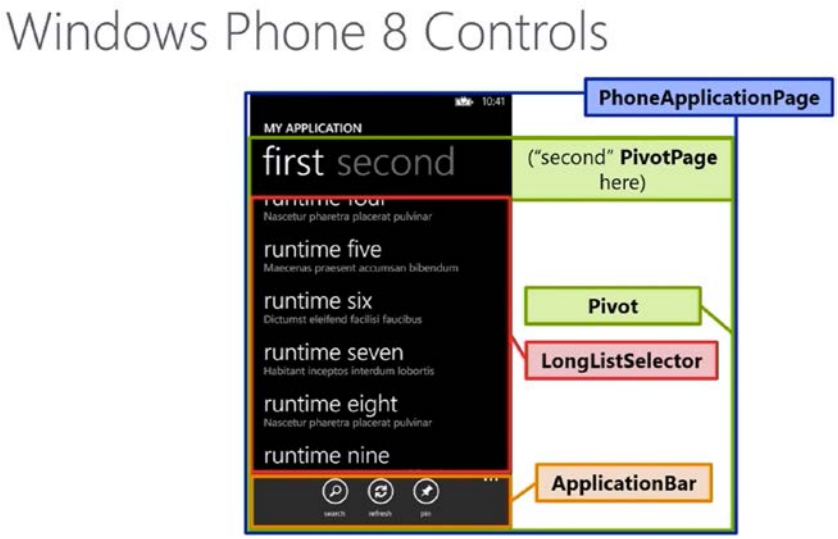

Abbildung 20: Windows Phone 8 Steuerelemente (Riga, 2013, Min. 22:48)

Allgemein funktionieren Steuerelemente auf Touch-Geräten und mit Maus und Tastatur gleichermaßen. Dank der Steuerelemente und Vorlagen von Windows 8 und Windows Phone 8 verfügen Ihre Apps stets über eine intuitiv bedienbare, einheitliche Oberfläche. (vgl. Microsoft, 2012, p. 11)

3.3.5 Speichmanagement

Beide Plattformen unterstützen das lokale Speichern von Schlüsselwerten (Einstellungen) und von Dateien und Verzeichnissen. Jeder Speicher für eine Anwendung ist isoliert, sodass keine andere App auf diesen Speicher zugreifen kann. (Riga, 2013, Min. 24:38)

Während Windows Phone 8 standardmäßig das Speichern in eine Datenbank (SQL CE) ermöglicht, stehen unter Windows 8 nur SQLite Bibliotheken zur Verfügung, welche man einbinden kann. (Riga, 2013, Min. 24:38)

Des Weiteren gibt es geteilte APIs im `Windows.storage` Namespace, welche den Vorteil haben, dass man sogar den lokalen Speicher so managen kann, das er übertragbar ist zwischen einer Windows 8 und einer Windows Phone 8 App. (Riga, 2013, Min. 25:30)

Beim Speichern werden viele der asynchronen und wartenden Möglichkeiten verwendet, die mit Windows 8 eingeführt wurden. Die APIs vom Windows Phone 8 sind Teile der vollen APIs von Windows 8. Zum Beispiel gibt es am Windows Phone keinen Roaming Speicher (Cloud), keinen Temporären Speicher und das Speichern von Einstellungen in ein lokales Verzeichnis wie unter Windows 8 ist auch nicht möglich. (Riga, 2013, Min. 25:50)

Da an einem Windows 8 Gerät unterschiedliche Benutzer ein Konto haben können, legt das System selbstständig für jeden Benutzer einen eigenen Speicher Ordner an, sodass der Entwickler nur mehr die richtigen APIs aufrufen braucht. (Bleske, 2013, p. 244)

Die verschiedenen Speichermöglichkeiten noch einmal zusammengefasst:

Tabelle 2: Speicheroptionen in Windows 8 und Windows Phone 8

Feature Namespaces	***Zweck***	***Windows Phone 8***	***Windows 8***
`Windows.Storage`	*Lokaler Speicher*	*Ja*	*Ja*
`System.IO.IsolatedStorage.IsolatedStorageFile`	*Lokaler Speicher ALT (Windows Phone 7)*	*Ja*	*Nein*
`ApplicationData-Einstellungen (LocalFolder)`	*Schlüssel Wert-Speicher*	*Nein*	*Ja*
`System.IO.IsolatedStorage.IsolatedStorageSettings`	*Schlüssel Wert-Speicher*	*Ja*	*Nein*
SQL CE	*Datenbank*	*Ja*	*Nein*

3.3.6 Contracts/Extensions

Contracts und Extensions gibt es nur bei der Entwicklung eines Windows 8 App für den Windows Store. Unter Windows Phone 8 gibt es Launchers und Choosers. Sie bieten dem Benutzer die gleichen Möglichkeiten aber ihre Implementierung ist unterschiedlich. Der Unterschied wird im folgenden Kapitel erklärt und gezeigt.

3.3.6.1 Windows 8

Ein Contract ist eine Schnittstelle zu Windows 8 und eine Extension erweitert eine bereits vorhandene Funktion. (Bleske, 2013, p. 181)

Mit diversen Schnittstellen und Erweiterungen ist es möglich die Funktionalität einer App zu erweitern, um bspw. auf integrierte Windows Funktionen zuzugreifen. (Bleske, 2013, p. 181)

Mit dem Search Contract ist es möglich einen Suchbereich in die App zu integrieren, welcher es ermöglicht die Inhalte aller Apps zu durchsuchen und die Suchabfrage in andere App zu übertragen. Umgekehrt wird auch der eigene Inhalt den anderen Apps mit einem Search Contract zur Verfügung gestellt. (vgl. Microsoft, 2012, p. 5)

Ein anderer Contract ist der Share Contract, mit diesem ist es möglich Inhalt zwischen zwei Apps oder mit einem Dienst zu teilen. Apps, die dies unterstützen, können Inhalt zu und von jeder anderen App automatisch teilen, wenn diese auch den Share Contract unterstützen. (Microsoft, 2013d)

Für das Dateihandling gibt es auch zwei Contracts, erstens den File Open Picker Contract, welcher den Benutzern ermöglicht, Dateien von ihrer App zu picken, während sie eine andere gerade verwenden. Der andere ist der File Save Picker Contract, welcher den Benutzern ermöglicht, Dateien von ihrer App zu speichern während eine andere gerade verwendet wird.

Weitere Contracts sind: Cached File Updater Contract, Play To Contract und der Settings Contract. (Microsoft, 2013d)

Mit der Extension Background Tasks kann die App Hintergrundaufgaben durchführen während die App im Hintergrund liegt und eingefroren ist oder sogar geschlossen wurde und keine Eingaben vom Benutzer benötigt. (Microsoft, 2013d)

Mit der Camera Settings Extension kann eine App eine kundenspezifische Schnittstelle zur Verfügung stellen und Kameraoptionen und Effekte auszuwählen, wenn eine Kamera verwendet wird. (Microsoft, 2013d)

Die Extension Contact Picker ermöglicht es, Kontaktdaten der App zu listen, wenn der Benutzer Zugang zu seinen Kontakten braucht. (Microsoft, 2013d)

File Activation Extension ermöglicht es, Dateitypen wie .txt zu verwenden bzw. einen neuen Dateitypen zu schaffen. Die Dateiaktivierungserweiterung ermöglicht es, einen neuen Dateityp oder ein Verzeichnis zu definieren, um einen Dateityp zu behandeln.

Weitere Extensions sind: Print Task Settings, AutoPlay, Account Picture Provider, Game Explorer, Protocol Activation und SSL/Certificates. (Microsoft, 2013d)

3.3.6.2 Windows Phone 8

Ein Chooser ist eine API, über welche der Zugriff auf die Kontakt-App oder die Kamera App möglich ist. Wenn die neue App geöffnet wird, kann der Benutzer entscheiden, ob er diese nun auch verwendet oder nicht. Wird die App geschlossen, wird die eigentliche App reaktiviert und empfängt den entsprechenden Status und die Daten (Foto). (Microsoft, 2013m)

Ein Launcher ist ebenso eine API, die eine bereits installierte App wie den WWW-Browser startet. Auch hier kann der Benutzer nach dem Öffnen der App entscheiden ob er diese verwendet oder nicht. So wie bei einem Chooser kehrt der Benutzer beim Schließen der App wieder zur reaktivierten zurück, von der die neue App geöffnet wurde. Im Gegenteil jedoch zum Chooser werden hier jedoch keine Daten oder ein Status übergeben. (Microsoft, 2013m)

Ein Launcher eignet sich also für Vorgänge von welchem die App keine Daten benötigt. Möchte man zum Beispiel eine Website in der App verlinken, wird ein Launcher implementiert.

Wird mittels einem Chooser oder Launcher eine andere App geöffnet, wird die vorherige App deaktiviert oder kommt in den `Tombstoned` Status. (Microsoft, 2013m)

3.3.6.3 Kamera Beispiel

Als Beispiel wird die unterschiedliche Implementierung der Kamera auf beiden Plattformen gezeigt:

Listing 4: Kameraimplementierung Windows 8 - XAML

```
<Page
    …
    <Grid Background="{StaticResource
ApplicationPageBackgroundThemeBrush}">
        <StackPanel HorizontalAlignment="Left" Height="100"
VerticalAlignment="Top" Width="100"/>
        <StackPanel Margin="90,85,90,-85">
            <TextBlock FontSize="42">
             <Run Text="Kamera - Beispiel"/>
            </TextBlock>
            <Image x:Name="picture" Height="550"/>
            <Button x:Name="btnMakePicture" Content="Aufnahme"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
Click="btnMakePicture_Click"/>
        </StackPanel>
    </Grid>
</Page>
```

Listing 5: Kameraimplementierung Windows 8 –Code

```
//Für die Foto Aufnahme folgende Namespaces hinzufügen
using Windows.Media.Capture;
using Windows.Storage;
using Windows.Storage.Streams;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Media.Imaging;
// Funktion wenn der Button gedrückt wird
private async void btnMakePicture_Click(object sender,
RoutedEventArgs e)
{
   CameraCaptureUI cameraCaptureDialog = new CameraCaptureUI();
   Size croppedAspectRatio = new Size(16, 9);
   cameraCaptureDialog.PhotoSettings.CroppedAspectRatio =
   croppedAspectRatio;

   StorageFile storageFile = await
   cameraCaptureDialog.CaptureFileAsync(CameraCaptureUIMode.Photo);

   if (storageFile != null)
   {
      BitmapImage bitmapImage = new BitmapImage();
      using (IRandomAccessStream randomFileStream = await
      storageFile.OpenAsync(FileAccessMode.Read))
      {
      bitmapImage.SetSource(randomFileStream);
      }
      picture.Source = bitmapImage;
   }
}
```

Die gesamte Funktion wird dann ausgeführt, wenn der Benutzer auf den „Aufnahme“ Button klickt und das Event `btnMakePicture_Click` auslöst. Es wird zuerst eine Instanz der Klasse `CameraCaptureUI` mit einem Dialog (Aufnahmedialog) erzeugt. Das Size Objekt wird für den Bildausschnitt benötigt und erhält die Eigenschaft `CroppedAspectRatio`.

Die Methode `CaptureFileAsync` wird erst dann ausgeführt, wenn der Benutzer sich entschließt, das Foto zu speichern. Ist dies der Fall, wird eine Instanz der Klasse Bitmap `bitmapImage` angelegt.

Da man einem leeren `ImageControl` kein Bild direkt übergeben kann, wird ein `IRandomAccessStream` deklariert und die `OpenAsync(FileAccessMode.Read)` liest das Foto von der `storageFile` Instanz aus und dient als Quelle für die `ImageBitmap` Instanz. Erst dann kann das Bild der Source des `ImageControls`

zugewiesen werden um in der App nach dem Schließen des Dialogs angezeigt werden. (Bleske, 2013, p. 274)

Ist auf dem Gerät keine Kamera angeschlossen, erscheint ein einfacher Hinweis „Schließen Sie eine Kamera an.“.

Listing 6: Kameraimplementierung Windows Phone 8 - XAML

```
<phone:PhoneApplicationPage
    …
   <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
      <TextBlock Text="Kamera Beispiel" Style="{StaticResource
      PhoneTextNormalStyle}" Margin="12,0"/>
      <TextBlock Text="Kamera" Margin="9,-7,0,0" Style="{StaticResource
      PhoneTextTitle1Style}"/>
   </StackPanel>

   <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
      <Image x:Name="picture" Height="550" Margin="0,-1,0,90"/>
      <Button x:Name="btnMakePicture" Content="Aufnahme"
      HorizontalAlignment="Stretch" VerticalAlignment="Bottom"
      Click="btnMakePicture_Click" Height="100"/>
   </Grid>
   …
</phone:PhoneApplicationPage></Page>
```

Listing 7: Kameraimplementierung Windows Phone 8 –Code

```
//Für die Foto Aufnahme folgende Namespaces hinzufügen
using Microsoft.Phone.Tasks;
using System.Windows.Media.Imaging;

public partial class MainPage : PhoneApplicationPage
{
   // Task Objekt vorm Konstruktor deklarieren
   CameraCaptureTask cameraCaptureTask;

   // Konstruktor
   public MainPage()
   {
      InitializeComponent();

      // Initialiseren des Task Objektes
      cameraCaptureTask = new CameraCaptureTask();
      cameraCaptureTask.Completed += new
      EventHandler<PhotoResult>(cameraCaptureTask_Completed);
   }

   void cameraCaptureTask_Completed(object sender, PhotoResult e)
   {
      if (e.TaskResult == TaskResult.OK)
      {
```

```
        MessageBox.Show(e.ChosenPhoto.Length.ToString());

        // Anzeigen des Fotos in einem ImageControl
        BitmapImage bitmapImage = new BitmapImage();
        bitmapImage.SetSource(e.ChosenPhoto);
        picture.Source = bitmapImage;
      }
   }

   private void btnMakePicture_Click(object sender, RoutedEventArgs e)
   {
      cameraCaptureTask.Show();
   }
}
```

3.3.7 Hintergrundaufgaben

Möchte man innerhalb einer App Musik streamen, eine Datei herunterladen oder nach Updates fragen, so gibt es die Möglichkeit, sogenannte Backgroundtasks (Hintergrundaufgaben) zu erstellen, welche ausgeführt werden können, auch wenn die App geschlossen ist, da sie App unabhängig sind und unter Kontrolle des Betriebssystems stehen. Dadurch können bspw. LiveTiles aktualisiert werden, ohne geöffnet zu sein. (Reyes, 2013)

Diese Möglichkeit steht unter beiden Plattformen zur Verfügung und es wird unterschieden zwischen periodischen und ressourcenintensiven Tasks. Periodische Tasks sind jene, die das Betriebssystem alle 30min ausführen kann. Ressourcenintensive Tasks sind längere Aufgaben, die ausgeführt werden, wenn das Gerät geladen wird oder über genügend Batteriekapazität verfügt. (Reyes, 2013)

Unter Windows Phone 8 muss die Verwendung einer Hintergrundaufgabe nicht im Manifest deklariert werden wie unter Windows 8.

Listing 8: Hintergrundaufgabe Windows 8

```
using Windows.ApplicationModel.Background;

public class SimpleBackgroundTask : IBackgroundTask
{
   public static BackgroundTaskRegistration RegisterBackgroundTask
   (string taskEntryPoint, string taskName, IBackgroundTrigger trigger,
   IBackgroundCondition condition)
   {
      // Überprüft ob Task schon registriert wurde.
      foreach (var cur in BackgroundTaskRegistration.AllTasks)
      {
```

```
        if (cur.Value.Name == taskName)
        {
           // Task ist bereits registriert.
           return (BackgroundTaskRegistration)(cur.Value);
        }
     }

     // Falls Task noch nicht registriert ist wird er es jetzt.
     var builder = new BackgroundTaskBuilder();

     builder.Name = taskName;
     builder.TaskEntryPoint = taskEntryPoint;
     builder.SetTrigger(trigger);

     if (condition != null)
     {
        builder.AddCondition(condition);
     }

     BackgroundTaskRegistration task = builder.Register();
     return task;
  }

  public void Run(IBackgroundTaskInstance task)
  {
     // TODO
  }
}
```

Für Windows 8 wird die Hintergrundaufgabe in der leeren App Vorlage geschrieben. In Windows Phone 8 gibt es dafür eine eigene Vorlage: Windows Phone-Agent für geplante Aufgaben.

Listing 9: Periodische Hintergrundaufgabe Windows Phone 8

```
using Microsoft.Phone.Scheduler;

namespace ScheduledTaskAgent
{
    public class ScheduledAgent : ScheduledTaskAgent
    {
        // ScheduledAgent-Konstruktor, initialisiert den
        // UnhandledException-Handler
        static ScheduledAgent()
        {
            // Handler für verwaltete Ausnahmen abonnieren
            Deployment.Current.Dispatcher.BeginInvoke(delegate
            {
                Application.Current.UnhandledException +=
                UnhandledException;
```

```
            });
        }

        // Code, der bei nicht behandelten Ausnahmen ausgeführt wird
        private static void UnhandledException(object sender,
        ApplicationUnhandledExceptionEventArgs e)
        {

        // Diese Methode wird aufgerufen, wenn eine regelmäßige oder
        // ressourcenintensive Aufgabe aufgerufen wird
        protected override void OnInvoke(ScheduledTask task)
        {
            //TODO: Code zum Ausführen der Aufgabe im Hintergrund
            NotifyComplete();
        }
    }
}
```

Die folgende Abbildung zeigt den Prozess des Registrierens und Launch einer Hintergrundaufgabe. Wenn eine App eine Hintergrundaufgabe startet wird die Klasse `BackgroundTaskBuilder` verwendet um diese zu registrieren. (Microsoft, 2012c)

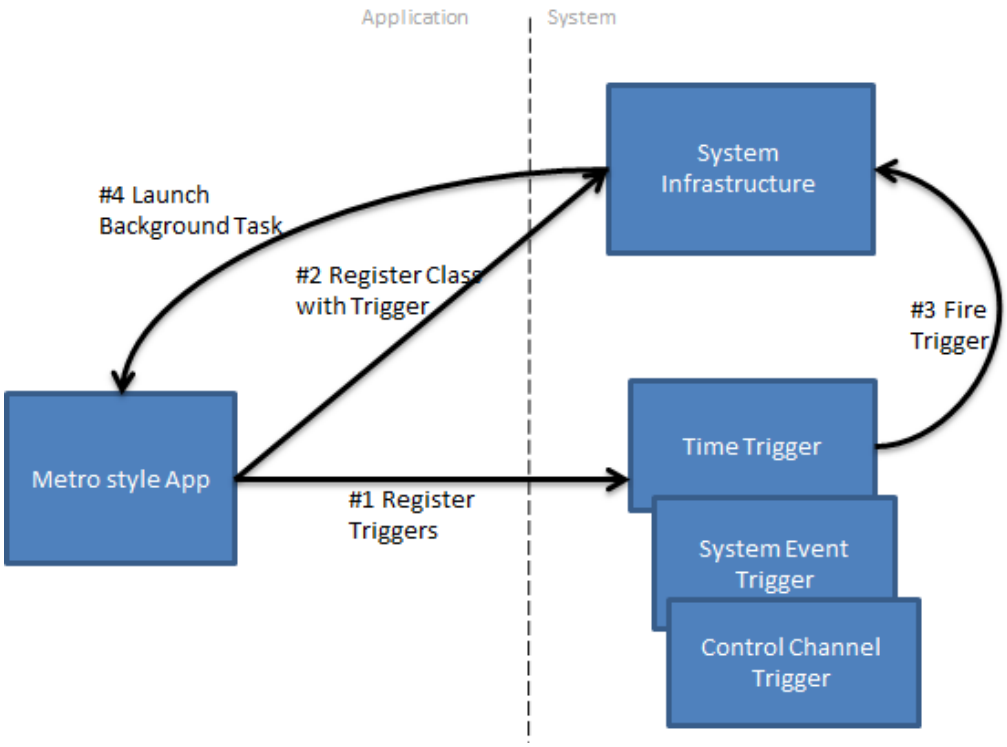

Abbildung 21: Registrieren/Abschließen Hintergrundaufgabe (Microsoft, 2012c)

3.3.8 Kacheln (Tiles)

Eine jede App wird bei Windows 8 als Kachel auf dem neuen Windows Modern UI dargestellt. Prinzipiell werden alle Apps mittels einer statischen Kachel oder einer dynamischen (LiveTile) angezeigt. Auch Win 32/.NET-Anwendungen erzeugen eine Kachel (als Verknüpfung zum Programm auf dem herkömmlichen Desktop) auf dem Windows Modern UI.

3.3.8.1 Standardkachel

Wie die Kachel einer App aussieht, wird durch die Einstellungen im Manifest der App festgelegt. Hierzu zählen Einstellungen wie die verschiedenen Logogrößen, der Titel, die Ausrichtungen, die Sprache sowie eine Beschreibung. (Christian Bleske 2013, S. 184) Diese Datei gibt es zwar auf beiden Plattformen, jedoch sind unterschiedliche Einstellungen notwendig.

Unter beiden Plattformen ist ein Anzeigename und eine Beschreibung zu hinterlegen. Auch Bilder können hinterlegt werden, jedoch in verschiedenen Größen.

Beim Punkt Visuelle Anlagen unter Windows 8 wird durch ein Standardlogo dem Entwickler angezeigt, welche Logogrößen auf jeden Fall benötigt werden. Und zwar muss das Logo in den Größen 150x150px für die Standardansicht am Windows 8 Modern UI, 30x30px für die Liste bei Suchergebnissen oder in der Liste „Alle Programme“ und 50x50px für das Store-Logo und für den Startbildschirm 620x300px vorliegen. Ein breites Bild von 310x150px ist optional möglich.

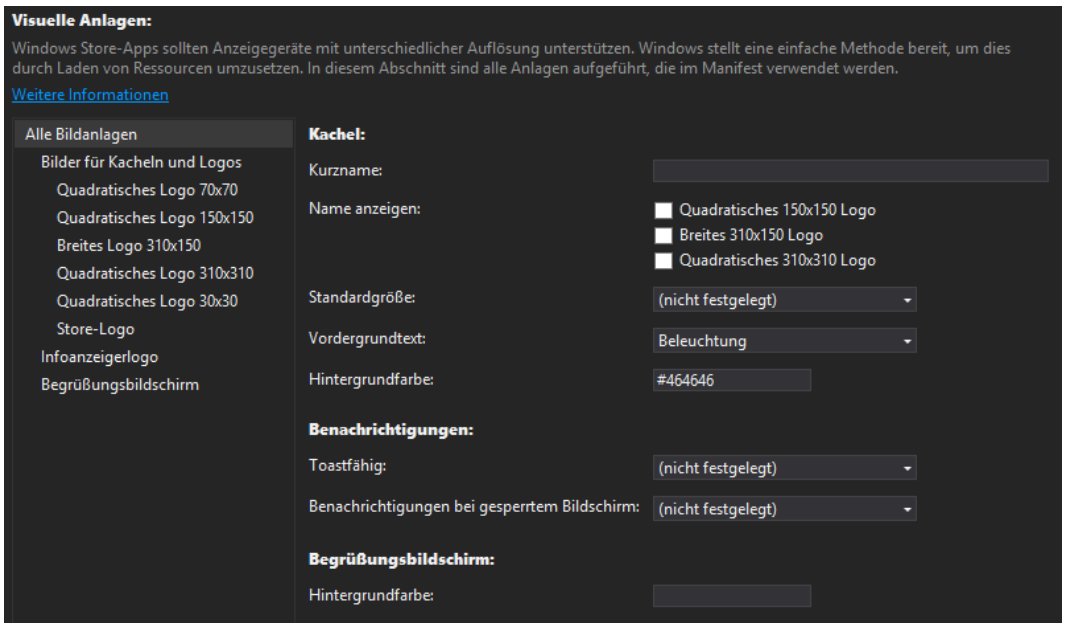

Abbildung 22: Windows 8 Logovarianten im Manifest

Zu jeder Logo-Art können auch noch weitere Skalierungen angelegt werden, um bei den unterschiedlichsten Auflösungen trotzdem ein nicht verzerrtes Logo darzustellen. Wann welche Skalierung verwendet wird, entscheidet das System selbst.

Unter dem Windows Phone 8 ist die Logogröße vom gewählten Template abhängig, welches gewählt werden muss und dieses im Code nicht mehr änderbar ist. Es gibt zur Auswahl das `TemplateFlip`, `TemplateCycle` und das `TemplateIconic`.

Für die ersten beiden werden die Logos in den Größen 159x159px (Small), 336x336px (Medium) und 691x336px (Wide) benötigt, für `TemplateIconic` in den Größen 110x110px (Small) und 202x202px (Medium). (Microsoft, 2013n)

Die gewohnte Manifest Ansicht von Windows 8 ist unter anderem im Ordner Properties zu finden und nennt sich hier `WMAppManifest.xml`. Es ist jedoch auch möglich den Code selbst in die Datei `AppManifest.xml` zu schreiben.

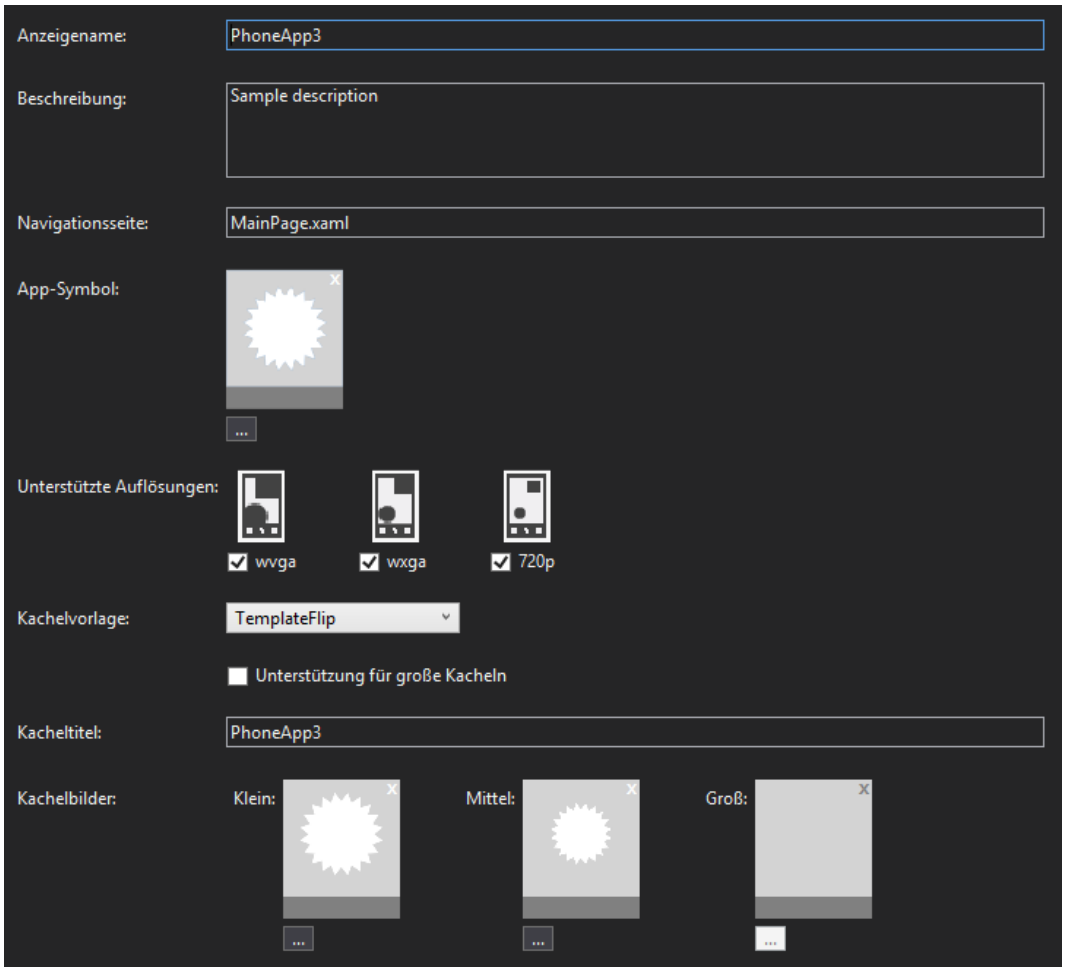

Abbildung 23: Windows Phone 8 Manifest (grafisch)

Das `TemplateFlip` basiert auf Windows Phone 7 und ist daher so manchen Benutzern und Entwicklern bekannt, beim `TemplateCycle` können bis zu 9 Bilder im Kreis nacheinander dargestellt werden, das `TemplateIconic` zeigt ein Logo auf der Kachel.

Abbildung 24: `TemplateIconic`

Abbildung 25: `TemplateFlip`

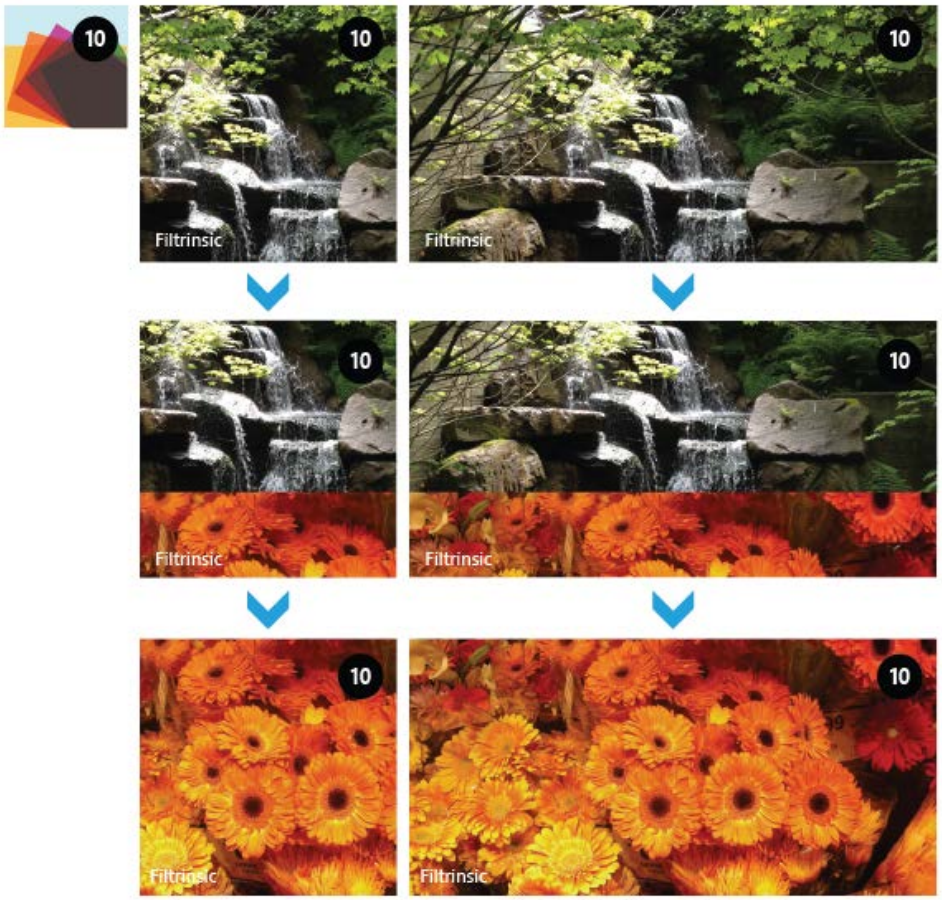

Abbildung 26: `TemplateCycle`

3.3.8.2 LiveTile

Beide Plattformen unterstützen Live-Kacheln, auf denen Inhalte periodisch aktualisiert werden, aber die APIs sind völlig unterschiedlich.

Bei einer App mit einem LiveTile handelt es sich um eine normale Kachel wie bereits vorher beschrieben mit dem einzigen Unterschied, dass Live Informationen bereits innerhalb der Kachel am Windows Modern UI dargestellt werden können. Bestes Beispiel hierfür ist die Wetter-App. Diese zeigt das aktuelle Wetter, sowie die zugehörige Temperatur auf dem Windows 8 Modern UI selbst an und man muss nicht erst in die Wetter-App selbst wechseln.

Je nachdem welche Live Informationen dem Benutzer unter Windows 8 darstellen möchte, muss man das richtige Format der Kachel wählen - neben reinen Textinformationen ist auch die Anzeige von Bilder oder Animationen möglich. Wenn zum Beispiel Bild und Text angezeigt werden soll, eignet sich das Format von 310x150px. Bilder haben im JPG oder PNG Format vorliegen, um verwendet werden zu können. (Bleske, 2013, p. 181)

Microsoft bietet für Windows 8 über die `TileTemplateType Enumeration` mehr als 40 Vorlagen an wie eine LiveTile aussehen und strukturiert sein kann.

Der Benutzer kann unter Windows 8 Kachelbenachrichtigungen jederzeit deaktivieren und aktivieren. Aus diesem Grund kann der Entwickler im Manifest trotzdem eine statische Kachel angegeben. Unter Windows Phone 8 kann der Benutzer die Kachel vollständig vom Windows Modern UI entfernen. (Reyes, 2013)

Es wird jedoch empfohlen LiveTiles in die Apps zu implementieren, um den Benutzer in diese zu locken und ihm Informationen zur Verfügung zu stellen, ohne dass die App gestartet werden muss.

Unter Windows 8 wird das aktualisieren der Kachelmithilfe von `TileUpdateManager` und `TileNotification (Windows.UI.Notifications)` ermöglicht. Der Text muss in Form eines XML-Text-Nodes vorhanden sein. Unter Windows Phone 8 erfolgt das aktualisieren mit Hilfe von `ShellTile.Update (Microsoft.Phone.Shell).` Pushbenachrichtigung erfolgt mit Hilfe von `ShellTileSchedule`. (Reyes, 2013)

Toastbenachrichtigungen sind ebenfalls unter beiden Plattformen möglich und können dem Benutzer über ein Ereignis informieren. Diese können auch als Hintergrundaufgabe umgesetzt werden. Mithilfe dieser ist es bspw. möglich auf der Kachel die Anzahl der eingegangenen Chat-Nachrichten anzuzeigen.

Windows Phone 8 unterstützt zwei Arten von Benachrichtigungen: Alarm und Erinnerung. Mit `ShellToast` können auch Toasts angezeigt werden. Alarme und Erinnerungen sind in `Microsoft.Phone.Scheduler` enthalten“. (Reyes, 2013)

Windows 8 verwendet Toastbenachrichtigungen mithilfe von `ToastNotificationManager` und `ScheduledToastNotification`. (Reyes, 2013)

3.3.8.3 Win 32/.NET-App (Verknüpfung)

Eine jede App, welche man unter Windows 8 installiert legt auf dem Windows Modern UI eine Verknüpfung an, ähnlich wie die Verknüpfung zum herkömmlichen Desktop. Die neue Windows Modern UI ersetzt also das eigentlich das bisher bekannte Startmenü bis Windows 7. Dies ist unter der neuen Windows 8.1 Version deutlicher dargestellt als noch unter Windows 8, wo es teilweise sehr schwierig war, bestimmte bekannte Programme/Einstellungen aufzurufen bzw. zu finden. Da es am Windows Phone keine Win 32/.NET-Anwendung gibt, sind diese Kacheln natürlich dort nicht vorhanden.

3.3.9 Teilen von Code

Wenn man für verschiedene Plattformen entwickelt, ist ein Ziel für den Entwickler wieder verwendbaren Code zu schreiben. Dies ist bei der Entwicklung einer Windows 8 bzw. Windows Phone 8 App möglich.

Hierzu hat Microsoft die Möglichkeit einer PCL (=Portable Class Library) geschaffen, welche es ermöglicht Code in beide Plattformen wiederverwenden zu können.

Jedoch muss auch hier beachtet werden, dass nicht alle Funktionen beide Plattformen unterstützen. Nur die Core Klassen sind überall verfügbar. (Bleske, 2013, p. 309)

Aus einer PCL wird eine einzelne Binärdatei und kann somit keine Direktiven zur bedingten Kompilierung enthalten. Es werden plattformspezifische Funktionen mithilfe von Schnittstellen oder abstrakten Klassen abstrahiert. Wenn portabler Code mit plattformspezifischem Code interagieren muss, werden Entwurfsmuster mit Abhängigkeitsinjektion verwendet, die plattformspezifische Implementierungen der Abstraktionen bereitstellen. (Holland, 2013)

Mit dem MVVM-Entwurfsmuster werden die Ansichtsmodelle und Modelle zusammen mit Abstraktionen plattformspezifischer Funktionen in der PCL zusammengefasst. Die Windows Store-Apps und die Windows Phone-Apps stellen die Startlogik, Ansichten und Implementierungen aller Abstraktionen plattformspezifischer Funktionen bereit. (Holland, 2013)

4 Entwicklungsumgebungen

Um eine Windows Anwendung oder App, egal welche, programmieren zu können, stellt Microsoft Entwicklungsumgebungen kostenpflichtig und kostenlos zur Verfügung.

Dabei handelt es sich um die Entwicklungsumgebung Visual Studio, welche es kostenpflichtig je nach den eigenen Anforderungen in den Versionen Professional, Premium und Ultimate gibt. Der Vorteil dieser Versionen ist, dass alle Windows Anwendungen wie fürs Web, für Windows 8, fürs Windows Phone und für den Windows Desktop in einer Version entwickelt werden können.

Kostenlos handelt es sich um die Express Versionen vom Visual Studio, hiervon gibt es um einige mehr, da es für jede Anwendung eine eigene Version gibt. Das heißt, um eine Windows 8 Anwendung und eine Windows Phone 8 Anwendung zu programmieren, benötigt man zwei verschiedene Visual Express Versionen, nämlich Visual Studio Express 2012 für Windows 8 und Visual Studio Express 2012 für Windows Phone.

Zusätzlich benötigt man eine Entwickler Lizenz und, um die Anwendung im Windows Store veröffentlichen zu können, noch einen Windows Entwickler Account. Beides lässt sich einfach über alle Visual Studio Versionen verwalten.

Für die Programmierung der Beispiele in dieser Arbeit wurde sogleich ie neue Visual Studio 2013 Professional Preview verwendet.

Diese gibt es für Bildungseinrichtungen und deren Mitglieder (Lehrkörper, Schüler/Studenten) kostenlos von Microsoft, wenn man sich bei Microsoft DreamSpark (www.dreamspark.com) registriert, verifiziert und das Konto mit seinem Microsoft Konto verknüpft. Auch bekommt man einen Code um ein kostenloses Entwickler Konto anzulegen, welchen man benötigt um eine App zu zertifizieren und im Windows Store zu laden bzw. zu veröffentlichen.

4.1 Visual Studio 2013 Professional Preview

Bei dieser Version ist natürlich auch das Designtool Blend mit an Bord, mit welchem das Design einer Anwendung grafisch gestaltet werden kann. Die Programmlogik ist aber mit dem Visual Studio zu erstellen, da sich hierfür Blend nicht eignet.

4.1.1 Installation

Bei der Auswahl der benötigten Ressourcen bei der Installation ist das Windows Phone SDK automatisch deaktiviert. Dieses muss aktiviert werden. Beim ersten Start kann man noch sein Microsoft Konto angeben, das Farbschema und die Entwicklungseinstellungen auswählen. Allgemeine Entwicklungseinstellungen wurden erstmals eingestellt.

4.1.2 Entwickler Lizenz

Beim ersten Start vom Visual Studio erscheint automatisch ein Dialog Fenster um eine Entwickler Lizenz zu beantragen. Nur mit dieser ist es möglich nicht zertifizierte Apps auszuführen, Apps zu zertifizieren und in den Windows Store hochzuladen. Mit einem Microsoft Konto muss diese alle 30 Tage, mit einem Entwicklerkonto alle 90 Tage erneuert werden.

Der Dialog kann auch über den Menü Pfad Projekt→Store→Entwicklerlizenz abrufen aufgerufen werden.

4.1.3 Entwickler Konto

Um die App auf die Anforderungen und Richtlinien von Microsoft zu testen und eventuell zu zertifizieren benötigt man des Weiteren ein Entwickler Konto bei Microsoft. Als Einzelperson würde das Konto (derzeit 2013) € 37,- jährlich kosten, durch DreamSpark und einen dort angeforderten Code ist es kostenlos. Nachteil ist jedoch, dass man trotz des Codes seine Kreditkartendaten angeben muss.

Ist das Entwickler Konto eingerichtet sind alle Voraussetzungen für die Entwicklung einer Windows 8 App oder einer Windows Phone 8 App gegeben.

4.2 Blend für Visual Studio

Blend ist ein zusätzliches Werkzeug zum Visual Studio und wurde speziell für Designer konzipiert, um visuell eine App entwickeln und das Gestalten der Benutzeroberfläche zu ermöglichen.

Auch ist es möglich einfache Animationen innerhalb der GUI zu erzeugen oder Objekte in 3D zu darzustellen.

Da Blend ein Werkzeug für Designer ist, unterstützt es natürlich auch den Import von Adobe Photoshop (*.psd) Dateien.

4.3 Vorlagen

Für beide Plattformen gibt es verschiedenen Vorlagen. Diese liegen in gängigen Layout- und Interaktionsmodellen vor. Vorlagen unterscheiden sich je Plattform und gewählter Programmiersprache.

5 Fazit

Diese Arbeit beschäftigte sich mit der Fragestellung wie „Wie groß ist der Unterschied zwischen einer Windows 8 Desktop und einer Windows 8 Phone App für Webentwickler?“

Als Fazit dieser Arbeit hat sich herausgestellt, dass beide Apps unter verschiedenen Plattformen laufen und der Unterschied laut meinen Recherchen und die Programmierung eigener kleinen Codefragmente als Arbeitshypothesen recht groß ist.

Es ist zwar möglich, für beide Plattformen eine Library zu verwenden, jedoch können selbst hier nicht alle Funktionen verwendet werden, da nicht alle Funktionen von jeder Plattform unterstützt werden.

Komplett unterschiedlich ist eigentlich alles, was mit der Benutzeroberfläche zu tun hat. Die Kacheln werden unterschiedlich definiert, die Design Anforderungen sind andere, es wird verschieden navigiert und die User Experience ist unterschiedlich.

Im technischen Bereich liegen die Unterschiede im Speichermanagement, im Lebenszyklus einer App, in den Hintergrundaufgaben, Contracts/Extensions, (API-)Architektur. So ist zum Beispiel der Zugriff auf eine Kamera in beiden Plattformen unterschiedlich zu implementieren.

Wenn man diese Unterschiede berücksichtigt und die Möglichkeit der Wiederverwendung von Code durch eine Library verwendet kann für beide Plattformen entwickelt werden. Als Entwickler ist es von Vorteil, wenn man sich auf diesen beiden Plattformen gut auskennt um im Vorhinein den Aufwand und die Möglichkeiten abschätzen zu können.

Microsoft ist auf dem Weg, diese beiden Plattformen so gut wie möglich zu verbinden, denn selbst zwischen einer unter Windows 8 programmierten App gibt es schon Unterschiede zur neuen Version Windows 8.1.

Eine Frage in absehbarer Zeit könnte lauten, wie sehr die Programmierung einer App für beide Plattformen vereinheitlicht wurde.

Quellenverzeichnis

Bleske, C. (2013). *Windows 8 Apps entwickeln - Apps für die neue Oberfläche von Windows 8 - Entwickeln mit HTML5, JavaScript, XAML und C# - Von den Grundlagen der Programmiersprachen bis zur fertigen App.* Deutschland: Franzis Verlag GmbH.

Byrne, A., & Rothaus, D. (01. November 2012). *Channel 9*. Abgerufen am 14. Juli 2013 von How to Leverage your Code across WP8 and Windows 8 (Repeat): http://channel9.msdn.com/Events/Build/2012/3-043R

Holland, D. (10. August 2013). *MSDN Magazin*. Abgerufen am 10. August 2013 von Gemeinsamer Code für Windows Phone 8- und Windows 8-Anwendungen: http://msdn.microsoft.com/de-de/magazine/dn201744.aspx

Immler, C. (2012). *Der App-Entwickler Crashkurs - Die wichtigsten Entwicklungsumgebungen und Frameworks zur App-Programmierung.* Deutschland: Franzis Verlag.

Intel Software. (14. Juli 2013). *Intel Software*. Abgerufen am 14. Juli 2013 von Windows 8* Store vs Desktop App Development: http://software.intel.com/de-de/articles/windows-8-store-vs-desktop-app-development#windowsapps

Microsoft. (2012a). Windows 8 - User experience guidelines.

Microsoft. (2012b). Windows 8 - Produkthandbuch für Entwickler. Deutschland: Microsoft Corporation.

Microsoft. (2012c). Introduction to background tasks - Guidelines for developers. United States of America: Microsoft Corporation.

Microsoft. (02. August 2013a). *Windows Store Apps*. Abgerufen am 02. August 2013 von Lernen Sie die Welt der Windows Store-Apps kennen: http://msdn.microsoft.com/de-de/library/windows/apps/hh974576.aspx

Microsoft. (02. August 2013b). *Windows Phone*. Abgerufen am 02. August 2013 von Phones: http://www.windowsphone.com/de-de/phones

Microsoft. (02. August 2013c). *Windows Phone Development Center*. Abgerufen am 02. August 2013 von First look at Windows Phone:

http://msdn.microsoft.com/de-de/library/windowsphone/design/hh202905%28v=vs.105%29.aspx

Microsoft. (03. August 2013d). *Dev Center - Windows Store apps*. Abgerufen am 03. August 2013 von App contracts and extensions (Windows Store apps): http://msdn.microsoft.com/en-us/library/windows/apps/hh464906.aspx

Microsoft. (05. August 2013e). *Microsoft*. Abgerufen am 05. August 2013 von Microsoft-Designprinzipien: http://msdn.microsoft.com/de-DE/library/windows/apps/hh781237.aspx

Microsoft. (06. August 2013f). *Windows Phone Development Center*. Abgerufen am 06. August 2013 von Central app hub with home page menu (Panorama or Pivot control) for Windows Phone: http://msdn.microsoft.com/en-us/library/windowsphone/design/hh202892%28v=vs.105%29.aspx

Microsoft. (06. August 2013g). *Windows Phone Development Center*. Abgerufen am 06. August 2013 von Central app hub with panel areas (Panorama control) for Windows Phone: http://msdn.microsoft.com/en-us/library/windowsphone/design/hh202885%28v=vs.105%29.aspx

Microsoft. (06. August 2013h). *Windows Phone Development Center*. Abgerufen am 06. August 2013 von App tabs (Pivot control) for Windows Phone: http://msdn.microsoft.com/en-us/library/windowsphone/design/hh202890%28v=vs.105%29.aspx

Microsoft. (02. August 2013i). *Windows Phone Development Center*. Abgerufen am 02. August 2013 von List with details drilldown for Windows Phone: http://msdn.microsoft.com/en-us/library/windowsphone/design/hh202886%28v=vs.105%29.aspx

Microsoft. (06. August 2013j). *Windows Phone Development Center*. Abgerufen am 06. August 2013 von Uniform page shuffle for Windows Phone: http://msdn.microsoft.com/en-us/library/windowsphone/design/hh202908%28v=vs.105%29.aspx

Microsoft. (06. August 2013k). *Windows Store Apps*. Abgerufen am 06. August 2013 von Navigationsdesign für Windows Store-Apps: http://msdn.microsoft.com/de-de/library/windows/apps/hh761500.aspx

Microsoft. (10. August 2013l). *Windows Phone Development Center*. Abgerufen am 10. August 2013 von Windows Phone API reference: http://msdn.microsoft.com/en-

us/library/windowsphone/develop/ff626516%28v=vs.105%29.aspx#BKMK_NETAPIforWindowsPhone

Microsoft. (10. August 2013m). *Windows Phone Development Center*. Abgerufen am 10. August 2013 von Launchers and Choosers for Windows Phone: http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769542%28v=vs.105%29.aspx

Microsoft. (10. August 2013n). *Windows Phone Development Center*. Abgerufen am 10. August 2013 von Tiles for Windows Phone: http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202948%28v=vs.105%29.aspx

Neumann, J. (2013). *Windows Store Apps entwickeln mit C# und XAML.* Deutschland: Microsoft Press.

Novak, I., Zoltan, A., & Fülöp, D. (2013). *Windows Store Apps entwicklen.* Deutschland: Wiley VCH Verlag GmbH.

Quandt, R. (06. August 2013). *WinFuture*. Abgerufen am 06. August 2013 von Microsoft: Windows Phones mit Intel-CPU möglich: http://winfuture.de/news,73988.html

Reyes, J. (10. August 2013). *MSDN Magazin*. Abgerufen am 10. August 2013 von Erstellen von Apps für Windows 8 und Windows Phone 8: http://msdn.microsoft.com/de-de/magazine/dn296517.aspx

Riga, B. (26. Februar 2013). Building Apps for Both Windows 8 and Windows Phone 8: (01a). *Comparing Windows 8 and Windows Phone 8, Part 1*. (Microsoft, Hrsg.)

Scheer, O. (November 2012). Das geht App. *windows.developer*(11).

Shapiro, M., & Burtoft, J. (09. Mai 2013). Developing in HTML5 and Javascript for Windows Phone.

Walther, S. (2012). *Windows 8 Apps with HTML5 and JavaScript - Unleashed.* United States of America: SAMS.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis