# Einleitung

Super! Die Idee für eine neue tolle Windows 8 App ist gefunden und somit ein wichtiger Schritt schon mal getan. Bevor Sie sich nun aber Hals über Kopf in die Entwicklung stürzen, sollte Sie noch einmal kurz überlegen, was Sie von einer App erwarten, damit Sie eine ihrer Lieblings-Apps wird. Natürlich soll sie ihren Zweck erfüllen, also genau die von Ihnen benötigen UseCases abdecken. Aber damit allein ist es nicht getan. Damit es wirklich Spaß macht mit einer App zu arbeiten, muss sie auch schnell starten, sofort reagieren, nie abstürzen, intuitiv bedienbar sein, gut aussehen, immer funktionieren, usw. Nicht ganz einfach, oder? Selbst Apps der großen Social Media Plattformen können diese Erwartungen nicht immer erfüllen und werden mit schlechten Bewertungen bestraft. Vielleicht kennen Sie das ja auch schon: Sie sitzen gerade im Zug und wollen sich die Neuigkeiten aus Ihrem Bekanntenkreis anschauen, müssen aber gefühlte Stunden warten, bis überhaupt etwas auf dem Bildschirm angezeigt wird – wenn nicht gerade noch ein Tunnel dazwischen kommt. Große Anbieter können sich so etwas vielleicht leisten, sofern Sie aber gerade erst in den Markt einsteigen wollen, muss der erste Wurf sitzen. Also verabschieden wir uns von der idealen Welt, in der wir immer online sind und Webdienste nie ausfallen und kümmern uns um die Offlinefähigkeit Ihrer Anwendung. Die Frage „Wie stelle ich eigentlich fest, ob ich offline bin?“ wird im Kasten „Offlinetypen“ behandelt.

# Methodisches Vorgehen

Wie bei jeder guten Anwendung steht am Anfang der Entwicklung das Requirements- und Usability-Engineering. Auch im Bereich der Offlinefähigkeit sind diese Disziplinen unabdingbar. Sie sollten genau überlegen, welche UseCases offline für Ihre App überhaupt Sinn haben und was im Offline-Modus angezeigt werden soll. Können Sie entscheiden, welche Informationen offline relevant sind oder sollte die Entscheidung dem Anwender überlassen werden? Eine App, die zum Beispiel auf StackOverflow Suchen durchführt, kann offline nicht den vollen Funktionsumfang bieten. Es wäre aber vorstellbar, schon früher gefundene und evtl. markierte Diskussionen (Anwenderentscheidung) offline zu speichern und diese dem Nutzer zu präsentieren. Wahrscheinlich wären für diesen Anwendungsfall auch sehr unterschiedliche Views für den Online- und Offline-Modus notwendig, so dass auch geklärt werden muss, wann man zwischen den Modi hin und her wechselt. Kann dies automatisch geschehen? Im Zweifelsfall sollten Sie dem Anwender die Kontrolle überlassen, denn nichts nervt mehr, als wenn die App ständig automatisch ihre Views umschaltet. Sollte Ihre App für beide Modi dieselben Views verwenden, ist es dennoch wichtig über den Status der Internetverbindung zu informieren. Sie wollen natürlich wissen, ob Sie gerade über die aktuellsten oder nur ältere Daten verfügen. Generell eignet sich jedoch Content, welcher wenigen Änderungen unterworfen ist, gut für die Offline-Verwaltung.

Als weiteres Beispiel – auch für den Lernprozess den man bei der Entwicklung einer guten App durläuft – soll hier die Xing-App für Windows Phone 7 herangezogen werden. Der erste Entwurf sah vor, dass viele Entitäten, wie Kontakte, Nachrichten oder Neuigkeiten aus dem Sozialen Netzwerk zwar offline gespeichert werden, dies aber nur für schon betrachtete Einträge. Bei internen Tests zeigte sich schnell, dass für es für die Anwender vollkommen intransparent und frustrierend war, wenn einzelne Kontakte offline zur Verfügung standen, andere aber nicht. Die Lösung war möglichst alle Entitäten offline zu speichern. Nur zu welchem Zeitpunkt muss dies geschehen? Sollten alle Kontakte geladen und gespeichert werden, wenn man das erste Mal die Kontaktübersicht öffnet? Das Ergebnis wäre eine recht lange Wartezeit, was bei dem Anwender wiederum für Unmut sorgt. Am ehesten wird die Wartezeit beim ersten Start der Anwendung akzeptiert. Jedoch auch zu diesem Zeitpunkt muss der Anwender informiert werden, was die App gerade tut. In der Xing-App gibt es dafür einen Lade-Bildschirm, welcher genau darüber informiert, was gerade geladen wird und wie weit der Prozess schon fortgeschritten ist. (Abbildung 1)

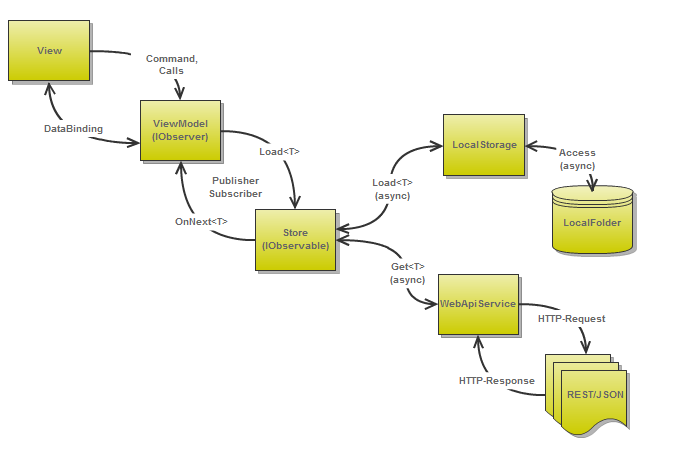
Sind erst einmal alle Daten geladen, ergibt sich ein weiterer Vorteil dieser Vorgehensweise. Beim zweiten Start der Anwendung sind die meisten Daten schon lokal abgelegt und können sehr schnell zugegriffen werden. Die Anwendung startet bedeutend schneller und kann sofort Inhalte anzeigen. Gleichzeitig kann asynchron online nach den neuesten Daten gesucht werden, welche nachträglich den Anwender präsentiert werden. Die Architektur dieses „OfflineFirst“-Ansatzes wird im nächsten Abschnitt genauer erläutert.

Als Hilfestellung für Ihre App hier noch mal die wichtigsten Frage, die Sie sich stellen sollten:

* Was soll offline gespeichert werden? Geschieht dies automatisch oder wird Content explizit durch den Benutzer gespeichert.
* Muss es unterschiedliche Views für den Online/Offline-Fall geben?
* Wie zeige ich den Online-Status an?
* Wie wechselt man den Online/Offline-Modus? Automatisch oder explizit durch den Benutzer?

# Eine mögliche OfflineFirst Architektur

Im folgenden Abschnitt soll ein möglicher Architekturansatz für eine OfflineFirst Strategie aufgezeigt werden. Die dabei verwendeten Klassen werden in Abbildung 2 abgebildet.



Die Beispielarchitektur ist nach dem für WPF Anwendungen allgemein empfohlenen MVVM Pattern umgesetzt. Die Store-Klasse übernimmt hierbei die Rolle des Models. Die ViewModel Klasse selbst registriert sich als Observer über das von Microsoft in .NET4 eingeführt IObservable Interface für alle neuen Daten, die vom Store gelesen werden. Dabei ist es für das ViewModel irrelevant, ob die Daten offline oder online geladen werden. Das ViewModel bekommt alle Daten durch den Aufruf von IObserver.MoveNext geliefert.

Die Klasse Store selbst fragt beim ersten Zugriff zuerst die möglicherweise schon offline Daten ab und liefert diese per Aufruf von OnNext erst einmal aus, bevor der Zugriff auf den WebApi Service erfolgt.

Der Benutzer sieht dadurch die zuletzt gelesenen Daten relativ schnell und muss nicht warten, bis die Daten über den möglicherweise langsamen WebService geladen werden. Wie in Abbildung 3 zu sehen ist, werden die offline geladenen Daten schon angezeigt. Dem Benutzer wird aber durch die Einblendung eines im WinRT Framework ProgressRing genannten Steuerlements suggeriert, das Daten geladen werden.

