

Offline-Nutzung webbasierter Informationssysteme

Kerstin Blumenstein & Grischa Schmiedl

Fachhochschule St. Pölten,

ICM/T – Institut für Creative Media/Technologies

kblumenstein@fhstp.ac.at

Zusammenfassung

Der Zugriff auf das Internet über mobile Endgeräte hat im letzten Jahrzehnt enorm an Bedeutung gewonnen. In Österreich nutzten 2011 37% der Bevölkerung das Internet über mobile Geräte. Der Trend zum verstärkten Einsatz mobil nutzbarer Medien kann auch im Bereich der Lehre beobachtet werden. Webbasierte Systeme haben sich hier in den letzten Jahren als Standard etabliert und können – prinzipiell – auch auf mobilen Geräten verwendet werden.

Betrachtet man die Nutzungsszenarien von Smartphones, so wird deutlich, dass ein mobiles Gerät nicht nur situativ sondern auch geografisch immer und überall genutzt wird. Vor allem abseits urbaner Ballungsräume sowie innerhalb von Gebäuden bekommt der mobile Lernwillige immer wieder Probleme mit der Netzwerkverbindung. Eine flächendeckende Netzaufdeckung für jegliche mobile Netzwerkverbindung wird auch in Zukunft nicht vollständig gewährleistet werden können. Dieses Problem begründet den Wunsch nach Anwendungen, die auch ohne permanente Internetverbindung genutzt werden können. Während dies bei nativen Applikationen üblich ist, sind Webanwendungen dieser Art Raritäten.

Das vorliegende Paper untersucht die Möglichkeiten der Offline-Nutzung für Webapplikationen anhand der Erweiterung eines existierenden mobilen Informationssystems.

Basierend auf Erkenntnissen aus der Implementierung einer Erweiterung für das Framework Mobilot, welche die Anforderungen für dieses Szenario erfüllt, werden Empfehlungen abgegeben, die EntwicklerInnen bei der erfolgreichen Umsetzung einer offline-fähigen Webanwendung unterstützen.

1 Einleitung

Die Popularität des mobilen Internets ist im letzten Jahrzehnt rasant gestiegen. Der Markteinstieg des amerikanischen Computerkonzerns Apple in den

Smartphone-Markt mit der ersten Version des iPhones¹ wird oft als Auslöser zur Akzeptanz der Nutzung des Internets mittels mobiler Geräte angesehen. Unterstützend dafür war zum einen das bereits 2006 eingeführte HSDPA (High Speed Downlink Packet Access), welches höhere Datenübertragungsraten bis 3,6 MBit/s zuließ (mobilkom Austria, 2006), und zum anderen die deutlich vereinfachte Handhabung der entstehenden Geräteklasse Smartphone. Nur drei Jahre später – im Jahr 2011 – nutzten bereits 37% der ÖsterreicherInnen das mobile Internet (WUNDERMANN PXP, 2011). Zahlen aus dem Nachbarland Deutschland² verdeutlichen diese Entwicklung. Während 2006 nur 6% der Bevölkerung das Internet mobil nutzten, waren es 2010 bereits 28,4%. Einer Prognose von PwC – der in Deutschland führenden Wirtschaftsprüfungs- und Beratungsgesellschaft – zur Folge werden im Jahr 2015 mehr als die Hälfte (rund 54%) der Bevölkerung das Internet über mobile Geräte nutzen (PwC, 2011; pdwb, o.J.; PwC, 2010; PwC & Willkofsky Gruen Associates, 2011).

Diese Zahlen zeigen das Potenzial von Anwendungen für mobile Endgeräte. Jedoch sollten die Nutzungsorte von mobilen Geräten beachtet werden. Eine Studie von Kaikkonen (2008) zeigt, dass europäische NutzerInnen das mobile Internet in den Anfängen der Nutzung über mobile Geräte vorwiegend zu Hause einsetzten. Unterwegs, während des Fahrens mit öffentlichen Verkehrsmitteln oder während des Wartens auf diese sowie auf Arbeit oder im Büro wurden als weitere relevante Nutzungssituationen angeführt. Die selben Orte bildeten auch 2012 die Top-4 der beliebtesten Orte zur Nutzung von Smartphones (Ipsos & MMA, 2012). Dabei zeigt sich deutlich, dass der Smartphone-User immer und überall erreichbar sein möchte – ob zu Hause im Gebäude oder im Zug unterwegs quer durchs Land. Und gerade dann hat der mobile User oft mit einer unzureichenden Netzabdeckung zu kämpfen. HSDPA steht selten durchgängig zur Verfügung. Die unzuverlässige und oftmals schwankende Verbindungsqualität erschwert die Nutzung einer Anwendung, die auf eine ständige Internetverbindung angewiesen ist. In solchen Fällen wäre die Möglichkeit zur Nutzung von Webanwendungen auch ohne permanente Internetverbindung vorteilhaft. Bisher ist man dies nur von nativen Applikationen gewohnt.

Die Möglichkeiten der Offline-Nutzung von Webapplikationen werden durch die derzeit noch in Entwicklung stehende HTML-Spezifikation HTML 5 eröffnet. Zu den relevanten Erweiterungen gehören der Application Cache,

1 Das erste Modell des iPhones ist seit März 2008 auf dem österreichischen Markt erhältlich.

2 Aus Österreich liegt ein Vergleich dieser Art nicht vor.

das im HTML-5-Umfeld entstehende Web Storage API sowie die APIs für lokale Datenbanken (Web SQL und Indexed Database API).

Ob und wie diese Erweiterungen in realen Szenarien mit beschränkter Konnektivität tatsächlich sinnvoll genutzt werden können, wird im Zuge dieses Papers durch Beantwortung folgender Fragen gezeigt:

- Wie kann eine Webanwendung nach dem angeführten Szenario (siehe Abschnitt 2) umgesetzt werden?
- Sind die Möglichkeiten, die HTML 5 bietet, geeignet, um die Anforderungen zu erfüllen?
- Welche Probleme können bei der Umsetzung von webbasierten Offline-Anwendungen auftreten und warum?

Diese Fragen werden anhand einer konkreten Umsetzung einer Erweiterung des Frameworks Mobilot beantwortet.

2 Nutzungsszenario

Das folgende Nutzungsszenario zeigt eine typische Nutzung mobiler Informationssysteme zur Lehr- und Lernunterstützung und wird durch die Definition einer Persona beschrieben:

Kein bzw. nur kurzzeitiger Internetzugriff vorhanden

Person A plant mit ihren Kindern eine Wanderung durch einen Naturpark, dabei möchten sie gemeinsam mehr über die Natur und die Umgebung lernen. Person A informiert sich vorab auf der Website des Naturparks und erfährt, dass durch den Naturpark ein Lehrpfad führt. Die Informationen werden jedoch nicht wie üblich über Tafeln an den jeweiligen Stationen transportiert. An den entsprechenden Orten sind QR-Codes³ angebracht. Diese kann Person A mit ihrem Smartphone einscannen und wird auf eine mobile Webanwendung mit ausführlichen, saisonal abhängigen Informationen zum jeweiligen Standpunkt geleitet.

Da in einem Naturpark keine permanent stabile Netzwerkverbindung gewährleistet werden kann, ist Person A darauf angewiesen, alle benötigten Informationen an einem Ort mit guter Internetverbindung (z.B. bereits zu Hause) auf das Smartphone zu laden. Die Applikation muss deshalb beim ersten Aufruf alle benötigten Ressourcen und Daten auf dem Smartphone speichern. Es ist davon auszugehen, dass während des Abgehens des Lehrpfades lediglich kleinere Updates möglich sind, wenn sich Person A in einem Bereich mit ausreichender Verbindung befindet. Der Großteil der Interaktion mit der Applikation findet ohne Netzwerkverbindung statt.

³ Ein Quick-Response-Code (kurz QR-Code) ist ein zweidimensionaler Code, der zur Verschlüsselung von Informationen (z.B. URLs) dient.

3 Offline-Erweiterung des Frameworks Mobilot

Das Ziel der im Rahmen dieser Arbeit erstellten Erweiterung ist die Umsetzung des angeführten Szenarios (vgl. Abschnitt 2). Als Basis für diese Entwicklung wird das Framework Mobilot in der Version 1.0 genutzt, das auf eine permanente Internetverbindung angewiesen ist.

3.1 Ausgangslage

Mobilot ist ein Framework, welches das Umsetzen von mobilen kontextsensitiven Systemen erleichtert. Es können Anwendungen erstellt werden, die auf die GPS-Position der NutzerInnen, die aktuelle Zeit, einen Code (bspw. QR-Code) und definierbare Regeln reagieren. Mobilot, der mobile Pilot, bietet einen Zugang zu unterschiedlichsten Modulen, die Mobidule. Ein Mobidul besteht aus für Smartphones optimierten Inhalten. Das erweiterbare System eröffnet einen intuitiven Zugang zu diesen Informationen. Dabei wird der Zugriff auf diese Informationen den BenutzerInnen erleichtert. Wird ein QR-Code mit der Kamera am Smartphone eingescannt oder der Anwendung erlaubt, die aktuelle Position zu ermitteln, erhalten die NutzerInnen in Abhängigkeit von Code, Zeit und Position die zur Situation passenden Informationen. Inhalte können dabei Texte, Bilder, Videos, aber auch Karten sein. Im Grunde sind alle beliebigen Webinhalte darstellbar. Außerdem ist die Anbindung an bestehende Content-Lieferanten wie Datenbanken und Content-Management-Systeme möglich (Eitler, Blumenstein & Schmiedl, 2011; Schmiedl, 2011).

Mobilot ist zur Unterstützung in beliebigen mobilen Szenarien verwendbar. Interaktive Städte-, Reise- und Museumsführer sind genauso wie GPS-basierte Spiele oder Location Based Services und Eventinformationssysteme auch für ProgrammierneinsteigerInnen leicht umsetzbar (Eitler u. a., 2011; Schmiedl, 2011). Auch die Umsetzung pädagogisch interessanter Szenarien, bspw. Lehr-Rätselrallyes, Lehrpfade und geobasierter Spiele, ist mit dem System einfach möglich.

Folgende Features sollen integriert werden:

- **Offline-Fähigkeit:** Die Webanwendung soll nach erfolgter Initialisierung ohne Netzwerkverbindung genutzt werden können. Dies gilt auch für den parametrisierten Aufruf der Webanwendung beim Aufruf durch eine externe Applikation oder der Eingabe der URL durch BenutzerInnen. Das Scannen eines QR-Codes mit einer externen Applikation (QR-Code Rea-

der) ist dafür ein typisches Anwendungsszenario. In diesem Fall muss sichergestellt werden, dass beim Aufruf einer Webseite die vorab zwischengespeicherte Kopie genutzt werden kann.

- Lokale Datenhaltung: Sämtliche Daten (Inhaltsdaten und Ressourcen) sollen lokal auf dem mobilen Geräte gespeichert und automatisch synchronisiert werden, sobald eine Datenverbindung zustande kommt.
- Usage-Log: Die Aktionen des Users sollen auch bei fehlender Netzwerkverbindung lokal gespeichert und erst bei erneuter Konnektivität an den Webserver gesendet werden.

3.2 *Verwendete Design Pattern, Methoden und Technologien*

Die Anwendung wird als „modeless application“ implementiert. Sie arbeitet grundsätzlich lokal. Die Prüfung auf eine bestehende Internetverbindung wird nur durchgeführt, wenn diese explizit erforderlich ist. Die BenutzerInnen müssen nicht aktiv eingreifen, um zwischen Online- und Offline-Nutzung zu unterscheiden.

Der gesamte Synchronisationsprozess erfolgt im Hintergrund (Background Sync). Der User muss in den Prozess nicht eingreifen.

Als Delivery Mode wird wie bei Webanwendungen üblich die pull-only Variante umgesetzt. Der Datentransfer vom Server zum Client wird mit einem Pull des Clients gestartet.

Es werden drei Synchronisationsintervalle implementiert: periodisch, bedingte Auslieferung und ad hoc. In einer periodischen Synchronisation werden Daten in regelmäßigen Intervallen von Server zu Client gesendet. Bei der bedingten Auslieferung werden nur Daten vom Server gesendet, wenn bestimmte Bedingungen erfüllt sind. Eine einfache Bedingung kann bspw. das Erreichen eines bestimmten Zeitpunktes sein. In einem pull-basierten System wird meist eine ad-hoc- oder irreguläre Synchronisation implementiert. Die Daten werden ad hoc vom Server gepullt, wann immer die Clients anfragen (Ozsu & Valduriez, 2011, S. 6). In diesem Zusammenhang wird Periodic Refresh für das periodische Synchronisationsintervall umgesetzt. Der Browser sendet in einem bestimmten Intervall einen XMLHttpRequest, um neue Daten vom Server anzufordern (Mahemoff, 2006, S. 215 ff.).

Mithilfe des Cache Manifest⁴ erfolgt das Laden der Ressourcen in den Local Cache of Documents⁵ und in den Local Cache of Data⁶ nach dem Eager Load Pattern. Dabei werden alle Dateien und Daten sofort geladen (van Zyl, Kourie, Coetzee & Boake, 2009, S. 151; Wider, 2007, S. 26). In diesem Zusammenhang wird ebenfalls ein Browser-Side Cache (Mahemoff, 2006, S. 290) umgesetzt.

Realisiert wird außerdem ein Cache Manager nach dem Proxy Design Pattern (Gamma, Helm & Johnson, 2001, S. 254 ff.). Er kapselt den Zugriff auf die Datensätze, in dem die Abfrage der Daten auf den Cache Manager umgeleitet werden. Dieser entscheidet, woher die Datensätze abgerufen werden (lokal oder über Netzwerkverbindung). Bisher wurde diese Abfrage stets per direktem AJAX-Call an den Server realisiert.

Innerhalb dieser Implementation sind keine Updates von Daten der Datenbank vom Client zum Server vorgesehen. Deshalb wird als Replikationsmethode die Master-Slave-Datenbank-Replikation implementiert (IBM, 2001; Qui, 2010, S. 22 f.). Dabei stellt die serverseitige Datenbank den Master und die clientseitigen Kopien die Slaves dar. Änderungen werden nur in der serverseitigen Master-Datenbank durchgeführt und bei der unidirektionalen Synchronisation an alle Clients (Slaves) übergeben. Es ist somit nicht erforderlich, einen Locking Mechanismus (Optimistic Offline Lock und Pessimistic Offline Lock (Fowler, 2003, S. 470)) zu implementieren.

Im Framework Mobilot sowie zur Erstellung der Content-Seiten kommen clientseitig die Sprachen HTML, JavaScript und CSS zum Einsatz. Die Synchronisation der Daten mit dem Server wird mit AJAX umgesetzt. Aus dem Umfeld von HTML 5 werden das Cache Manifest und das damit verbundene Application Cache API⁷ eingesetzt, um die Ressourcen des Local Cache of

4 Ist eine Cache-Manifest-Datei mit dem Attribut `manifest` im `html`-Tag der Startdatei einer Webseite verlinkt, werden alle GET-Anfragen zum lokalen Speicher umgeleitet. In der Cache-Manifest-Datei wird gespeichert, welche Ressourcen für die Webseite benötigt werden. (Hickson, 2012; Kessin, 2011; Spiering & Haiges, 2010)

5 Der Local Cache of Documents beinhaltet alle zur Darstellung der Applikation benötigten Ressourcen (bspw. HTML-, JavaScript-, CSS- und Bilddateien). Dieser Cache muss nur lesbar sein.

6 Im Local Cache of Data werden alle Daten einer Applikation gehalten. Relevant sind hier alle Daten, die zum Betreiben der Anwendung notwendig sind. Der Local Cache of Data muss sowohl gelesen als auch geschrieben werden können.

7 Das Application Cache API dient zum Ansprechen des Offline-Speichers.

Documents auf dem Client zu speichern. Außerdem werden die JavaScript-APIs Browser Status⁸, Web Storage⁹ (im Speziellen Local Storage) und WebSQL Database¹⁰ verwendet. Zudem kommen MD5-Hashes¹¹ zum Einsatz, um Änderungen im Cash Manifest und an der Datenbank zu erkennen.

Serverseitig wird der Webservice in PHP implementiert. Der Datenaustausch zwischen Client und Server erfolgt im JSON-Format. Als Datenbanksprache kommt sowohl client- als auch serverseitig SQL zum Einsatz.

3.3 Umsetzung

Zu Beginn der Umsetzung musste entschieden werden, wie die Daten zu den einzelnen Stationen des Naturlehrpfades aufbereitet werden. Hierzu standen zwei Varianten zur Auswahl:

- *Variante 1:* Jede Station des Naturlehrpfades wird als vollständige, statische HTML-Datei erstellt, welche in der Offline-Variante in den Local Cache of Documents geladen wird.
- *Variante 2:* Alle notwendigen Daten zu den Stationen werden in einer Datenbank gehalten. Es wird nur eine HTML-Datei erstellt, die als Template für die Erstellung der resultierenden Content-Seiten fungiert. Mittels JavaScript wird dieses Template mit den jeweils relevanten Daten aus der Datenbank befüllt.

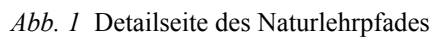
Für diese Implementation wurde Variante 2 gewählt, da bei einer großen Anzahl an Stationen, diese Variante ressourcenschonender ist. Abbildung 1 zeigt eine Detailseite einer Station des Naturlehrpfades.

8 Der Browser-Status kann mithilfe des window-Attributes `navigator.onLine` abgefragt werden.

9 Der Web Storage erlaubt das Speichern von Schlüssel/Wert-Paaren (Key/Value) in einem Storage-Objekt.

10 Die Spezifikation zu Web SQL definiert eine Schnittstelle zu einer clientseitigen SQL-Datenbank. Web SQL wird jedoch nicht mehr durch das W3C (World Wide Web Consortium) weiterentwickelt und wird daher in der finalen Version von HTML 5 keine Berücksichtigung finden (Hickson, 2010).

11 Der Message-Digest-Algorithmus 5 (kurz MD5) erstellt aus einer beliebigen Nachricht einen 128-bit-Hashwert. Dargestellt wird diese Prüfsumme als 32-stellige Hexadezimalzahl (Rivest, 1992).



Das Framework Mobilot auf der Clientseite nach der Umsetzung der Anforderungen

Ressource-Anfragen und das Abspeichern der History erfolgen über den Mobilot-Cache-Manager (1) (`mobicachemanager.js`). Dieser wiederum greift über die Datei `mobidbsql.js` (2) auf die lokale Datenbank (3) zu. Auf Mobidul-Seite wird ebenfalls ein Cache Manager (`mobidul_cachemanager.js`) (4) implementiert. Dieser liest bspw. die Inhaltsdaten über die Datei `mobidul_dbsql.js` (5) aus der lokalen Datenbank (3).

Im Folgenden werden die Schritte beschrieben, die für die Implementation einer Offline-Nutzung notwendig sind:

- *1. Inhaltsdateien mit clientseitigen Sprachen*
Alle Dateien, die lokal am Client lauffähig sein sollen, müssen mithilfe clientseitiger Sprachen (HTML, JavaScript, CSS) realisiert werden. Die Abfrage der Datenbank erfolgt über einen in PHP implementierten Webservice, welcher mittels AJAX angesprochen wird.
- *2. Ressourcen offline verfügbar machen*
Im 2. Schritt werden die Dateien identifiziert, die notwendig sind, um die Offline-Verfügbarkeit der Anwendung zu gewährleisten. Darunter fallen alle HTML-, JavaScript-, CSS-, XML- und Bilddateien.
Um die Ressourcen der Webanwendung ohne Netzwerkverbindung verfügbar zu machen, wird das Cache Manifest verwendet. Grundsätzlich kann diese Datei als Textdatei erstellt werden, muss im HTML-Response aber mit dem Content-Type `text/cache-manifest` ausgeliefert werden. Änderungen an den zwischengespeicherten Dateien werden nur erkannt, wenn sich das Cache Manifest ändert. Um bei Änderung der Inhaltsdaten (bspw. neue oder andere Bilddatei) nicht per Hand die Manifest-Datei ändern zu müssen, wird die Datei dynamisch mithilfe eines PHP-Skriptes erzeugt. Änderungen an bereits bestehenden Files werden dadurch jedoch noch nicht erkannt. Im Skript wird daher aus jeder Datei ein MD5-Hash generiert und die Summe dieser Hash-Werte am Ende der Manifest-Datei als Kommentar eingefügt. Dadurch kann sichergestellt werden, dass bei Änderung einer Datei, das Cache Manifest verändert wird und somit der Browser die Ressourcen neu lädt.
- *3. Lokale Datenbanken anbindung schaffen*
Um die Replikation der vorhandenen Datenbanktabellen auf der Clientseite zu ermöglichen, wird eine lokale Datenbank eingerichtet. Die Umsetzung erfolgt im Mobidul Naturlehrpfad mit dem Web SQL Database API. Das in der Spezifikation von HTML5 vorgeschlagene Indexed

Database API¹² wird derzeit noch nicht auf mobilen Browsern unterstützt.

Für die Anbindung der Datenbank wurde ein Mobilot-Plugin (vgl. (2) in Abb. 2) geschrieben, in dem der Namensraum `dbsql` definiert wurde. In diesem Plugin sind alle Funktionen, die für die Kommunikation mit der Datenbank notwendig sind, enthalten bspw. zum Öffnen der Datenbankverbindung und Erstellen der Tabellen. Innerhalb des Plugins `dbsql` sind außerdem Funktionen definiert, welche die Mobilot-Ressourcen-Abfragen ersetzen, wenn die Daten lokal gehalten werden.

- *4. Cache Manager schreiben*

Die Hauptfunktionalitäten des Mobilot-Cache-Managers sind das Behandeln der Anfragen nach Mobilot-Ressourcen, die zeitgerechte Durchführung des Datenbank-Downloads, die Realisierung des Ressourcen-Downloads sowie der Usage-Log. Zusätzlich werden auch Zugriffe auf Links in Content-Seiten vom Cache-Manager kontrolliert. Sind Seiten im Offline-Fall nicht verfügbar, so werden den BenutzerInnen sinnvolle Fehlermeldungen angezeigt.

4 Fazit

Die Umsetzung der Offline-Funktionalität für das Mobidul Naturlehrpfad hat gezeigt, dass mit den aktuell vorhandenen Möglichkeiten von HTML 5, die gestellten Anforderungen erfüllt werden können. Jedoch treten derzeit (Stand: Mai 2012) beim Arbeiten mit den JavaScript-Schnittstellen, die im Umfeld von HTML 5 entstehen, noch einige Probleme auf:

- *Zukunftsträchtigeres Indexed Database API nicht nutzbar*

Da das Indexed Database API bisher nicht in den relevanten Browsern (Mobile Safari und Android Browser) implementiert ist, musste die lokale Datenbank mit dem Web SQL Database API realisiert werden.

- *Neuladen der Ressourcen im Local Cache of Documents*

Damit Ressourcen neu geladen werden, muss die Cache-Manifest-Datei verändert werden. Das Cache Manifest kann jedoch auch dynamisch erstellt werden. So kann bspw. ein MD5-Hash der integrierten Dateien erzeugt und dieser als Kommentar angehängen werden.

¹² Mithilfe des Indexed Database API können JavaScript-Objekte direkt in eine indizierte Datenbank geschrieben werden (Mehta, Sicking, Graff, Popescu & Orlow, 2011).

- *Immer alle Dateien im Cache Manifest laden*
Wird das Cache Manifest verändert, werden immer alle Dateien vom Browser neu angefordert. Dieses Problem lässt sich programmierseitig nicht umgehen, wenn man mit dem Manifest arbeitet, da es nicht möglich ist, nur einzelne Dateien neu laden zu lassen.
- *Keine GET-Parameter bei per Cache Manifest gecachten Dateien möglich*
Die über eine GET-Request aufgerufene URL muss mit dem in der Manifest-Datei angegebenen Namen übereinstimmen. Dies wird zum Problem, wenn an einen GET-Request per „?“-Parameter angehängt werden. Sind die URLs nicht mit denselben Parameterangaben in der Manifest-Datei hinterlegt, sendet der Browser den Request prinzipiell zum Server, auch wenn eine lokale Kopie der Datei vorliegt. Da die Auswertung der Parameter aber nicht serverseitig, sondern clientseitig per JavaScript erfolgen soll, ist dies nicht erwünscht.
Lösung 1: Statt die zu übergebenden Parameter als gewöhnliche GET-Parameter, also mittels „?“ an die URL zu hängen, werden diese durch ein Hash-Zeichen (#) von der Basis-URL getrennt und per JavaScript mit `window.location.hash` ausgelesen. Da das Hash-Tag i.A. zum direkten Anspringen eines Anchor-Tags innerhalb der aufgerufenen HTML-Seite genutzt wird, führt dieses nicht zum ungewollten serverseitigen Zugriff.
Lösung 2: Der Wert des GET-Parameters kann bei Aufruf der entsprechenden Seite im Web Storage zwischengespeichert und auf der Zielseite wieder ausgelesen werden.
- *Kein Zugriff auf die Namen der gecachten Dateien durch Application Cache API*
Bei dynamisch erstellten Manifest-Dateien muss eine Liste der eingebundenen Dateien serverseitig gespeichert werden. Diese kann anschließend per AJAX separat geladen werden. Die volle Kontrolle würde jedoch nur eine clientseitig erzeugte Liste mit den tatsächlich gecachten Dateien gewährleisten, die das Application Cache API zur Verfügung stellen könnte.
- *Keine Events für Browser Status*
Im Working Draft zu HTML 5 sind zwei Events (`online` und `offline`) definiert, die je nach Zustand der Netzwerkverbindung ausgelöst werden (Hickson, 2012). Diese Events sind jedoch in den verwendeten Browser (Mobile Safari und Android Browser) bisher nicht implementiert. Um

eine Änderung des Browser-Status zu erkennen, muss ein Intervall definiert werden, welches den Status überprüft.

5 Empfehlungen zur Umsetzung einer offline-fähigen Webanwendung

Die folgenden neun Empfehlungen sollen WebentwicklerInnen bei der erfolgreichen Umsetzung einer offline-fähigen Webanwendung unterstützen:

1. *Cache Manifest nutzen*

Das Cache Manifest ermöglicht die Nutzung einer Anwendung ohne Internetverbindung. Dafür müssen Programmdateien, die auf der Client-seite gecacht werden sollen, in einer clientseitig interpretierbaren Sprache geschrieben sein: HTML und JavaScript.

2. *Webservice für Kommunikation mit serverseitiger Datenbank*

Die Kommunikation mit der serverseitigen Datenbank kann über einen Webservice realisiert werden, der per AJAX angesprochen wird.

3. *Kommentar im Cache Manifest dynamisch generieren*

Damit die durch das Cache Manifest gecachten Ressourcen aktualisiert werden, muss sich die Manifest-Datei verändern. Ein dynamisch generierter Kommentar, der bspw. MD5-Hashes der integrierten Dateien enthält, erleichtert diesen Update-Prozess.

4. *Keine GET-Parameter bei gecachten Dateien verwenden*

GET-Parameter können bei gecachten Dateien nicht verwendet werden, da der im Cache Manifest angegebene Dateiname mit dem aufzurufenden Namen übereinstimmen muss. Die Parameter können als Hash-Wert mit „#“ an die URL gehangen oder der Web Storage zum Zwischenspeichern der Parameter verwendet werden.

5. *Intervall für Update des Cache Manifest (für Entwicklungsprozess)*

Vor allem im Entwicklungsprozess hilft das Setzen eines Intervalls zum Updaten des Cache Manifest. Dieses Intervall beinhaltet die regelmäßige Überprüfung des Cache Manifest auf Änderungen und nach erfolgreichem Download den Wechsel zum aktuellen Cache.

6. *Maximales Alter der lokalen Datenbank festlegen*

Um das Synchronisationsintervall der Datenbank zu definieren, sollte das maximale Alter der lokalen Datenbank bestimmt werden. Innerhalb eines bestimmten Intervalls wird anschließend überprüft, ob dieses Alter er-

reicht ist und dementsprechend kontrolliert werden muss, ob Änderungen an der serverseitigen Datenbank erfolgt sind.

7. *Änderungen der Datensätze mit MD5-Hash erkennen*

Bei überschaubaren Datenmengen kann ein MD5-Hash des Resultats der serverseitigen Datenbankabfrage im Local Storage zwischengespeichert werden, um Änderungen an den Datensätzen zu erkennen.

8. *Local Storage zum Zwischenspeichern für den Offline-Fall verwenden*

Wenn keine Netzwerkverbindung besteht, müssen die für den Server relevanten Daten bzw. eingegebene User-Daten im Local Storage zwischengespeichert werden. Sobald wieder eine Verbindung verfügbar ist, sollten die entsprechenden Daten automatisiert an den Server gesendet werden.

9. *Externe (nicht gecachte) Links überprüfen*

Externe (nicht gecachte) Links sollten abgefangen werden, um im Offline-Fall dem User eine Nicht-Verfügbarkeits-Meldung anzeigen zu können.

Außerdem ist bei der Entwicklung von Webanwendungen mit HTML 5 zu beachten, dass dieses noch nicht standardisiert ist und demnach Veränderungen (vor allem in der Implementierung durch die Browser) möglich sind. Im Mai 2011 war der Last Call, der die letzte Möglichkeit darstellte, Änderungswünsche für die Spezifikation einzubringen. Laut W3C ist der Status Recommendation für 2014 geplant (W3C, 2011). Dieser Zeitpunkt ist jedoch nicht unbedingt gleichbedeutend mit einer vollständigen Unterstützung durch die Browser.

Dennoch zeigt diese Arbeit, dass vor allem in mobilen Webanwendungen bereits jetzt mit HTML 5 gearbeitet werden kann.

Referenzen

- Eitler, T., Blumenstein, K. & Schmiedl, G. (2011). Mobilot: Architektur und technische Optionen bei der Entwicklung eines mobilen Informationssystems. *Proceedings of 4. Forum Medientechnik*. Gehalten auf der 4. Forum Medientechnik, St. Pölten, Austria.
- Fowler, M. (2003). *Patterns für Enterprise-Application-Architekturen*. Bonn: Mitp.
- Gamma, E., Helm, R. & Johnson, R. (2001). *Entwurfsmuster . Elemente wiederverwendbarer objektorientierter Software* (2. Aufl.). Addison-Wesley.

- Hickson, I. (2010). Web SQL Database. *W3C*. Abgerufen April 14, 2012, von www.w3.org/TR/webdatabase/
- Hickson, I. (2012). 5.6 Offline Web applications — HTML 5. Abgerufen Mai 17, 2012, von www.w3.org/TR/html5/offline.html#offline
- IBM (2001). *Enterprise replication: a high-performance solution for distributing and sharing information*. San Jose, CA, USA: IBM Corporation. Abgerufen von www.iug.org/library/ids/whitepaper/GC27-1570-00.pdf
- Ipsos & MMA (2012). Our Mobile Planet. Abgerufen Mai 19, 2012, von <http://tinyurl.com/8noy8ab>
- Kaikkonen, A. (2008). Full or tailored mobile web- where and how do people browse on their mobiles? *Proceedings of the International Conference on Mobile Technology, Applications, and Systems – Mobility '08* (S. 1). Yilan, Taiwan.
- Kessin, Z. (2011). *Programing web applications in HTML 5*. Sebastopol CA: O'Reilly Media.
- Mahemoff, M. (2006). *Ajax design patterns* (1st ed.). Sebastopol CA: O'Reilly.
- Mehta, N., Sicking, J., Graff, E., Popescu, A. & Orlow, J. (2011). Indexed Database API. *W3C*. Abgerufen April 14, 2012, von www.w3.org/TR/IndexedDB/
- mobilkom Austria (2006). A1 startet HSDPA-Netz: Verkauf der ersten HSDPA-Datenkarten läuft an. Abgerufen Mai 19, 2012, von www.presstext.com/news/20060120029
- Ozsü, M. T. & Valduriez, P. (2011). *Principles of Distributed Database Systems*. Springer.
- pdwb (o. J.). Bevölkerungsentwicklung Deutschlands. Abgerufen Mai 19, 2012, von www.pdwb.de/kurz_deu.htm
- PwC (2010). pwc.de: Partner im weltweiten Verbund. Abgerufen April 28, 2012, von www.pwc.de/de/unternehmensinformationen/profil.jhtml
- PwC (Hrsg.) (2011). *German Entertainment and Media Outlook 2011–2015: Onlinewerbung übernimmt Pole-Position*. Frankfurt am Main. Abgerufen von www.pwc.de/de/technologie-medien-und-telekommunikation/german-entertainment-media-outlook-2011.jhtml
- PwC & Wilkofsky Gruen Associates (2011). Mobiles Internet: Nutzer in Deutschland bis 2015 | Prognose. Abgerufen April 14, 2012, von <http://ezproxy.fh-stp.ac.at:2078/statistik/daten/studie/180578/umfrage/anzahl-der-nutzer-des-mobilen-internets-in-deutschland-seit-2005/>
- Qui, X. (2010). *A Publish-Subscribe System for Data Replication and Synchronization Among Integrated Person-Centric Information Systems* (Master). Utah State

- University, Logan, Utah. Abgerufen von <http://digitalcommons.usu.edu/cgi/view-content.cgi?article=1616&context=etd>
- Rivest, R. (1992). The MD5 Message-Digest Algorithm. Abgerufen Mai 11, 2012, von <http://tools.ietf.org/html/rfc1321>
- Schmiedl, G. (2011). Mobilot | Fit für Forschung. Abgerufen Mai 5, 2012, von <http://www.fit-fuer-forschung.eu/mobilot>
- Spiering, M. & Haiges, S. (2010). *HTML 5-Apps für iPhone und Android entwickeln*. Poing: Franzis-Verl.
- van Zyl, P., Kourie, D. G., Coetzee, L. & Boake, A. (2009). The influence of optimisations on the performance of an object relational mapping tool, in: *2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists* (S. 150–159). Vynderbijl Park, South Africa: ACM Press.
- W3C (2011). W3C Confirms May 2011 for HTML 5 Last Call, Targets 2014 for HTML 5 Standard. Abgerufen Mai 20, 2012, von www.w3.org/2011/02/htmlwg-pr.html.en
- Wider, A. (2007). *Komponentenorientierte Anwendungsentwicklung auf der .NET-Plattform* (Diplomarbeit). Technische Fachhochschule Berlin, Berlin. Abgerufen von http://metrik.informatik.hu-berlin.de/grk-wiki/images/0/07/Diplomarbeit_Wider.pdf
- WUNDERMANN PXP GmbH (2011). Media Use Index Österreich 2011. Abgerufen von <http://www.media-use-index.at/>