

## OFF - Offline Webapps

### Hinweis:

Diese Druckversion der Lerneinheit stellt aufgrund der Beschaffenheit des Mediums eine im Funktionsumfang stark eingeschränkte Variante des Lernmaterials dar. Um alle Funktionen, insbesondere Animationen und Interaktionen, nutzen zu können, benötigen Sie die On- oder Offlineversion. Die Inhalte sind urheberrechtlich geschützt.  
©2018 Beuth Hochschule für Technik Berlin


## OFF - Offline Webapps



## Überblick und Lernziele

In der vorangegangenen Lerneinheit haben Sie mit IndexedDB eine Technologie zur lokalen Speicherung größerer Mengen strukturierter Daten auf dem Endgerät eines Nutzers kennengelernt. Im Gegensatz zu den Anwendungen der früheren Lerneinheiten, in denen wir CRUD-Operationen auf Datenbeständen durchgeführt haben, benötigte die Beispielanwendung in der Lerneinheit LDS einen Webserver lediglich für die Bereitstellung statischer Bilddateien, die in den Ansichten der Anwendung verwendet wurden, sowie der statischen Programmressourcen in Form von HTML, CSS und JavaScript. Damit wurden die Ansichten der Anwendung aufgebaut und gestaltet sowie die Nutzerinteraktion kontrolliert. Hingegen wurden die bei der Verwendung der Anwendung durch den Nutzer erstellten Inhalte lokal auf dem Endgerät gespeichert.

Den erreichten Entwicklungsstand wird diese Lerneinheit aufnehmen und ihn dahingehend erweitern, dass wir die im Browser ausgeführte Anwendung vollständig unabhängig von der Server-Seite machen. Den Server benötigen wir dann nur noch für die initiale Bereitstellung der statischen Programmressourcen sowie für deren Aktualisierung. Abgesehen davon wird die Anwendung jedoch vollständig offline-fähig sein, d. h. sie wird unabhängig von der Verfügbarkeit des Servers gestartet und bedient werden können. In einem letzten Schritt werden wir den Zugriff auf die Anwendung durch die Nutzer dann von einem explizit zu startenden oder einem laufenden Browser und dessen Bedienelementen entkoppeln. Der Nutzer wird dann die Anwendung wie eine auf dem Endgerät *installierte* Anwendung unabhängig vom UI des Browsers starten und bedienen können.

Im Rahmen unserer Lehrveranstaltung wird die vorliegende Lerneinheit also zwei wesentliche und offensichtliche Lücken schließen, die zwischen den bisher betrachteten Webanwendungen bei einem Zugriff via mobiler Endgeräte und nativen mobilen Applikationen noch bestehen. Die Offline-Fähigkeit von Webapplikationen erlaubt einen Betrieb der Anwendung unabhängig von der Online-Verfügbarkeit des Programmcodes und entspricht damit einem Nutzungsmodell, bei dem die Anwendung über einen Distributionskanal wie einen „App Store“ auf das Endgerät geladen und dann unabhängig vom letzteren verwendet werden kann. Ein erneuter Zugriff auf den Distributionskanal ist dann nur noch für etwaige Aktualisierungen erforderlich. Die Installation von Anwendungen als  **Webapps** auf einem Endgerät schließlich entkoppelt diese Anwendungen auf der Ebene des User Interface von den Bedienelementen eines Browsers und ermöglicht damit ebenfalls eine Nutzung, die dem Nutzungskonzept nativer mobiler Anwendungen entspricht.

Erwähnt sei jedoch, dass diese beiden Aspekte der *Offlinefähigkeit* einer Webanwendung und deren *Installierbarkeit als Webapp* unabhängig zu betrachten sind und auch in der Umsetzung nicht aneinander gekoppelt sind. So müssen einerseits Offline-fähige Webanwendungen nicht als Webapps installierbar sein und müssen Webapps andererseits nicht offline-fähig sein. Mit Blick auf diverse Unzulänglichkeiten, die in Bezug auf Webapps noch zu beobachten sind, wird das Übungsprogramm den auch die Offline-Fähigkeit der Anwendungen in den Mittelpunkt stellen.

Im folgenden werden Sie zunächst die im Rahmen der HTML-Spezifikation erarbeiteten Ausdrucksmittel für Offline-fähige Webanwendungen kennenlernen. Danach werden wir zeigen, wie Webanwendungen als Webapps für Firefox installiert werden können.



Lernziele

### Lernziele

Nachdem Sie die Lerneinheit durchgearbeitet haben, sollten Sie in der Lage sein:

- Zu erläutern für welche Inhalte die Verwendung des Offline Caches geeignet sind und welche Anpassungsmöglichkeiten es gibt.
- Die Verwendung von Cache-Manifest-Dateien zu erklären und anzuwenden.
- Den Begriff Offline WebApp hinsichtlich seines Bedeutungsspektrums zu erklären.
- Eine von Ihnen erstellte Anwendung als OWA bereitzustellen.



Gliederung

**Gliederung**

- Offline Webanwendungen
- Webapps
- Zusammenfassung
- Prüfungsfragen
- Übungen



Zeitbedarf

**Zeitbedarf und Umfang**

Für die Bearbeitung der Lerneinheit benötigen Sie etwa 3 Stunden. Für die Bearbeitung der Übungen der Beispielanwendung etwa 2.5 Stunden und für die Fragen zur Wissensüberprüfung ca. 1.5 Stunden.

## 1 Offline Webanwendungen

Unter dem Titel *Offline Web Applications* findet sich eine Spezifikation der nachfolgend vorgestellten Funktionalität sowohl in der [HTML5 Spezifikation des W3C](#) als auch im entsprechenden [HTML „Living Standard“ Spezifikationsdokument der WHATWG](#). Vorgesehen wird dafür die Hinzufügung einer *Cache Manifest Datei* zu einer Webanwendung. Mittels dieses Manifests können Aussagen zur Offline Verfügbarkeit oder Nichtverfügbarkeit der von der Anwendung verwendeten Ressourcen getroffen werden. Dies betrifft alle Ressourcen, die von einer Anwendung verwendet werden, d. h. HTML Dokumente und die daraus referenzierten Skripte, Styles und Medienressourcen sowie Ressourcen, auf die mittels XMLHttpRequest zugegriffen wird.

Die Spezifikation legt zum einen die Syntax für die Inhalte der Cache Manifest Datei fest und schreibt zum anderen die Art und Weise vor, wie das Cache durch den Browser aufgebaut und wie dessen Inhalte zur Laufzeit behandelt werden sollen. Darüber hinaus definiert sie mit dem [Interface ApplicationCache](#) eine API, mittels derer eine Webanwendung u. a. Feedback über die Verwendung des Offline Cache durch den Browser erhalten kann. Die Nutzung der gecachten Inhalte zur Laufzeit bleibt jedoch vollkommen vor dem Anwendungsentwickler verborgen.

So wird denn auch ein direkter Zugriff inklusive einer Manipulation einzelner Ressource des Offline Cache auf Ebene der API nicht eingeräumt. In dieser Hinsicht entspricht die Behandlung des Offline Cache dem Umgang mit anderen durch den Browser gecachten Inhalten, die entsprechend der Lerneinheit LDS erwähnten Caching Verfahren gehandhabt werden, welche für das HTTP-Protokoll vorgesehen sind. Auch auf deren Verwendung haben Sie als Anwendungsentwickler nur diejenigen Einflussmöglichkeiten, die Ihnen durch die Regeln für Caching-bezogene Header von Request und Response gegeben sind. Einen Zugriff auf gecachte Inhalte über die JavaScript API ist jedoch auch dort nicht möglich.

Wie andere Zugriffe, die aus dem Browser heraus bezüglich lokaler Ressourcen stattfinden, wird auch die Nutzung des Dateisystems für die Speicherung des Offline Cache durch den Browser für die gegebene Ursprungsdomain rückbestätigt, wie Sie in folgender Abbildung sehen.

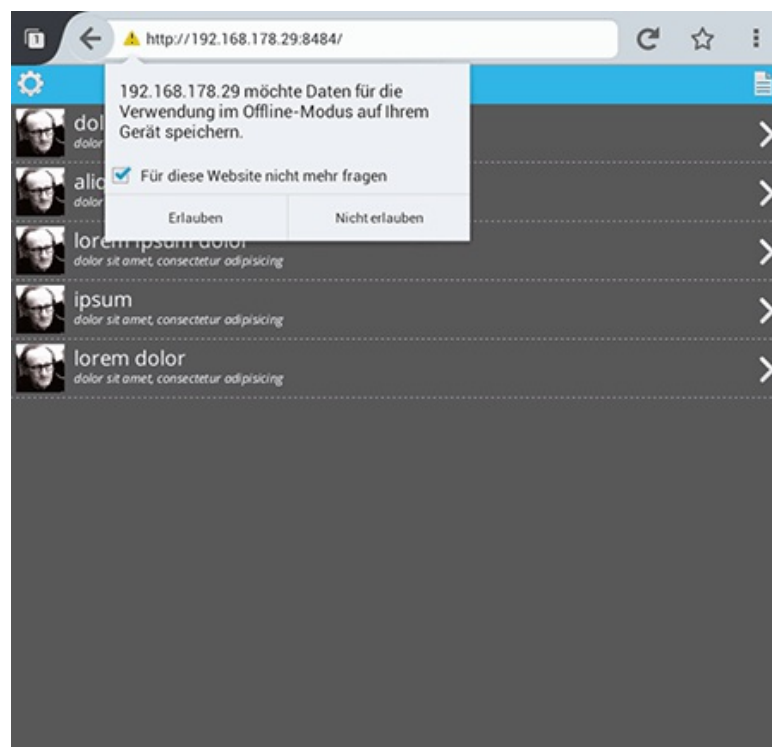


Abb.: Rückbestätigung der Offline Cache Nutzung durch Firefox auf Android

Wir stellen zunächst die Konzepte für das Cache Manifest und den Aufbau einer Manifestdatei vor und gehen dann in Kürze auf die `ApplicationCache` API ein.

## 1.1 Cache Manifest

Aufgabe der Manifestdatei ist es festzulegen:

- Welche Ressourcen einer Webanwendung offline verfügbar sein sollen (*Offline Ressourcen*).
- Auf welche Ressourcen grundsätzlich online zugegriffen werden sollen (*Online Ressourcen*).
- Welche „Fallback“-Lösungen für den Fall der Nichtverfügbarkeit einer Ressource anzuwenden sind.

Zu diesem Zweck muss jeder von einer Anwendung genutzten Ressource einer der hier genannten Ressourcenstatus zugeordnet werden. Die Auflistung der offline verfügbaren Ressourcen ist dabei erschöpfend und explizit, d. h. sie kann nicht pauschal für alle Ressourcen vorgenommen werden, deren URL einem bestimmten Muster entspricht. Letzteres ist jedoch für die Deklaration der online zuzugreifenden Ressourcen möglich.

Wie wir im folgenden Unterabschnitt sehen werden, liegt der Grund für diese verschiedene Behandlung darin, dass bei der Erstellung eines Offline Cache alle deklarierten Offline-Ressourcen auf einmal vom Browser geladen werden. Dafür muss jedoch auch jede einzelne Ressource, die geladen werden soll, bekannt sein. Im Gegensatz dazu wird die Deklaration der Online-Ressourcen nur Schritt für Schritt zur Laufzeit einer Anwendung berücksichtigt, sobald ein Zugriff auf eine Ressource erfolgt.

### Aufbau

Erstellt wird die Cache Manifest Datei als Textdatei, die mit dem Text `CACHE MANIFEST` beginnen muss. Die verschiedenen Status von Ressourcen in Bezug auf die Offline-Fähigkeit, die wir oben erwähnt haben, werden durch definierte Keywords gefolgt von einem Doppelpunkt – ':' – markiert. Vorgesehen sind die Keywords `CACHE`, `NETWORK` sowie `FALLBACK` für Offline und Online Ressourcen bzw. für Fallbacks bezüglich der letzteren.

Beispiel für den Aufbau einer Cache Manifest Datei (*Ausschnitt aus dem Manifest der Beispielanwendung*)

```
001 CACHE MANIFEST
002
003 # we cache all views, styles, and scripts
004 index.html
005
006 # js
007 js/off.js
008 js/mme2_xmlhttp.js
009 js/mme2_webapp.js
010 # (...)
011
012 # style and style imgs
013 css/mwF.css
014 css/img/save.png
015 css/img/ok.png
016 # (...)
017
018 NETWORK:
019 http2mdb/*
020 available
```

#-Suffix einer URL ist nicht zulässig.

Wie Sie in Beispiel oben sehen, werden nach dem Keyword die URLs der jeweiligen Ressourcen zeilenweise aufgelistet, wobei die Referenzierung von Dokument-Fragmenten – gekennzeichnet durch ein '#'-Suffix einer URL – nicht zulässig ist. Das '#' kann jedoch zur Auskommentierung von Inhalten der Manifest-Datei verwendet werden. Ressourcen, die unmittelbar nach der `CACHE MANIFEST` „Überschrift“ deklariert werden, wird der Status `OFFLINE` zugeordnet. Beachten Sie außerdem, dass für jeden der genannten Ressourcenstatus mehr als ein „Block“ von Ressourcen deklariert werden kann, d. h. jedes Keyword kann ggf. mehrfach in einer Manifestdatei auftauchen und bezeichnet dann den Status der nachfolgend genannten Ressourcen, ggf. bis zu einem nachfolgenden Keyword oder zum Dateiende. Nachfolgend finden Sie Einzelheiten, die bezüglich der einzelnen Status von Ressourcen wissenswert sind.

## CACHE

Wie eingangs erwähnt, werden dem `CACHE` Status alle Ressourcen zugeordnet, die Offline verfügbar sein sollen. Hier wie für die anderen Status kann für die Identifikation einer Ressource eine relative URLs verwendet werden, die gegenüber der URL der Manifest Datei aufgelöst wird. Absolute URLs unterliegen nicht der Same Origin Policy, da beim Caching der Ressourcen zur Laufzeit nur deren Inhalte ins Cache übertragen werden, aber keine Ausführung der Ressource erfolgt.

Ob eine Ressource tatsächlich verwendet werden kann oder ob ihre Ausführung aufgrund geltender Restriktionen unterbunden wird, wird dann beim tatsächlichen Zugriff auf die Ressource zur Laufzeit entschieden, z. B. beim Laden eines Skripts aus einem HTML Dokument heraus oder bei der Ausführung eines `XMLHttpRequest`. Wie vorstehend erwähnt, ist der `CACHE` Status insofern der Default-Status bezüglich eines Manifests als er für Ressourcen, die unmittelbar am Beginn eines Dokuments aufgelistet werden, nicht explizit deklariert zu werden braucht.

## NETWORK

Dem `NETWORK` Status werden diejenigen Ressourcen zugeordnet, die grundsätzlich online zugegriffen werden sollen. Da hier für die Formulierung von URLs Wildcards verwendet werden können – `*` – ist eine explizite Auflistung aller `NETWORK` Ressourcen nicht erforderlich.

**Wildcards**

Probleme bereiteten Wildcards bei der Ausführung in Safari. Hierfür reichte es aber, das Wildcard-Symbol zu entfernen. Danach wurden alle Zugriffe auf Ressourcen, deren URLs mit dem angegebenen URL Segment – `http2mdb` – beginnen, als Netzwerkzugriffe ausgeführt.

Zu beachten ist jedoch, dass beim Zugriff auf Online Ressourcen, die nicht als solche via `NETWORK` im Manifest deklariert werden, Fehler auftreten können. So wurde in Firefox für die Ausführung von `XMLHttpRequest` bezüglich solcher Ressourcen der Status Code 0 ermittelt, obwohl die Ressource über eine Netzwerkverbindung hätte zugegriffen werden können. Wird eine Ressource aber als Online Ressource deklariert, sind durch den Anwendungsentwickler geeignete Vorkehrungen zu treffen, falls die Ressource tatsächlich nicht verfügbar ist – sei es aufgrund Nichtverfügbarkeit eines Netzwerks oder aus anderen Gründen.

## FALLBACK

Ein einfache Möglichkeit, auf Ebene des Cache Manifest Vorkehrung für die Nichtverfügbarkeit einer Online Ressource zu treffen, ist die Deklaration einer `FALLBACK` Regel für die betreffende Ressource. Solche Regeln werden zeilenweise als Paare der Form `URL1 URL2` notiert, wobei `URL1` die betreffende Ressource ist und `URL2` eine Ersatzressource für diese identifiziert. Für `URL1` ist hier die Verwendung von Wildcards erlaubt, d. h. eine einzelne Regel kann auf eine Menge von Ressourcen angewendet werden, deren URLs dem betreffenden Muster entsprechen. Damit eine Fallback-Regel auf eine oder mehrere Ressourcen angewendet werden kann, ist es nicht erforderlich, dass die Ressourcen zusätzlich als `NETWORK` Ressourcen deklariert werden.

## Verwendung und Referenzierung

Mit den genannten Ausdrucksmitteln könnte ein Cache Manifest also als eine erschöpfende Deklaration aller von einer Anwendung verwendeten Ressourcen und deren jeweiliger Status in Bezug auf ihre Offline-Verfügbarkeit angesehen werden. Wie aber kann eine Anwendung überhaupt zum Ausdruck bringen, dass sie ein Offline Cache bereitstellt? Dafür bedient sich die Spezifikation eines der fundamentalsten Ausdrucksmittel von HTML – nämlich der Möglichkeit, aus einem HTML-Dokument heraus andere Ressourcen zu referenzieren – und sieht die Verwendung eines `manifest` Attributs vor, das auf dem Wurzelement eines HTML-Dokuments gesetzt wird:



Quellcode

**Cache Manifest**

```
001 <!DOCTYPE html>
002 <!-- Deklaration eines Cache Manifest -->
003 <html manifest="iamwebapp.manifest">
004 <!-- (...) -->
005 </html>
```

Eine solche Attributsetzung veranlasst den Browser dazu, das betreffende Manifest vom Server zu laden und dessen Inhalte entsprechend der oben dargestellten Semantik und entsprechend dem nachfolgend gezeigten Verfahren zu behandeln. Dabei wird die nachvollziehbare Annahme getroffen, dass ein HTML-Dokument, das eine Referenz auf ein Offline Manifest verwendet, selbst offline verfügbar sein soll. Solche Dokumente brauchen nicht ihrerseits im Offline Manifest aufgelistet zu werden – die Spezifikation bezeichnet sie denn auch als „Master Entries“ unter den im Cache enthaltenen Inhalten. Eine Anwendung kann dabei auch grundsätzlich mehrere solcher Master Entries ggf. bezüglich verschiedener Manifest Deklarationen enthalten.

Inwiefern eine solche „verteilte Lösung“ mit Blick auf die Handhabbarkeit einer Anwendung aber empfehlenswert ist, muss jeweils unter Berücksichtigung der Komplexität einer Anwendung und der daraus ggf. resultierenden Notwendigkeit einer Modularisierung entschieden werden.

Content-Type und  
Referenzierung

Als Endung eines Cache Manifest wird das Suffix `.manifest` empfohlen, und ausgeliefert werden müssen deren Inhalte vom Server mit dem Wert `text/cache-manifest` als Content-Type. Wie für andere Unzulänglichkeiten einer Anwendung gemessen an den Anforderungen der HTML-Spezifikation gilt aber auch hier, dass Abweichungen davon durchaus liberal von Browsern gehandhabt werden können, eine solche Behandlung jedoch nicht garantiert ist. So ist es denn zur Behandlung einer Cache Manifests als Manifest wesentlich, dass dieses aus einem HTML-Dokument heraus über das `manifest` Attribut referenziert wird.

Zu beachten ist außerdem, dass wir hier zwar meistens von „Manifest Dateien“ sprechen und solche auch in den Implementierungsbeispielen verwenden, dass es sich hier aus Sicht des Browsers jedoch um eine Ressource handelt, deren Inhalte nicht notwendigerweise als Datei im Dateisystem des Servers vorliegen müssen, sondern alternativ auch dynamisch generiert werden können. Auf diese Weise könnten z. B. dynamisch durch die Nutzer einer Anwendung erstellte Inhalte, die nicht zu den „initialen Programmressourcen“ der Anwendung gehören, gleichermaßen über ein Cache Manifest als Offline Ressourcen deklariert werden.


## 1.2 Verwendung des Manifests

Verwendet ein HTML-Dokument ein `manifest` Attribut, dann erfolgt der Zugriff auf die darin referenzierte Ressource und die Verarbeitung von deren Inhalten nach einem durch die Spezifikation für Offline Webanwendungen exakt beschriebenen Verfahren, das die folgenden Schritte vorsieht:

1. **Zugriff** auf das referenzierte Cache Manifest. Falls nicht zugreifbar, wird die Anwendung offline ausgeführt.
2. **Überprüfung**, ob das Cache Manifest seit dem letzten Zugriff modifiziert wurde – dafür reicht die Änderung eines Kommentars! Falls nicht modifiziert, bleibt das browserseitige Cache unverändert.
3. Ggf. **Laden** aller im Cache Manifest referenzierter Ressourcen.
4. Nach erfolgreichem Laden ggf. **Ersetzung** eines bereits existierenden Cache.
5. **Verwendung** der Ressourcen aus dem Cache.

Dieser Vorgang erfolgt asynchron, d. h. der Browser stellt das Dokument, das das `manifest` Attribut verwendet, unabhängig vom Laden der Cache Inhalte dar und kann Interaktionsereignisse des Nutzers behandeln. Erst nach vollständigem Laden aller im Cache referenzierter Offline Ressourcen werden diese für den Fall, dass sie im Laufe der Anwendungsnutzung benötigt werden, tatsächlich aus dem Cache geladen.

ApplicationCache API

Eine Anwendung kann auf die Durchführung von Teilschritten des vorstehend beschriebenen Ladevorgangs sowie auf Ereignisse reagieren, die bei deren Ausführung auftreten können – z. B. die Nichtverfügbarkeit einer im Cache referenzierten Offline Ressource. Zu diesem Zweck steht Ihnen ein Objekt zur Verfügung, das das im Standard spezifizierte  `ApplicationCache` Interface implementiert und das über das Attribut `applicationCache` aus dem globalen `window` Objekt ausgelesen werden kann. Die Reaktion auf Ereignisse kann dann durch geeignete Event Handler auf dem `applicationCache` Objekt veranlasst werden.



Für eine Übersicht über alle verfügbaren Ausdrucksmittel von `applicationCache` verweisen wir auf die [www Spezifikation](#), Einblick in die [JavaScript API](#) gibt Ihnen im Rahmen einführender Hinweise zur Verwendung des Offline Cache auch ein [www einschlägiges Tutorial](#). Erwähnt sei hier lediglich noch, dass durch Aufruf der `update()` Funktion eine ggf. erforderliche Aktualisierung des Cache auch manuell initiiert werden kann.

#### Hinweise zur Verwendung

Die Verwendung des Offline Cache ist aus Entwicklersicht nicht immer frei von Irritationen. Zu deren Vermeidung oder rechtzeitigen Behebung sei insbesondere erwähnt, dass entsprechend dem vorstehend beschriebenen Ladevorgang eine serverseitig aktualisierte Ressource, die durch Zuordnung zum CACHE Status als Offline Ressource deklariert ist, erst dann vom Browser geladen wird, wenn auch eine Aktualisierung der Manifestdatei selbst vorliegt. Um diesen Fall zu detektieren, versucht der Browser auch beim Laden eines Dokuments aus dem Cache auf dessen ggf. deklariertes Manifest zuzugreifen. Falls eine Änderung von dessen Inhalten – betreffen sie auch nur die Modifikation eines Kommentars – festgestellt werden kann, wird das gesamte Cache aktualisiert und werden alle Ressourcen neu geladen.

Beachten Sie aber, dass diese Aktualisierung asynchron zur Darstellung des Dokuments erfolgt. Falls von Ihnen also dieses Dokument selbst oder eine der daraus referenzierten Ressourcen modifiziert wurde, stehen diese Änderungen auch bei erfolgreicher Aktualisierung des Cache noch nicht unmittelbar, sondern erst bei einem wiederholten Zugriff auf das betreffende Dokument zur Verfügung. Um beispielsweise eine Änderung in einer JavaScript Funktion testen zu können, die im initialen HTML-Dokument Ihrer Anwendung verwendet wird, müssen Sie also zunächst die Manifestdatei modifizieren, dann auf das Dokument zugreifen und schließlich nach Abschluss der dann erfolgenden Cache Aktualisierung mittels eines `refresh` einen erneuten Zugriff auf das Dokument veranlassen. In diesem Fall sollte Ihr aktualisiertes Skript aus dem Cache geladen werden.

Ein weiterer Grund möglicher Irritationen sind fehlerhafte URLs im Cache Manifest. Diese führen dazu, dass das Laden des Cache abgebrochen und etwaige bereits geladene Inhalte vom Browser verworfen werden – für den Fall, dass diese Problematik beim erstmaligen Zugriff auf eine Anwendung auftritt, wird für diese Anwendung also gar kein Cache erstellt, d. h. die Anwendung ist nicht im Offline Modus verfügbar. Um Transparenz bezüglich der im Rahmen einer Cache-Aktualisierung zugegriffenen Ressourcen zu erlangen, erscheint zur Entwicklungszeit daher u. a. die Verwendung von Logmeldungen auf Serverseite als probates Mittel.

Alternativ oder zusätzlich können Sie natürlich auch auf Ebene der clientseitigen Anwendung durch Event Handler bezüglich `applicationCache` und Logmeldungen bzw. Statusmeldungen im User Interface eine freudvolle Verwendung der Offline Cache Funktionalität bereits zur Entwicklungszeit ermöglichen.



### 1.3 Anwendungsbereich des Cache Manifest

Als vollständige Deklaration aller von einer Anwendung zu verwendenden Ressourcen – abzüglich etwaiger nicht aufgelisteter Master-Ressourcen – ist das Cache Manifest nicht in natürlicher Weise dafür geeignet, dynamische Inhalte, die remote bereitgestellt werden, als Offline-Inhalte zu deklarieren. Zwar müssen URLs für Offline-Ressourcen nicht notwendigerweise auf statische Inhalte verweisen, sondern können entsprechend unserer Überlegungen zu RESTful APIs - in Lerneinheit NJM Abschnitt 4 - auch Inhalte einer Datenbank referenzieren. Beachten Sie aber, dass für die Offline-Verfügbarkeit solcher Inhalte eine erschöpfende und explizite Auflistung unter Angabe der exakten absoluten oder relativen URLs notwendig ist, d. h. ggf. auch unter Einbezug von Query-Parametern, die Kontextinformation für den Zugriff übermitteln.

Grundsätzlich ist dafür zwar die dynamische Generierung der Manifest-Inhalte bei Zugriff auf die in einem `manifest` Attribut referenzierte Ressource möglich. Ziehen Sie aber auch in Betracht, dass bei einer Aktualisierung des Manifests eine vollständige Aktualisierung aller Ressourcen im Cache durch den Browser veranlasst wird.

Werden wiederholt Aktualisierungen einer Teilmenge der dynamischen Datenbestände vorgenommen, führt dies nicht nur zu einem erhöhten Datenverkehr zwischen Client und Server und zur Übertragung redundanter, weil unveränderter, Inhalte, sondern bei Zugriff auf Ressourcen in Datenspeichern ggf. auch zu einer erhöhten Bearbeitungslast auf Serverseite. Die technisch denkbare Nutzung des Offline Cache für dynamische Inhalte erscheint daher nicht notwendigerweise auch praktikabel.

Mit Blick auf die Funktionalität der Anwendungen würde eine Lösung wie die vorstehend beschriebene denn auch nur einen „read only“ Zugriffsmodus auf dynamische Inhalte im Offlinebetrieb ermöglichen. Wird andererseits zur Erbringung einer weitgehend unbeschränkten CRUD-Funktionalität unter Offline-Bedingungen ein clientseitiger Datenspeicher verwendet, dann kann dieser auch als Datenquelle für den lesenden Zugriff fungieren, d. h. eine Berücksichtigung dynamischer Inhalte im Offline Cache ist dann nicht mehr erforderlich. Vielmehr können jegliche Zugriffe auf diese Inhalte, wie wir es in der Beispielanwendung tun, mittels geeigneter `NETWORK`-Deklarationen abgedeckt werden:

```
NETWORK:
http2mdb/ *
```

Auf Clientseite ist dann nur noch die Verwendung einer geeigneten Fallunterscheidung erforderlich, die im Offlinebetrieb nicht via `XMLHttpRequest` auf eine serverseitige Anwendungskomponente zuzugreifen versucht, sondern direkt auf einen geeigneten lokalen Datenspeicher zugreift. Die Grundzüge einer Lösung hierfür finden Sie in den Implementierungsbeispielen dieser Lerneinheit.

Angeichts der vorgestellten Ausdrucksmittel für Offline Webapplikationen und der beobachtbaren Schwierigkeiten bezüglich der Behandlung dynamischer Inhalte erscheint eine Anwendung mit *serverseitiger* Markupgenerierung nur sehr bedingt für die Umstellung auf eine Offline-Anwendung geeignet. So müssten wir ja auch sämtliche dynamisch generierten Ansichten solcher Anwendungen mittels der jeweils verwendeten URLs im Cache Manifest deklarieren und würden hier auf dieselben Probleme stoßen, die wir gerade mit Blick auf die Durchführung von CRUD-Operationen beschrieben haben.

Eine entsprechende Ausweichmöglichkeit für den Offline-Betrieb müsste hier jedoch den clientseitigen Aufbau von Ansichten als Alternative zum Laden serverseitig generierter Ansichten vorsehen. Eine solche Duplikation von Funktionalität erscheint im Gegensatz zur Duplikation der CRUD-Operationen für lokale vs. remote Datenspeicher aber weder sinnvoll, noch erforderlich.

Verwendet eine Anwendung von vornherein eine Architektur, bei der jeglicher Aufbau von Ansichten clientseitig erfolgt und die serverseitige Anwendung lediglich die für die Anwendung erforderlichen statischen Programmressourcen sowie ggf. APIs für den CRUD-Datenzugriff bereitstellt, dann ist diese nämlich in sehr natürlicher Weise für die Unterstützung eines Offline-Betriebsmodus geeignet. Dies belegen denn auch die vorliegenden Implementierungsbeispiele, die auf den Beispielen aus der Lerneinheit LDS aufbauen.

Unterstützt wird die hier formulierte Sichtweise nicht zuletzt auch durch das W3C selbst, mit dessen O-Ton wir die Behandlung von Offline Anwendungen an dieser Stelle abschließen wollen:



Definition

#### Application Cache

*“The application cache feature works best if the application logic is separate from the application and user data, with the logic (markup, scripts, style sheets, images, etc) listed in the manifest and stored in the application cache, with a finite number of static HTML pages for the application, and with the application and user data stored in Web Storage or a client-side Indexed Database, updated dynamically using Web Sockets, XMLHttpRequest, server-sent events, or some other similar mechanism.”*

Quelle: <http://www.w3.org>, 5.7.1.1 Supporting offline caching for legacy applications

Es sei erwähnt, dass die Überlegung, das Lehrprogramm der vorliegenden Veranstaltung konsequent auf die Entwicklung Offline-fähiger Webanwendungen hinzuführen, sich durch diese Aussage im nachhinein bestätigt sieht.

Hingewiesen sei jedoch auch darauf, dass damit noch keine Musterlösung für die Behandlung dynamischen nutzergenerierten Contents in Form von Multimediainhalten vorliegt, da für deren lokale Speicherung die Nutzung von IndexedDB, wie in der Lerneinheit LDS dargelegt, suboptimal erscheint. Für ihre Offline-Verfügbarkeit scheint vielmehr der Einsatz von Caching gemäß der HTTP-Spezifikation, d. h. der Rückgriff auf ein sehr traditionelles Verfahrens im Umfeld von Webtechnologien nach wie vor viel versprechend.

## 2 Webapps

Im Umfeld mobiler Web-Technologien gehört der Begriff der „Webapp“ zu den recht zahlreichen Begriffen, für die keine exakte Definition existiert, sondern eine recht große Bandbreite an Auslegungen gebräuchlich ist. So wird darunter wahlweise u. a. folgendes verstanden:

- Web-Applikationen allgemein  
(Siehe z. B. [www http://en.wikipedia.org/wiki/Web\\_application](http://en.wikipedia.org/wiki/Web_application))
- Web-Applikationen, die HTML5 Ausdrucksmittel verwenden  
(Siehe [www http://www.hongkiat.com/blog/html5-web-applications/](http://www.hongkiat.com/blog/html5-web-applications/))
- Web-Applikationen, die für mobile Endgeräte optimiert sind  
(Siehe [www http://apps.ft.com/ftwebapp/index.html](http://apps.ft.com/ftwebapp/index.html) oder [www http://www.html-seminar.de/web-apps-erstellen.htm](http://www.html-seminar.de/web-apps-erstellen.htm))

Wir selbst verwenden den Begriff im folgenden zunächst im Sinne der unter verlinkten Spezifikation von Mozilla Open Web Apps (OWA), werden abschließend jedoch auch eine partielle Ausweitung auf den Gebrauch im Rahmen von iOS vornehmen.

[www https://developer.mozilla.org/en-US/Apps/Quickstart/Build/Intro\\_to\\_open\\_web\\_apps](https://developer.mozilla.org/en-US/Apps/Quickstart/Build/Intro_to_open_web_apps)

### 2.1 Open Web Apps (OWA)

Mit Open Web Apps unternimmt Mozilla den Versuch, das App-Paradigma, das sich im Bereich nativer Anwendungen für mobile Endgeräte seit Verfügbarkeit von Apps für iOS etabliert hat, auf den Bereich von Web-Anwendungen zu übertragen. Dies umfasst u. a. die folgenden Aspekte:

- Anwendungen werden durch spezifische Distributionskanäle – „App Stores“ – in unterschiedlichen *Zertifizierungsstufen* bereitgestellt.
- Anwendungen werden durch den Nutzer *auf einem Endgerät installiert* und können von dort *deinstalliert* werden.
- Im Rahmen der Installation werden durch den Nutzer *Permissions bezüglich des Zugriffs auf lokale Ressourcen* und Funktionen des Endgeräts gewährt.
- Die Anwendung wird durch Zugriff des Nutzers auf eine *ausführbare Datei* bzw. auf eine Verknüpfung diesbezüglich gestartet.
- Die Anwendung wird in einem eigenen Bildschirmbereich („Fenster“) und *ohne generische Browser-Bedienelemente* gestartet.


OWA und Firefox OS


OWAs versuchen alle diese Aspekte zu adressieren, bleiben dabei aber „echte“ Web-Applikationen insofern, als für ihre Ausführung ein auf dem Endgerät installierter Firefox Browser verwendet wird, der jedoch nicht als solcher erkennbar ist. OWAs sind damit insbesondere im Zusammenhang mit dem Versuch von Mozilla zu sehen, mit [www Firefox OS](#) eine eigene Plattform für mobile Anwendungen zu etablieren.

#### Plattform für mobile Anwendungen

Siehe für den Versuch einer Definition dieses gleichermaßen unscharfen Begriffs die Ausführungen im Veranstaltungsskript zu *Mobile Application Development*.

Sie weisen hinsichtlich der verwendeten Ausdrucksmittel zwar übereinstimmende Ideen zu Standardvorschlägen wie den vom W3C vorgeschlagenen [www Packaged Web Apps](#) bzw. Widgets auf, können aber nicht als ein offizieller oder de-facto Standard angesehen werden. So werden wir denn auch sehen, dass zumindest das „Look&Feel“ einer Webapp gemäß OWA als eigenständig laufende Anwendung für iOS mit deutlich einfacheren Ausdrucksmitteln erzielt werden kann.

Der Einsatz von OWAs im Rahmen von Firefox OS geht jedoch deutlich darüber hinaus, u. a. wird hier mit dem  Firefox Marketplace ein *Distributionsmechanismus für OWAs* zur Verfügung gestellt – analog zu den entsprechenden Angeboten für iOS und Android. Durch Deklaration von Permissions können OWAs für Firefox OS auch unter bestimmten Bedingungen und bei Installation über den Marketplace einen weitgehenden und pauschal gewährten Zugriff auf Gerätefunktionen nutzen.

Wenn Sie an unsere Ausführungen in Lerneinheit LDS zum Installationsvorgang als „Gewährung von Vertrauen“ zurückdenken, werden für OWAs damit durch Firefox OS gerade jene Funktionen bereitgestellt, die „gewöhnliche“ Webanwendungen als beim Zugriff über den Browser unmittelbar ausführbare Anwendungen nicht aufweisen. So wird Ihnen in letzteren für bestimmte Funktionen, z. B. beim Zugriff auf Eingabevorrichtungen wie Mikrophon oder Kamera, bei jedem Zugriff eine Rückbestätigung abverlangt und das Aussprechen einer pauschalen Erlaubnis gar nicht ermöglicht. Ausgehend von dieser Grundlinie als „unterster Stufe“ werden für OWAs verschiedene Zertifizierungsstufen definiert, die mit unterschiedlichen Berechtigungen einhergehen und unterschiedliche Prüfungsmaßnahmen bezüglich einer über den Marketplace installierten Anwendung durch dessen Betreiber erfordern. Details hierzu finden Sie  bei Mozilla.

Im Rahmen des Lehrprogramms unserer Veranstaltung werden wir uns bis auf weiteres nicht mit Firefox OS beschäftigen. OWAs sind für uns jedoch insofern von Interesse, als sie auf der untersten Zertifizierungsstufe weder Firefox OS, noch den Marketplace erfordern, sondern eine Installation von jedem beliebigen Webserver erlauben. Um OWAs zu installieren und auszuführen, reicht dann ein gewöhnlicher Firefox Browser aus, der auf dem betreffenden Endgerät ausgeführt wird. OWAs können denn auch auf im Vergleich zu Smartphones und Tablets mehr oder weniger „stationären“ Endgeräten wie einem Windows PC oder einem MacBook installiert und ausgeführt werden.

Bezüglich des Zugriffs auf Gerätefunktionen sind sie jedoch auf den Funktionsumfang eingeschränkt, der durch den Firefox Browser im Rahmen der Unterstützung des aktuellen HTML-Standards bereitgestellt wird, z. B. *Geolocations*, *Media Capture*, *local Storage*, *IndexedDB*, etc. Bedenken Sie jedoch auch, dass Ihnen Firefox für Android, wie in Lerneinheit MME beschrieben, mittels des `<input>` Elements auch einen direkten Zugriff auf die Kamera ermöglicht und dass auch andere native Funktionen wie das Auslösen eines Telefonanrufs gleichermaßen aus HTML-Dokumenten heraus initiiert werden können.

#### Telefon-URL

Hierfür kann ein `<a>` Element bezüglich einer „Telefon-URL“ verwendet werden, z. B.

```
<a href="tel:2125551212">2125551212</a>
```

Nachfolgend betrachten wir die grundlegenden Ausdrucksmittel, die erforderlich sind, um Web-Applikationen als OWAs auf einem Endgerät mit Firefox zu installieren.

## 2.2 Ausdrucksmittel für Open Web Apps

Voraussetzung für die Installation einer Web-Anwendung als Open Webapp ist die serverseitige Bereitstellung einer – weiteren... – Manifestdatei, die die Konfigurationseinstellungen der Anwendung in einer durch die OWA-Spezifikation festgelegten JSON-Syntax deklariert. Die Installation der Anwendung als OWA durch Firefox kann dann mittels einer JavaScript API clientseitig gesteuert werden.

Abgesehen davon können OWAs beliebige Ausdrucksmittel für Web Applikationen verwenden, die durch den Firefox Browser unterstützt werden. So stellen OWAs selbst insbesondere keine spezifischen Ausdrucksmittel für den Offline-Betrieb einer Anwendung zur Verfügung. Die Offlinefähigkeit muss also unter Verwendung der hierfür vorgesehenen standardisierten Ausdrucksmittel umgesetzt werden, die wir im vorangegangenen Abschnitt beschrieben haben.

OWA Manifest

Wie das nachfolgende Beispiel zeigt, legt ein OWA-Manifest insbesondere den Namen einer Anwendung fest, unter dem diese als OWA installiert wird und muss darüber hinaus die Startseite angeben, die beim Starten der Anwendung geöffnet werden soll. Diese muss relativ zur Ursprungsdomain angegeben werden, d. h. bei Verwendung von Tomcat oder Aptana muss die relative URL mit dem Pfadsegment beginnen, das den Namen der Anwendung bezeichnet. Dies gilt für alle Referenzen auf Ressourcen, die im Manifest verwendet werden können, z. B. für ein Icon, das ggf. in verschiedenen Größen festgelegt werden kann. Dieses wird verwendet, um nach erfolgreicher Installation dem Nutzer auf dem Startbildschirm oder im App Menü des Endgeräts eine Zugriffsmöglichkeit auf die Anwendung zu geben.

Zum Zweck der Installation einer OWA auf Firefox OS können außerdem Angaben bezüglich der durch den Nutzer bei der Installation auszusprechenden Zugriffsberechtigungen für Gerätefunktionen und Daten dem Manifest hinzugefügt werden. Nachfolgend sehen Sie als Beispiel eine Manifestdatei, wie sie in den Implementierungsbeispielen zur vorliegenden Lerneinheit verwendet wird. Für eine Dokumentation aller Ausdrucksmittel bezüglich eines OWA-Manifests verweisen wir auf die diesbezügliche Dokumentation von Mozilla.

[www https://developer.mozilla.org/en-US/Apps/Build/Manifest](https://developer.mozilla.org/en-US/Apps/Build/Manifest)



Quellcode

### OWA Manifestdatei

```
001 {
002     "name": "mme2webapp",
003     "description": "sample offline webapp",
004     "icons": {
005         "128": "/img/hmicon.png"
006     },
007     "launch_path": "/index.html",
008     "developer": {
009         "name": "dieschnittstelle.org",
010         "url": "http://www.dieschnittstelle.org"
011     },
012     "default_locale": "en"
013 }
```

Bei Auslieferung einer Manifestdatei durch den Server sollte als Content-Type der Wert `application/x-web-app-manifest+json` angegeben werden, und als Dateiendung ist das Suffix `.webapp` gebräuchlich.

OWA API

Die Installation einer Anwendung als OWA erfolgt über die [www Schnittstelle eines Objekts](#), das über das Attribut `navigator.mozApps` in Firefox zugreifbar ist. Wichtigste Angabe beim Installationsvorgang ist ein eindeutiger Identifikator der Anwendung, als welcher die URL der Manifestdatei verwendet wird. Dieser kann u. a. den folgenden Funktionen als `webappid` Argument übergeben werden:

- `checkInstalled(webappid)`
- `install(webappid)`

Vergleichbar der Verwendung der IndexedDB API werden beide Funktionen asynchron ausgeführt und geben ein `request`-Objekt zurück, auf dem Callbacks für `onsuccess` und `onerror` gesetzt werden können.



Quellcode

#### Callback

```
001 // Überprüfe, ob die Anwendung bereits installiert ist.
002 var checkrequest = navigator.mozApps.checkInstalled(webappid);
003
004 // Deklariere einen Callback für den Erfolg einer Überprüfung. Falls
005 // webappid bereits installiert ist, ist das result Attribut auf
006 // request.success bzw. event.target gesetzt.
007 checkrequest.onsuccess = function(event) {
008     if (checkrequest.result) {
009         alert("The webapp has already been installed!");
010     } else {
011         // initiiere die Installation
012         var installrequest = navigator.mozApps.install(webappid);
013         // deklariere Callbacks für den Erfolgs- und Fehlerfall
014         installrequest.onsuccess = function() {
015             alert("Installation successful!");
016         };
017         installrequest.onerror = function(err) {
018             alert("Installation failed!");
019         };
020     }
021 };
```

Im Gegensatz zu den Implementierungsbeispielen der Lerneinheit wird in diesem Codebeispiel keinerlei Nutzerinteraktion mit Ausnahme von finalen Statusmitteilungen via `alert()` initiiert. Bei Aufruf der `install()` Methode löst Firefox jedoch selbst einen Bestätigungsdialog aus, die in folgende Abbildung zu sehen ist und dem Nutzer die Möglichkeit gibt, die Installation ggf. abzubrechen. Nach erfolgreicher Installation wird in aktuellen Android-Versionen das im Manifest angegebene App Icon oder ein Default Icon auf dem Startbildschirm platziert – siehe hierfür ebenfalls folgende Abbildung.

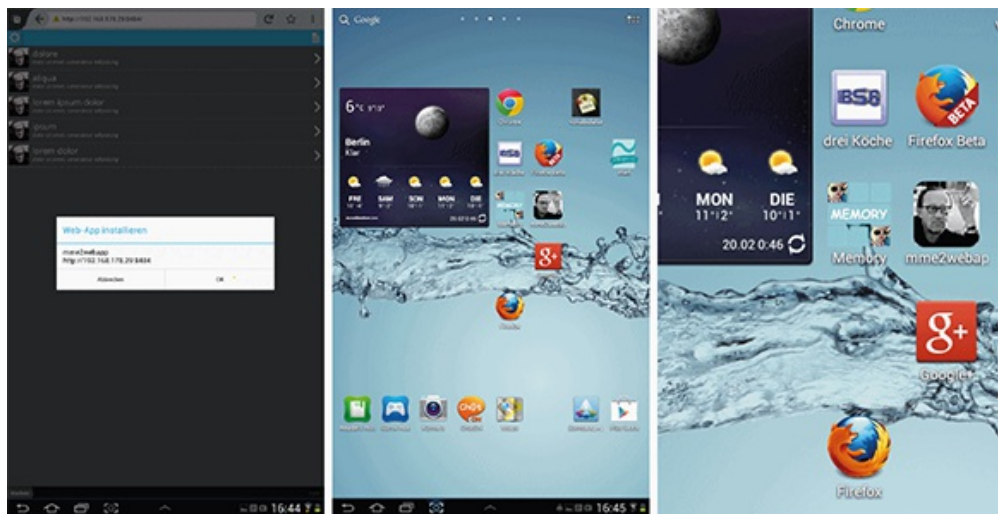


Abb.: Rückbestätigung der Webapp Installation (links) und Darstellung des Webapp-Icons der App `mme2webapp` auf dem Home Bildschirm eines Android Tablets (Mitte, rechts)

In älteren Versionen vor Version 4.\* muss dies evtl. durch Zugriff auf die interne URL `about:apps` manuell veranlasst werden. In MacOS resultiert die Installation hingegen in der Hinzufügung eines Eintrags für die App zur „Programme“-Liste.

Für weitere Informationen zur Installation verweisen wir auf die Dokumentation von Mozilla.

[www. https://developer.mozilla.org](https://developer.mozilla.org).



## 2.3 Alternativen zu OWA

Zwar sind sowohl die Ausdrucksmittel OWA für Manifestdateien, als auch die JavaScript API für die Installation von OWAs in ihrem Umfang überschaubar. Dennoch benötigen Sie für die Installation einer Anwendung als OWA eine individuelle JavaScript-Implementierung oder eine generische JavaScript-Bibliothek – sehen Sie für letzteres z. B. `webapp.js` in den Implementierungsbeispielen.

Im Gegensatz dazu gestaltet sich die „Installation“ von Webanwendungen über Safari auf iOS denkbar einfach. Erforderlich ist hierfür lediglich die Angabe eines `<meta>` Tags im Startdokument der Anwendung sowie, falls gewünscht, die Angabe eines `<link>` Tags bezüglich eines App Icons. Dieses muss jedoch den Anforderungen des jeweiligen iOS Geräts entsprechen und ggf. mittels Media Queries in verschiedenen Varianten bereitgestellt werden.



Quellcode

### Installation von Webanwendungen über Safari auf iOS

```
001 <!-- Deklaration der Installierbarkeit als Webapp -->
002 <meta name="apple-mobile-web-app-capable" content="yes">
003 <!-- Deklaration eines Start Icons -->
004 <link rel="apple-touch-startup-image" href="/img/hmicon.png">
```

Wird ein Dokument, für das der `<meta>` Tag für Webapps gesetzt ist, über das Menü von Safari als „Favorit“ dem Startbildschirm hinzugefügt, dann hat die Betätigung des dafür erzeugten Icons denselben Effekt wie das Öffnen einer via Firefox als OWA installierten Anwendung auf Android: in beiden Fällen werden die Anwendungen in Vollbildansicht und ohne die generischen Bedienelemente des Browsers geöffnet und unterscheiden sich darin nicht von der Darstellung nativer Anwendungen auf den betreffenden Plattformen.

Während OWAs jedoch darüber hinaus mit Blick auf die Verwendung in Firefox OS über weitere Ausdrucksmittel verfügen, insbesondere über die Möglichkeit zur Deklaration von Permissions, sind Webapps für iOS auf diesen Darstellungsaspekt eingeschränkt. Für die Nutzererfahrung einer App als App ist dieser freilich wesentlich.

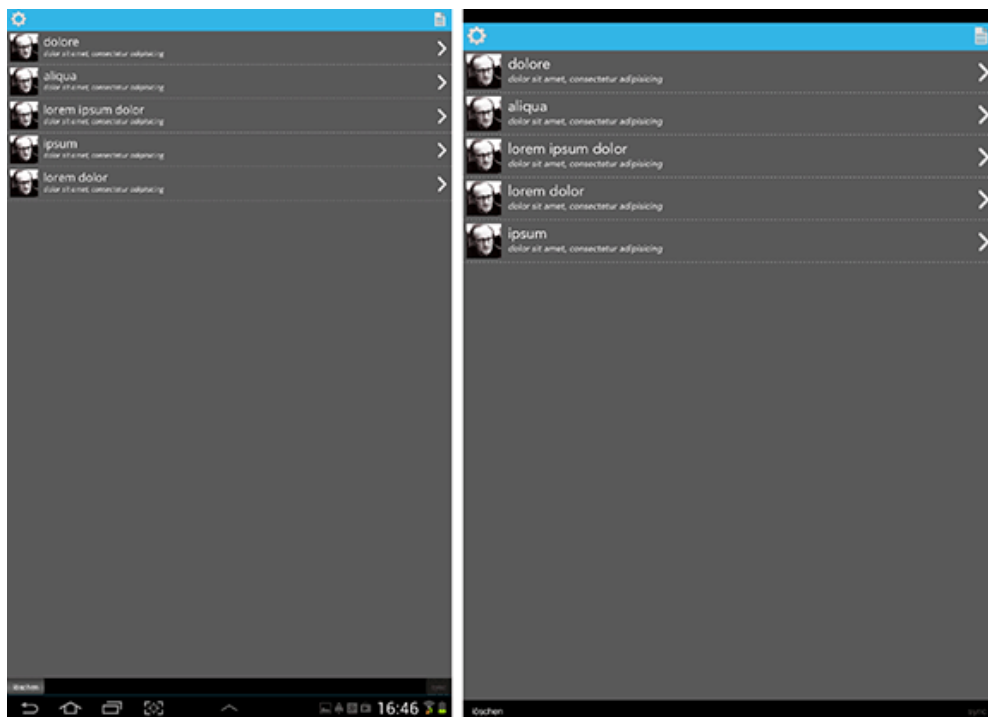


Abb.: Vollständiger Screenshot eines Android Tablets (links) und eines iPads (rechts) nach Installation einer Anwendung als Webapp



## Zusammenfassung

- Aufgabe der Manifestdatei ist es festzulegen welche Ressourcen einer Webanwendung offline verfügbar sein sollen, auf welche Ressourcen grundsätzlich online zugegriffen werden sollen und welche Fallback-Lösungen für den Fall der Nichtverfügbarkeit einer Ressource anzuwenden sind.
- Verwendet ein HTML-Dokument ein `manifest` Attribut, dann erfolgt der Zugriff auf die darin referenzierte Ressource und die Verarbeitung von deren Inhalten nach einem durch die Spezifikation für Offline Webanwendungen exakt beschriebenen Verfahren.
- Eine serverseitig aktualisierte Ressource, die durch Zuordnung zum CACHE Status als Offline Ressource deklariert ist, wird erst dann vom Browser geladen wird, wenn auch eine Aktualisierung der Manifestdatei selbst vorliegt.
- Voraussetzung für die Installation einer Web-Anwendung als Open Webapp ist die serverseitige Bereitstellung einer Manifestdatei, die die Konfigurationseinstellungen der Anwendung in einer durch die OWA-Spezifikation festgelegten JSON-Syntax deklariert.
- Die Installation von Anwendungen als Webapps auf einem Endgerät entkoppelt diese Anwendungen auf der Ebene des User Interface von den Bedienelementen eines Browsers und ermöglicht damit eine Nutzung, die dem Nutzungskonzept nativer - ggf. mobiler Anwendungen entspricht.

Sie sind am Ende dieser Lerneinheit angelangt. Auf den folgenden Seiten finden Sie noch Übungen.

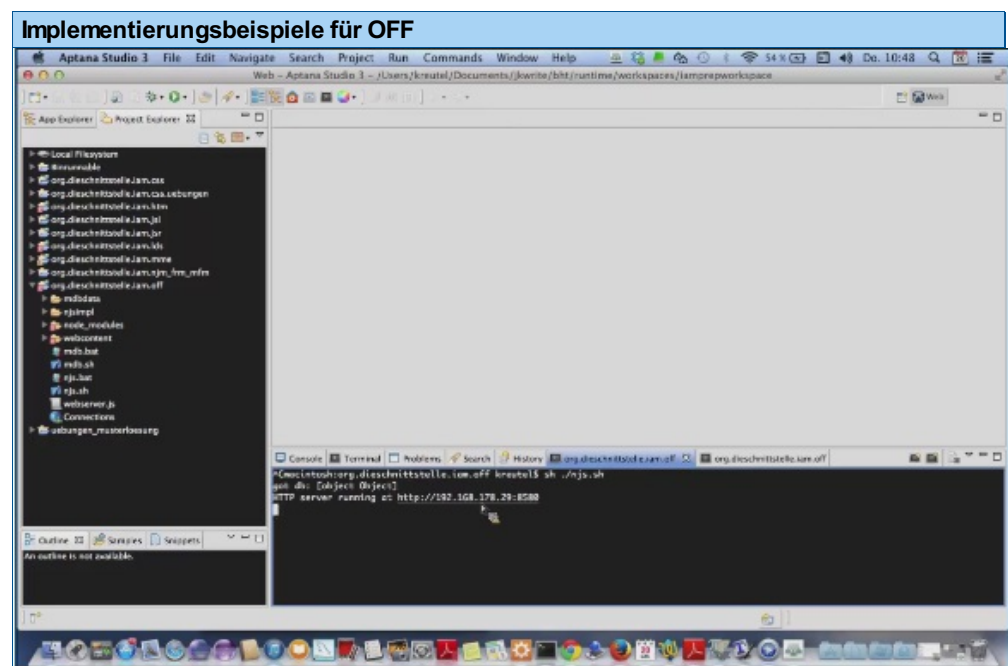
## Übungen

Das Projekt `org.dieschnittstelle.iam.off` enthält die Implementierungsbeispiele für die vorliegende Lerneinheit. Es übernimmt die in der Lerneinheit LDS verwendete Ansicht sowie weitgehend die Implementierung von `ListviewViewController.js`. Die Implementierung von `DataItemCRUDOperations` wurde dahingehend modifiziert, dass wir die darin enthaltenen Funktionen durch Aufruf der im `lib` Verzeichnis enthaltenen generischen Komponenten für lokalen bzw. remote sowie synchronisierten Datenzugriff implementieren.

Die prüfungsverbindlichen Übungen und deren Bepunktung werden durch die jeweiligen Lehrenden festgelegt.



Film



© Beuth Hochschule Berlin - Dauer: 08:06 Min. - Streaming Media 15 MB



## Übung OFF-01

### Beispielanwendung OFF

Die Error-Meldungen, die Ihnen Aptana evtl. für die Datei `js/lib/IndexedDBCRUDOperationsImpl.js` anzeigt, können Sie darauf zurückführen, dass die IndexedDB API der Entwicklungsumgebung unbekannt ist.

#### off.manifest

- Welche Funktionalität wird durch die Aufrufe der URLs mit dem Segment `http2mdb` erbracht und wo ist diese Funktionalität in der vorliegenden Applikation implementiert?
- Wo wird auf die URL `available` zugegriffen?
- Weshalb ist die Deklaration dieser beiden URLs als `NETWORK` URLs erforderlich?

#### js/lib/webapp.js

- Wie kann die Implementierung auf das Ergebnis der beiden Funktionsaufrufe bezüglich `checkInstalled()` und `install()` auf `navigator.mozApps` reagieren?
- Welche beiden JavaScript APIs, die in der Anwendung verwendet werden, erfordern ein dem entsprechendes Implementierungsmuster, um auf das Resultat eines Aufrufs zuzugreifen?

#### js/DataItemCRUDOperations.js

- Welche Aufgabe kommt der Variablen `crudimpl` zu und wie wird die Variable instantiiert?

#### js/lib/\*CRUDOperationsImpl.js

- Wo werden die Werte für die Argumente `onsuccess` und `onerror`, die den meisten Funktionen übergeben werden, gesetzt?
- Weshalb ist die hier umgesetzte Form der Entgegennahme von Funktionsergebnissen erforderlich?

#### js/lib/RemoteMasterSyncedCRUDOperationsImpl.js

- Wozu dient die `countdown` Variable, die an verschiedenen Stellen in der Funktion `syncObjects` verwendet wird? Weshalb ist sie hier erforderlich? (Siehe dazu auch Übung LDS-01)
- Welchen Funktionen wird in `syncObjects()` das Argument `{object: object}` übergeben, und wie wird dieses Argument von den betreffenden Funktionen verwendet?
- An welcher Stelle im Programmcode wird das besagte als Argument übergebene Objekt `{object: object}` tatsächlich verwendet?
- Weshalb ist diese Form der Übergabe von Werten „innerhalb einer Funktion“ im vorliegenden Fall erforderlich? Wäre es auch möglich den Wert der Variable `object` direkt auszuwerten, ohne ihn auf die vorliegende Art „durch einen Funktionsaufruf hindurch“ zu übergeben?

Bearbeitungszeit: 60 Minuten



## Übung OFF-02

### Offline-Anwendung

#### Aufgabe

Machen Sie die in NJM3-LDS2 entwickelte Anwendung als Offline-Webanwendung mit lokaler Datenspeicherung verfügbar.

#### Anforderung

1. Unter „Offline-Betrieb“ wird der Zugriff auf die Anwendung ohne laufenden NodeJSWebserver, aber mit verfügbarer Internet-Verbindung verstanden.
2. Für die Ausführung von CRUD Operationen soll die in LDS2 entwickelte Implementierung mit IndexedDB verwendet werden.
3. Falls beim Starten der Anwendung kein Zugriff auf den NodeJS Server möglich ist, soll die lokale CRUD Implementierung automatisch ausgewählt werden.
4. In der Anwendung selbst auf den Webserver hochgeladene Bilder, die von Imgbox-Elementen verwendet werden, brauchen nicht offline verfügbar zu sein.
5. Der Upload von Bildern braucht im Offline-Betrieb ebenfalls nicht funktionsfähig zu sein.

#### Bearbeitungshinweise

- Diese Aufgabe basiert im wesentlichen auf der Umsetzung von LDS2. Hier geht es dann vor allem noch darum, dass Sie das Offline-Manifest für Ihre Anwendung erstellen und verwenden.
- **Anforderung 4, Anforderung 5:** Getestet wird der Offline-Betrieb für Imgbox-Elemente, die externe Bilder via deren URL referenzieren.

Bearbeitungszeit: 90 Minuten

## Prüfungsfragen

Versuchen die hier aufgeführten Fragen zu den Inhalten der Lerneinheit selbständig kurz zu beantworten, bzw. zu skizzieren. Wenn Sie eine Frage noch nicht beantworten können kehren Sie noch einmal auf die entsprechende Seite in der Lerneinheit zurück und versuchen sich die Lösung zu erarbeiten.



Formulieren

### Übung OFF-03

#### Offline Applikationen

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Was legt die Cache Manifest Datei fest?
2. Was für ein Notationsformat (z. B. JSON, XML, andere Textformate) wird für die Cache Manifest Datei verwendet?
3. Was passiert, wenn ein Cache Manifest eine Ressource als Offline Ressource referenziert und diese nicht geladen werden kann?
4. Was passiert, wenn im Onlinebetrieb auf eine Ressource zugegriffen wird, die gar nicht im Cache Manifest angegeben wird?
5. Wann greift der Browser auf die Cache Manifest Datei zu?
6. Wann werden Offline Ressourcen vom Browser ins Cache geladen? Bei der Erstellung oder Aktualisierung des Cache oder wenn der erste Zugriff auf die Ressource erfolgt?
7. Wann werden serverseitig aktualisierte Offline-Ressourcen vom Browser geladen und wann sind sie im Browser verfügbar?
8. Wo können im Cache Manifest Wildcards verwendet werden?
9. Wozu dient die `ApplicationCache` API?
10. Weshalb ist die Verwendung des Offline Cache nicht „natürlicherweise“ für das Caching dynamischer Inhalte, die durch die Nutzer einer Anwendung bereitgestellt werden, geeignet? Was für ein Workaround könnte verwendet werden, um es geeignet zu machen?

Bearbeitungszeit: 45 Minuten



Formulieren

### Übung OFF-04

#### Webapps

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Ist der Begriff der „Webapp“ wohl definiert bzw. durch einen Standard abgedeckt?
2. Inwiefern werden mit Mozilla Open Web Apps (OWA) Funktionen und Handhabung nativer mobiler Anwendungen für Web-Applikationen verfügbar gemacht? Nennen sie 4 Aspekte.
3. Sind Mozilla OWA grundsätzlich offline-fähige Anwendungen?
4. Wie kann eine OWA bei Verwendung eines gewöhnlichen Firefox Browsers installiert werden?
5. Was kann im Manifest einer OWA angegeben werden? Nennen Sie 4 Angaben.

Bearbeitungszeit: 45 Minuten