DIG 4104C: WEB DESIGN WORKSHOP

Lecture 09: Extending web apps offline: Working in offline mode with the Application Cache

Summer 2015

D. Novatnak

Outline

Objectives

- Expand student knowledge to manage a web application while offline
- Expose students to the features of the Application Cache API in HTML5
- Expose students to the language and syntax necessary to leverage the Application Cache API in their own applications
- Expose students to Service Workers in HTML5

At the end of this lecture, you should be able to….

- Create an offline pages through usage the Application Cache API
- Format and utilize the application cache and manifest
- Set up the Apache websever to handle application caching
- Explain what a service worker is, what is it useful for, and how to integrate into a project.

HTML5 OFFLINE APPS

OFFLINE? PAGES STILL WORK!

Going offline

- In order to enable users to continue interacting with Web applications and documents even when their network connection is unavailable, you can provide a manifest which lists the files that are needed:

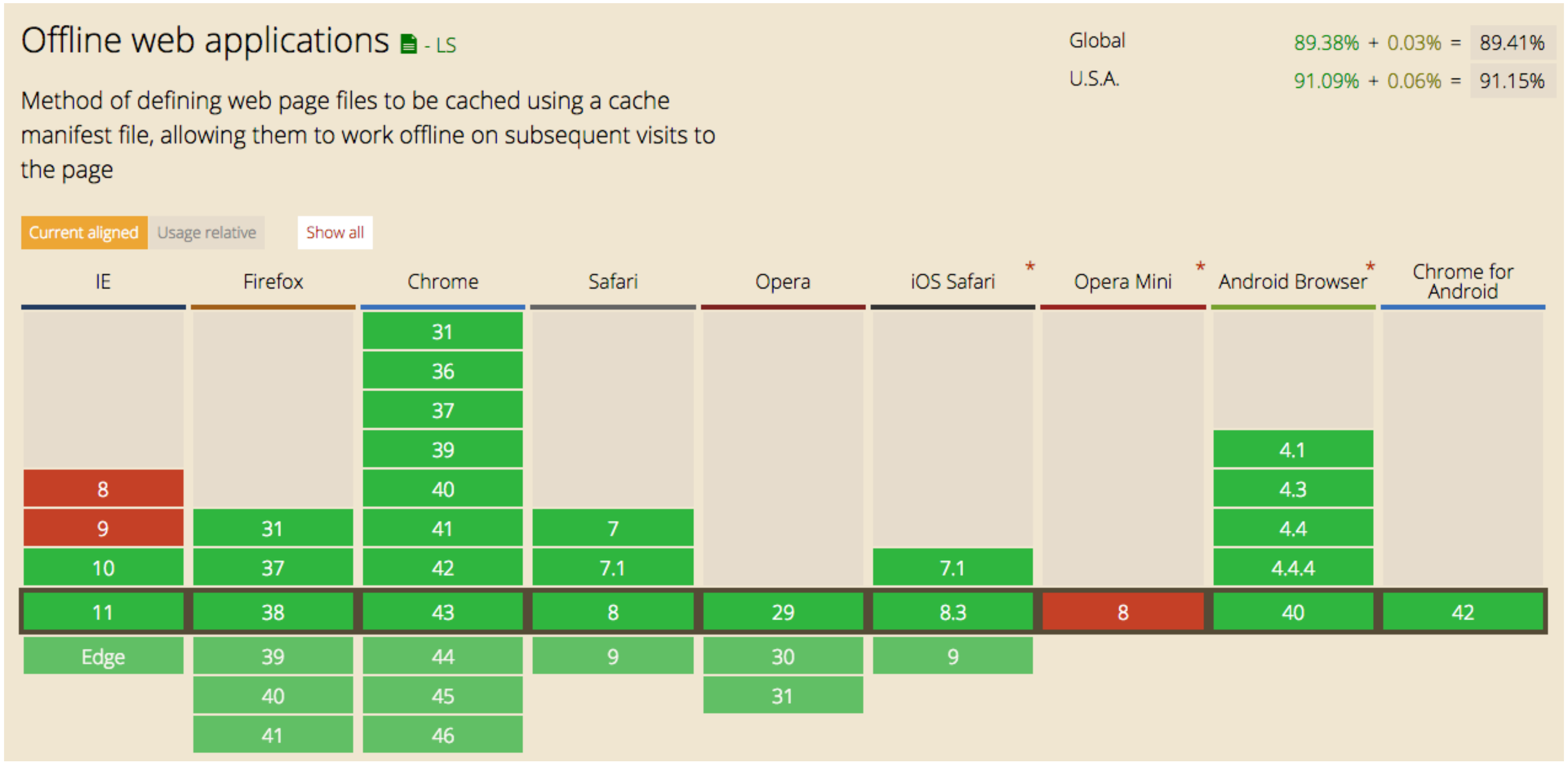
  1. for the web application to work offline
  2. which causes the user's browser to keep a copy of the files for use offline.

- These resources load and work correctly even if users click the refresh button when they are offline.

Overview

- An application cache provides the following benefits:
  - Offline browsing: users can navigate a site or use a web application although they are offline
  - Speed: cached resources are local, and therefore load faster
  - Reduced server load: the browser only downloads resources that have changed from the server.

Browser compatibility



Offline web applications - LS

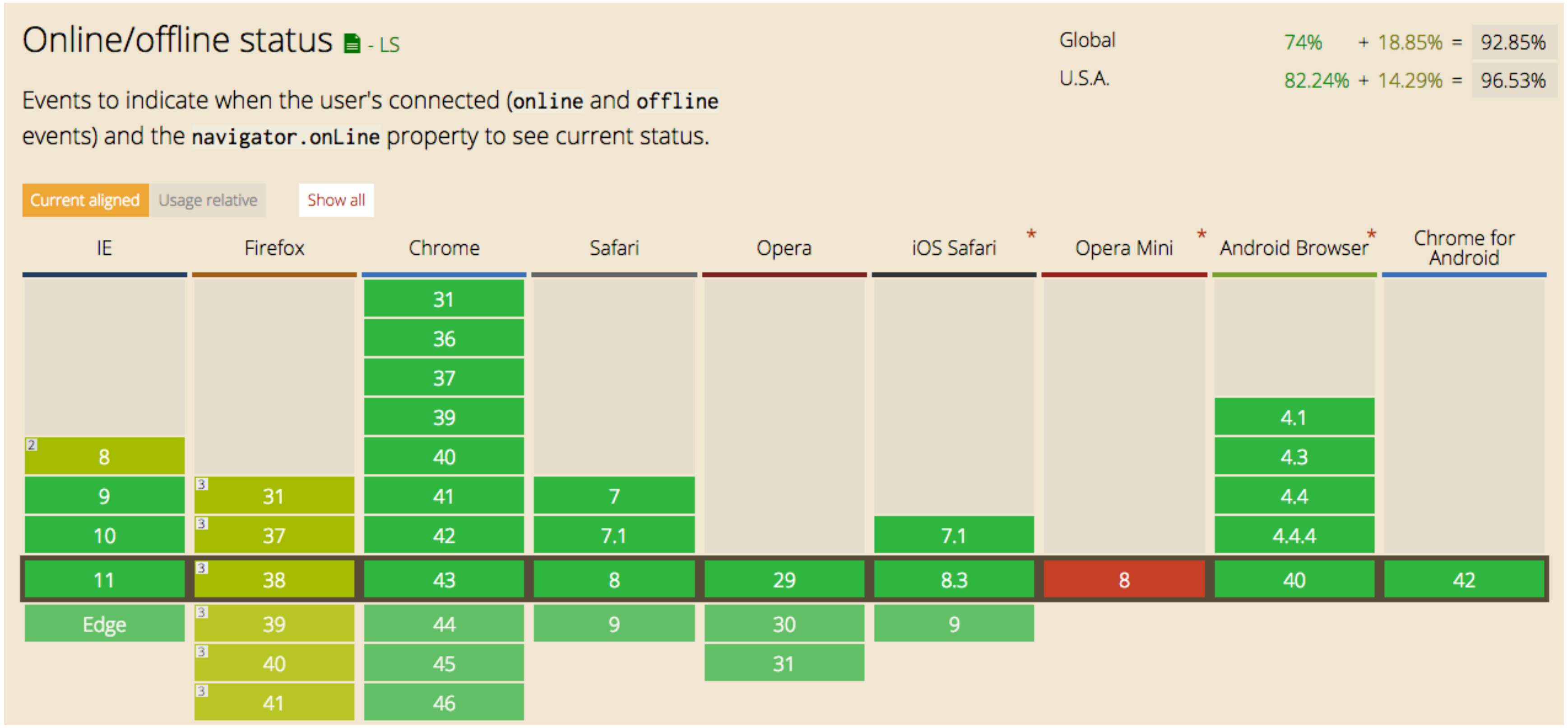
Method of defining web page files to be cached using a cache manifest file, allowing them to work offline on subsequent visits to the page

Global	89.38%	+ 0.03%	=	89.41%
U.S.A.	91.09%	+ 0.06%	=	91.15%

Current aligned | Usage relative | Show all

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		🟩 31						
		🟩 36						
		🟩 37						
		🟩 39					🟩 4.1	
🟥 8		🟩 40					🟩 4.3	
🟥 9	🟩 31	🟩 41	🟩 7				🟩 4.4	
🟩 10	🟩 37	🟩 42	🟩 7.1		🟩 7.1		🟩 4.4.4	
🟩 11	🟩 38	🟩 43	🟩 8	🟩 29	🟩 8.3	🟥 8	🟩 40	🟩 42
🟩 Edge	🟩 39	🟩 44	🟩 9	🟩 30	🟩 9			
	🟩 40	🟩 45		🟩 31				
	🟩 41	🟩 46						

Browser compatibility



Online/offline status - LS

Events to indicate when the user's connected (`online` and `offline` events) and the `navigator.onLine` property to see current status.

Global	74%	+ 18.85%	=	92.85%
U.S.A.	82.24%	+ 14.29%	=	96.53%

Current aligned | Usage relative | Show all

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		36						
		37						
		39					4.1	
[2] 8		40					4.3	
9	[3] 31	41	7				4.4	
10	[3] 37	42	7.1		7.1		4.4.4	
11	[3] 38	43	8	29	8.3	8	40	42
Edge	[3] 39	44	9	30	9			
	[3] 40	45		31				
	[3] 41	46						

Usage

- In order to cache files, you must do the following:
  - Create a manifest file that lists the files and other web resources you want to cache.
  - Reference the manifest file in the header of every page you want cached.

How does it work?

- A manifest file that contains a list of URLs
  - HTML
  - CSS
  - JavaScript
  - images, or any other kind of resource.
- Reads the list of URLs from the manifest file, download the resources, cache them locally, and automatically keep the local copies up to date as they change
- without a network connection, your web browser will automatically switch over to the local copies instead.

How do we use the manifest file?

- The manifest file defines which web resources are cached when a user browses to your site. The manifest file typically has the extension .appcache or .manifest.

- Each webpage must add a manifest attribute to the HTML element similar to this:

  - On the server side
    - must be served with the content type text/cache-manifest
    - for Apache server:
      - AddType text/cache-manifest

  - Inside HTML code
    - <!DOCTYPE HTML>
    - <html manifest="/cache .manifest">
    - <body>....

Inside the Manifest File

- You can use an absolute or relative URL. The cache manifest file lists the files that will be cached, using the format:

```
CACHE MANIFEST
# Version 1

CACHE:
script/scriptfilename1.js
css/cssfilename.css
images/imagename1.png
images/imagename2.jpg
images/imagename3.png

FALLBACK:
# This will enable imagename4.png to be as a replacement for
# all resources under the images dir when there is no network
connectivity.
images/ images/imagename4.png

NETWORK:
# This will prevent other network resources from being accessed.
images/imagename5.png
```

Inside the Manifest File

- The first line of the manifest must read CACHE MANIFEST and the lines that follow it list the web resources you want to cache. You can use the # symbol to make comments.
- Alternatively, web resources that need to be cached can be specified by adding a “CACHE:” header section before the resources (as shown previously).
- In addition, fallback resources can be defined to replace general or specific web resources when there is no network connectivity.
- This is done by adding a “FALLBACK:” header section before the resources. These resources can be wildcard to specify a catch all mechanism (as shown previously).

Inside the Manifest File

- Be aware that there is a space after the first "images/" in the FALLBACK: section.
  - This indicates that any files contained under the "images/" directory will fall back to a specific web resource (for example, images/imagename4.png) if they cannot be accessed from the network.
- Also, web resources can be specified to only be loaded from the network.
  - This is done by adding a "NETWORK: " header section before the resources. This functionality can be used to scope down the number of resources that can be accessed from the network, thus, creating an allowed only list (as shown previously).

In a nutshell

Section	Description
CACHE	This is the default section for entries. Files listed under this header (or immediately after the CACHE MANIFEST) will be explicitly cached after they're downloaded for the first time.
NETWORK	Files listed under this section are white-listed resources that require a connection to the server. All requests to these resources bypass the cache, even if the user is offline. Wildcards may be used.
FALLBACK	An optional section specifying fallback pages if a resource is inaccessible. The first URI is the resource, the second is the fallback. Both URIs must be relative and from the same origin as the manifest file. Wildcards may be used.

Dependence

- ApplicationCache functionality is independent of HTTP caching headers.
- The manifest file implicitly includes itself as a page to be cached.
- It also needs to have the **same** domain of origin as the page that contains it.

How does it work?

1. Finding “Manifest” attribute triggers checking event. This step always occur.
2. Browser never visited this web page. Then resources listed in the chache manifest will be cached by sequence of events.

   downloading->progress->cached
3. For Previously visited site, then check the last update.

   if no, then noupdate event triggars.

   if yes, then re-downloading every single resource listed in the cache manifest

Application Cache Properties

API Name	Summary
oncached	The resources listed in the manifest have been downloaded, and the application is now cached.
onchecking	The user agent is checking for an update, or attempting to download the manifest for the first time. This is always the first event in the sequence.
ondownloading	The user agent has found an update and is fetching it, or is downloading the resources listed by the manifest for the first time.
onerror	Indicates an error has occurred.
onnoupdate	Indicates the manifest has not changed.
onobsolete	The Webpage is associated with an application cache whose group is marked as obsolete.
onprogress	The user agent is downloading resources listed by the manifest. The event object's total attribute returns the total number of files to be downloaded. The event object's loaded attribute returns the number of files processed so far.
onupdateready	The ApplicationCache object's cache host is associated with an application cache whose application cache group's update status is idle, and whose application cache group is not marked as obsolete, but that application cache is not the newest cache in its group.
status	Returns the current status of the application cache, as given by the constants defined below.

Application Cache Methods

API Name	Summary
abort	Cancels the application cache download process. This method is intended to be used by Web applications showing their own caching progress UI, in case the user wants to stop the update (e.g., because bandwidth is limited).
swapCache	Switches to the most recent application cache, if there is a newer one. If there isn't, throws an **InvalidStateError**exception. This does not cause previously-loaded resources to be reloaded; for example, images do not suddenly get reloaded and style sheets and scripts do not get reparsed or reevaluated. The only change is that subsequent requests for cached resources will obtain the newer copies. The **updateready** event will fire before this method can be called. Once it fires, the Web application can, at its leisure, call this method to switch the underlying cache to the one with the more recent updates. To make proper use of this, applications have to be able to bring the new features into play; for example, reloading scripts to enable new features. An easier alternative to **swapCache()** is just to reload the entire page at a time suitable for the user, using **location.reload()**.
update	Invokes the application cache download process. Throws an InvalidStateError exception if there is no application cache to update. Calling this method is not usually necessary, as user agents will generally take care of updating application caches automatically. The method can be useful in situations such as long-lived applications. For example, a Web mail application might stay open in a browser tab for weeks at a time. Such an application could want to test for updates each day.

Examples

- Creating your Own Offline Application
- SitePoint Offline Web Application Tutorial
- Task Manager
- http://html5demos.com/offlineapp

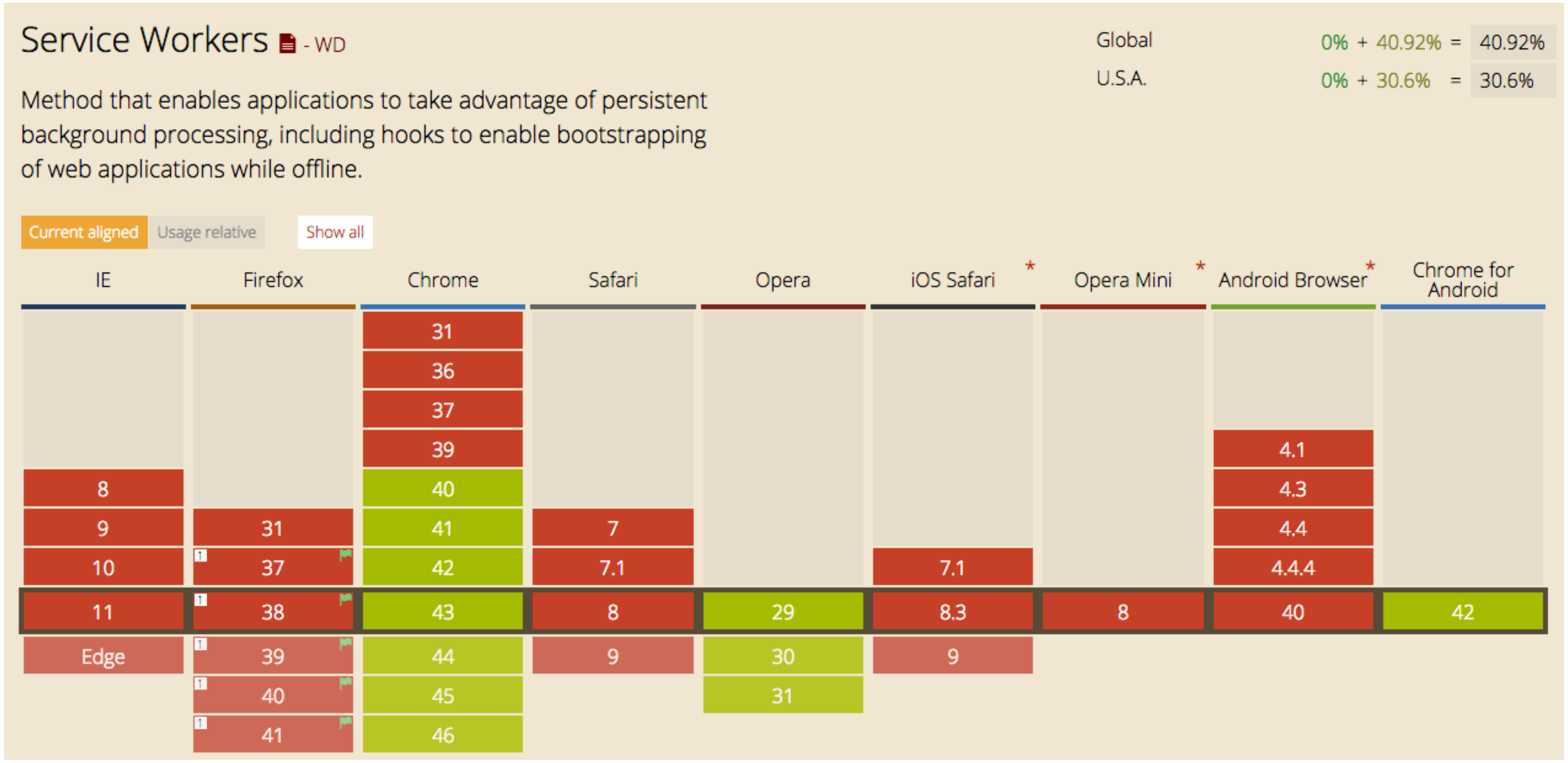
SERVICE WORKERS

Let someone else do the work…in the background

Using service workers

- A service worker is a script that is run by your browser in the background, separate from a web page, opening the door to features which don't need a web page or user interaction
    - push messages, background sync, and geofencing,
    - intercept and handle network requests, including programmatically managing a cache of responses.

Browser compatibility



Service Workers - WD

Method that enables applications to take advantage of persistent background processing, including hooks to enable bootstrapping of web applications while offline.

Global	0%	+ 40.92%	=	40.92%
U.S.A.	0%	+ 30.6%	=	30.6%

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		36						
		37						
		39					4.1	
8		40					4.3	
9	31	41	7				4.4	
10	37	42	7.1		7.1		4.4.4	
11	38	43	8	29	8.3	8	40	42
Edge	39	44	9	30	9			
	40	45		31				
	41	46						

Things to note about a service worker:

- It's a JavaScript Worker, so it can't access the DOM directly. Instead, a service worker can communicate with the pages it controls by responding to messages sent via the postMessage interface, and those pages can manipulate the DOM if needed.
- Service worker is a programmable network proxy, allowing you to control how network requests from your page are handled.
- It will be terminated when not in use, and restarted when it's next needed, so you cannot rely on global state within a service worker's onfetch and onmessage handlers. If there is information that you need to persist and reuse across restarts, service workers do have access to the IndexedDB API.
- Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Jake Archibald's article.

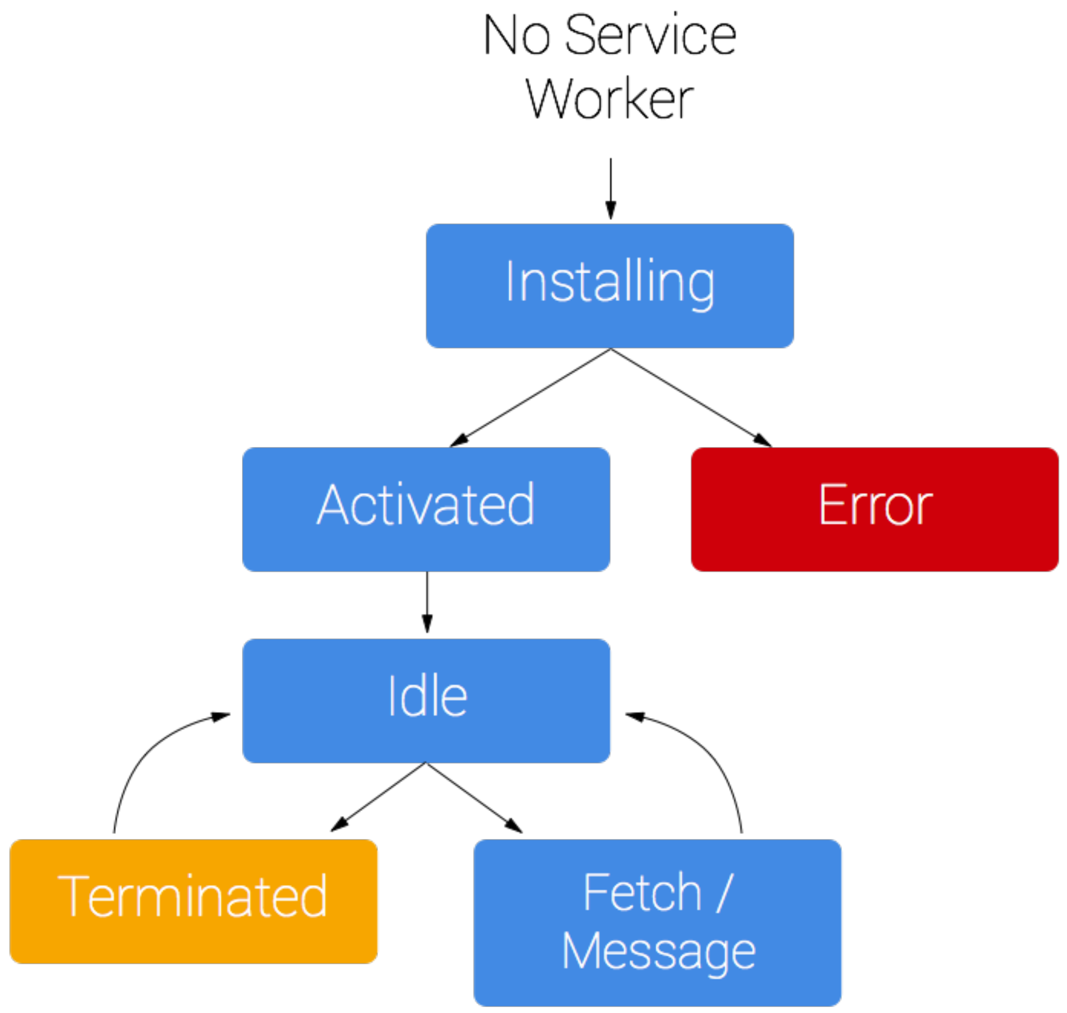
Service Worker Lifecycle

- A service worker has a lifecycle which is completely separate from your web page.
- To install a service worker for your site, you need to register it, which you do in your page's JavaScript. Registering a service worker will cause the browser to start the service worker install step in the background.
- Typically during the install step, you'll want to cache some static assets. If all the files are cached successfully, then the service worker becomes installed. If any of the files fail to download and cache, then the install step will fail and the service worker won't activate (i.e. won't be installed). If that happens, don't worry, it'll try again next time. But that means if it does install, you know you've got those static assets in the cache.

Service Worker Lifecycle

- When installed, the activation step will follow and this is a great opportunity for handling any management of old caches, which we'll cover during the service worker update section.
- After the activation step, the service worker will control all pages that fall under its scope, though the page that registered the service worker for the first time won't be controlled until it's loaded again.
- Once a service worker is in control, it will be in one of two states: either the service worker will be terminated to save memory, or it will handle fetch and message events which occur when a network request or message is made from your page.

Service Worker Lifecycle



More on Service Worker Lifecycle

- https://jakearchibald.github.io/isserviceworkerready/resources.html

Discussion

- How would you use “offline” applications or service workers?

Questions?

References

- Crowther, R., & Lennon, J. (n.d.). HTML5 in action.
- Offline Web Applications (n.d.). Retrieved June 12, 2015, from http://www.w3.org/TR/2011/WD-html5-20110525/offline.html
- Service Workers(Second Edition) (n.d.). Retrieved June 12, 2015, from http://www.w3.org/TR/service-workers/
- Others embedded.