WebApp & Firefox OS

Laurent Jouanneau
Course on Mozilla Education and Technologies @ Evry
Janvier, 2013

Evolution des applis web

- Traditionnelle : page générée par le serveur
  - Code métier coté serveur
  - → site de consultations
- Ajax
  - Services web avec xmlHttpRequest
  - Code métier coté serveur
  - → sites de consultations avec quelques fonctionnalités applicatives
- « all in one page »
  - toute l'interface dans une même page HTML
  - ajax à gogo
  - souvent avec une logique MVC en JS

Evolutions des applis web

- Autonome
  - Page embarquant tout : interface + code métier
- Autonome + mise en cache de tous les fichiers
  - « installée » en local
- Autonome packagée
  - installée réellement en local

Evolutions des applis web

Et si on lançait les applications en dehors du navigateur classique ?

Un peu d'histoire

- Demarrage projet mozilla (1998) : XUL
  - développement d'une interface avec un langage descriptif en xml
  - view source chrome://browser/content/
- Firefox (2004)
- Appli XUL : XulRunner (2006)
- Prism (2007) : une appli XulRunner pour « lancer » des applis web, en dehors du navigateur
- Open WebApps & MarketPlace (2012)

Transformer une app en Web App

Open WebApp

- Open Web App =
  - une appli web, autonome ou pas
    - → une url ou un ensemble de fichiers packagés
  - un fichier « manifest » pour déclarer l'application
  - « installée » via un navigateur à partir d'un site web (ex : un « store »)
    - → icône sur le bureau
  - lancé par un navigateur nu (sans interface) : le « web runtime »

Lanceur de webapp

- Une fois installée, une webapp sur un desktop, c'est :
  - Un raccourci placé dans le menu application du système
  - Un répertoire dans le $HOME, contenant
    - Un exécutable « stub » lançant l'appli via le « xulrunner » de Firefox
    - Un profil spécifique
    - L'icone de l'appli, un manifest...
- ex: Firefox 17 + installation d'une appli à partir de https://marketplace.firefox.com/

Web App : manifest

- fichier JSON qui sera utilisé lors de l'installation et/ou l'enregistrement de l'appli dans un store
- stocké quelque part sur un site, en *.webapp
- Content-type : application/x-web-app-manifest+json
- name et description obligatoire

```
{
  "name": "My App",
  "description": "My elevator pitch goes here",
  "launch_path": "/",
  "icons": {
    "128": "/img/icon-128.png"
  },
  "developer": {
    "name": "Your name or organization",
    "url": "http://your-homepage-here.org"
  },
  "default_locale": "en"
}
```

Web App : manifest

- activities
- fullscreen, orientation, version
- permissions, installs_allowed_from
- type = web, privileged, certified
- csp = Content Security Policy = chaine décrivant la politique de sécurité de l'appli (voir chapitre CSP)
- appcache_path = chemin fichier cache.manifest
- https://developer.mozilla.org/en-US/docs/Apps/Manifest

CSP – Content Security Policy

- Indique les sources de contenu autorisées dans les différentes parties de l'appli → défini quoi et où
  - sécurise contre attaques type XSS, data injection
  - attention : pas implémenté dans tout les nav.
- Source = quoi =
  - une ou plusieurs URL
  - 'none', 'self' (domain courant)
  - 'unsafe-inline' (<script>, javascript:.., <style>, onsomething=...)
  - 'unsafe-eval' (eval())
  - data:

CSP

- où dans la page =
  - default-src
  - script-src (javascript)
  - object-src (<object>, <embed>..), img-src, media-src (<audio> <video>), font-src (@font-face),
  - frame-src (<frame>, <iframe>)
  - connect-src (XHR, WebSocket, EventSource)
  - style-src (<style> et attr style)
  - report-uri : url où envoyer les rapports de violation de la politique

CSP

- CSP définie dans :
  - les Web App (manifest)
  - HTTP header : X-Content-Security-Policy
    - → envoyé par le serveur web
- Exemples
  - default-src 'self' *.mydomain.com
  - default-src 'self'; img-src *; media-src media1.com media2.com; script-src userscripts.example.com
  - Web App « Privileged » : default-src *; script-src 'self'; object-src 'none'; style-src 'self' 'unsafe-inline'
  - Web App « Certified » : default-src *; script-src 'self'; object-src 'none'; style-src 'self'

Application packagée

- Tous les fichiers de l'appli sont dans un zip
- Contient le manifest à sa racine :
  « manifest.webapp »
- A des règles de sécurité par défaut permettant d'utiliser plus ou moins des API sensibles
- Processus de mise à jour spécifique
- Protocole de chargement spécifique :
  app://<uuid>/... (au lieu de http://domain.com/...)

Application packagée

- Trois types d'applications packagées
  - certifiée
    - appli embarquée, a tous les privilèges
  - privilegiées
    - appli tierce, peut utiliser quelques APIs privilégiées
  - simple
    - appli tierce, n'a pas accés aux APIs privilégiées

Installation Web App

API Installation

- Pour proposer l'installation d'une appli web en tant que Web App, il faut un bouton sur une page web du site de l'appli.
- → utilisation d'une API spécifique pour lancer l'installation
- Actuellement deux versions
  - obsolète : Firefox <= 17
  - nouvelle : Firefox > 18 ou 19 ?

API installation (obsolète)

- **navigator.mozApps.install**(url, [install_data], [onsuccess], [onerror])
  - url = url du manifest de la webApp
  - install_data = un objet contenant des données pour l'installateur comme un « receipt » = signature digitale de l'appli
  - onsuccess = un callback appelé en cas de succés
  - onerror = un callback appelé en cas d'erreur, avec un code erreur (« denied », « manifestURLError », « invalidManifest » etc..)
- Install l'appli

API installation (obsolète)

- Pour avoir des infos sur l'appli ou savoir si elle est installée :
  - **navigator.mozApps.amInstalled**(callback) → récupération d'un objet « app » si installé, null sinon
- Objet App :
  - manifest : objet manifest
  - manifestURL, origin, installOrigin, installTime
  - launch() : lance l'appli
  - uninstall() : (obsolète ?)

API installation (new)

- **navigator.mozApps.install**(url, [receipts]);
  - url = url du manifest
  - renvoi un objet request

```
var pending = navigator.mozApps.install(manifestUrl);
pending.onsuccess = function () {
  // Save the App object that is returned
  var appRecord = this.result;
  alert('Installation successful!');
};
pending.onerror = function () {
  // Display the name of the error
  alert('Install failed, error: ' + this.error.name);
};
```

API installation (new)

- Info d'installation
  - var request = **navigator.mozApps.getSelf**();
  - request.result = objet App qui correspond au domaine courant

```
var pending = navigator.mozApps.getSelf();
pending.onsuccess = function () {
  // Save the App object that is returned
  var appRecord = this.result;
  alert('appli installée');
};
pending.onerror = function () {
  // Display the name of the error
  alert('application non installée');
};
```

Web API

Web API

- Ensemble de nouvelles API pour mieux inter-agir avec le matériel et pour pouvoir faire des applis « comme sur le desktop »

Web API

- Battery Status
- Network Information
- Vibration
- WebSMS
- WebTelephony
- Screen Orientation
- Settings
- PowerManagement
- Mobile Connection
- Web Bluetouth
- Wiki Information
- Alarm
- Web Activities
- WebFM
- Archive (zip)
- Ambient light sensor
- Proximity sensor

Exemple WebAPI : battery

- window.navigator.battery
  - propriétés : charging, chargingTime, dischargingTime, level
  - events : chargingchange, levelchange...

```
var battery = navigator.battery || navigator.mozBattery
                                || navigator.webkitBattery;

function updateBatteryStatus() {
  var msg = "Battery status: " + battery.level * 100 + " %"
  if (battery.charging) {
    msg += "\n Battery is charging";
  }
  alert(msg);
}

battery.addEventListener("chargingchange", updateBatteryStatus);
battery.addEventListener("levelchange", updateBatteryStatus);
```

exemple WebAPI : WebSMS

- navigator.mozSms : objet SMSManager
  - getMessage(), getMessages(), markMessageRead(), Send(), ...
  - ondelivered, onreceived, onsent

```
var sms = navigator.mozSms;

// envoi un message
sms.send("123456789", "Hello world!");

// listener pour la reception de messages
sms.onreceived = function (event) {
    console.log(event.message);
};

// creation d'un filtre pour lister
var filter = new SmsFilter()
filter.numbers = 10;
filter.delivery = "received"
var reverse = false
```

```
// recupération d'un SmsRequest sur la liste
var request = sms.getMessages(filter,
                              reverse);

request.onsuccess = function(event) {
  // SmsCursor
  var cursor = event.target.result
  if (cursor.message) {
     var msg = cursor.message
     console.log("envoyé par "+msg.sender
                 +": "+msg.body);
     cursor.continue();
  }
  else
     console.log("fin des message")
}
```

Doc Web API

- Etat des spécifications et implémentations
  - https://wiki.mozilla.org/WebAPI/
- Doc des nouvelles API stables :
  - https://developer.mozilla.org/en-US/docs/WebAPI

Context offline

Offline

- Tout ce qui permet à une appli de fonctionner sans connection :
  - Stockage local : local storage, indexedDB
  - être informer de la connectivité
  - mettre en cache les ressources de l'application → « application cache »

Surveiller la connectivité

- window.navigator.onLine : true si connecté
- évènements « online » et « offline »

```
window.addEventListener("offline", function(e) {alert("offline");})

window.addEventListener("online", function(e) {alert("online");})
```

- Attention : comportement différent selon les plateformes et navigateurs
  - status qui peut basculer en fonction du succés d'une requête
  - status qui peut basculer en fonction de l'état de la connectivité du système (débranchement cable)
- → il est préférable d'avoir sa propre détection (via XHR+settimeout...)

Application cache

- But : mettre en cache les fichiers dont a besoin l'application pour être disponible en offline
- comment :
  - un fichier manifest de cache contenant la liste des fichiers à charger et mettre en cache
  - attribut manifest sur <html> indiquant l'url du manifest
- au rechargement de la page, le browser
  - check la validité du cache du manifest
  - si invalide, recharge tout

Cache manifest

- 3 sections
  - CACHE : liste des fichiers à mettre en cache au premier chargement
  - NETWORK : liste des fichiers à ne pas mettre en cache et nécessite d'être rechargé à chaque fois
  - FALLBACK : liste des ressources à utilisées si d'autres ne sont pas disponibles
- le document doit être servi avec le content-type text/cache-manifest

Cache Manifest

```
CACHE MANIFEST
# v1 2011-08-14
# This is another comment
CACHE:
index.html
cache.html
style.css
image1.png

# Use from network if available
NETWORK:
network.html

# Fallback content
FALLBACK:
/ fallback.html
```

Cache api

- window.applicationCache : objet permettant de gérer le cache
- entres autres :
  - propriété status : indique l'état du cache de la page et de ses fichiers
    - window.applicationCache.UNCACHED, .IDLE, .CHECKING, .DOWNLOADING, .UPDATEREADY, .OBSOLETE
  - update() pour forcer la vérification du cache
  - addEventListener() pour écouter l'un des évènements : checking, updateready, cached, downloading...

Cache : avertissement

- Attention aux effets de bords, avec la combinaison du cache HTTP par exemple
- Forcer la mise à jour d'une application peut être très compliqué
- Ne pas hésiter à se documenter sur le sujet
- Lire par exemple cet article de Jake Archibald http://www.alistapart.com/articles/application-cache-is-a-douchebag/ .
- http://www.html5rocks.com/en/mobile/workingoffthegrid/
- etc..

Firefox OS

Kesako

- Un système d'exploitation pour mobile
- Stack :
  - noyau linux + drivers
  - plateforme Mozilla :
    - Gecko = moteur de rendu + des centaines de composants et API web
    - → affichage et exécution des applis
  - applications exclusivement en HTML5/JS, y compris le bureau

En quoi est-ce différent ?

- Plateforme full web
  - → des centaines de milliers de développeurs prêts pour développer des applis
- proposant des **APIs standardisés** ou **en passe de le devenir**
  - Mozilla ne fait pas son truc dans son coin
  - → ouverture de la technologie, qui sera dispo chez la concurrence
- Pas de « store » imposé → lutte contre les « silos » emprisonnant l'utilisateur
- code 100% open source et dev ouvert
  - → tout fabricant peut l'utiliser sans royalties..
  - … et peut contribuer

Mozilla Manifesto

« promouvoir l'ouverture, l'innovation et les opportunités sur le web »

http://www.mozilla.org/about/manifesto.fr.html

Firefox OS App Days

- 25 janvier de 18 à 20 heures - 26 janvier de 09 à 22 heures dans les locaux de l'EPITA
- Découvrir, Hacker et Célébrer Firefox OS
- S'inscrire : http://firefoxosappdaysparis.eventbrite.com/