Auteur externe

Zeste de savoir

Créez des extensions pour tous les navigateurs

12 août 2019

Table des matières

WebExtensions est une API permettant de créer des extensions qui fonctionnent sur les navigateurs les plus utilisés de nos jours (c'est-à-dire Firefox, Chrome, Opera et Edge). Et cela très simplement grâce aux technologies utilisées partout sur le Web : HTML 5, CSS 3 et JavaScript.

Nous n'allons pas étudier chaque API disponible une à une dans ce tutoriel. Nous allons nous contenter de créer une extension qui nous servira de fil rouge, pour que vous découvriez comment créer, tester et publier vos extensions, ainsi que quelques API comme `alarms`.

Les WebExtensions sont encore en développement dans de nombreux navigateurs. Ainsi, Edge ne les supporte pas encore, mais devrait le faire d'ici le mois de juin 2016. Firefox a déjà commencé à les implémenter depuis la version 42. Vous pouvez suivre l'avancement du développement pour ce dernier ici , et voir les API qui ne sont pas encore implémentées sur le Mozilla Developer Network .

Alors, prêt ?

Prérequis
Connaître le langage JavaScript.
Avoir au moins des notions de HTML et de CSS.
Savoir ce qu'est une extension pour navigateur.
Objectif
Découvrir comment créer et publier des extensions pour plusieurs navigateurs.
Aspects non traités
De nombreuses API comme `cookies` ou `history`.

1. Prérequis pour une WebExtension

Les WebExtensions sont écrites en JavaScript, HTML, CSS et un peu de JSON pour le manifeste. Il ne vous sera pas nécessaire d'avoir des connaissances extrêmement avancées (même si c'est toujours mieux), mais si vous ne les connaissez pas du tout, ~~Google~~ Internet est votre ami.

Pour tester vos extensions, il vous faudra un, ou plutôt des navigateurs, étant donné que les implémentations ne sont pas complètes partout. Je vous conseille donc de tester sur les navigateurs Chrome (ou Opera), Edge et Firefox. Si vous êtes sous Linux ou Mac, Microsoft vous permet de l'utiliser dans une machine virtuelle .

Une fois que c'est bon, il va falloir mettre en place les fichiers de base de notre extension.

Le seul fichier qui soit réellement nécessaire à notre extension est le `manifest.json`, qui sera à la racine de notre extension. Eh oui, c'est tout pour le moment.

Et maintenant, le code !

2. Le manifeste

Le manifeste, c'est ce fichier que vous venez de créer. C'est ici que nous définirons les paramètres de notre extension, comme son nom, sa version ou ses permissions.

Bien, ouvrez votre manifeste, et copiez-y ceci.

```
{
    "name": "Mon extension",
    "version": "0.1",
    "manifest_version": 2,
    "applications": {
        "gecko": {
            "id": "extension@bat.fr"
        }
    }
}
```

Vous avez ici le manifeste minimum de toute extension. Comme vous pouvez le voir, il contient les clés suivantes :

— `name`, le nom de l'extension ;
— `version`, la version de votre extension. Vous remarquerez que c'est une chaîne de caractères, notamment pour avoir des versions alpha ou bêta notées `1.0a` ou `1.0b` ;
— `manifest_version`, la version du votre manifeste. La version 1 est une ancienne spécification qui n'est plus supportée aujourd'hui ;
— `applications`, qui n'est requise et fonctionnelle que dans les navigateurs avec le moteur de rendu Gecko (comme Firefox). C'est un objet qui contient lui-même d'autres objets. Le seul disponible pour le moment est `gecko`. Ce dernier doit contenir le champ `id`, c'est-à-dire un identifiant unique pour votre extension.

> Comment savoir si mon identifiant est vraiment unique ?

Pour éviter tout conflit avec une autre extension, je vous conseille de suivre le schéma suivant :

nomminusculesansespace@*example.org*

Si vous n'avez pas de nom de domaine, utilisez quelque chose comme **pseudo**.*fr* (ou *.com*, *.org*, *.be*, *.ca*, etc.).

> Sans ces champs, votre extension ne fonctionnera pas sur Firefox. Il est cependant possible que Chrome (ou autre) vous dise que ces champs sont incorrects. Vous avez vu, c'est bien standardisé tout ça.

3. Empaquetez et testez votre extension

3.1. Pour Firefox

Firefox supporte depuis longtemps des extensions au format `.xpi`. Ce sont en fait des fichiers `.zip` dont on a changé l'extension. Pour empaqueter votre extension pour Firefox, il vous suffira donc de taper cette commande.

```
# Linux et Mac uniquement.

zip monextension.xpi .
```

Si vous êtes sous Windows et que vous avez le logiciel 7zip, vous pouvez aussi taper ceci.

```
7z a monextension.xpi .
```

Si vous n'avez pas 7zip, vous pouvez toujours faire une copie du dossier contenant votre code, faire un clic droit dessus, sélectionner « Envoyer vers » puis « Dossier compressé ». Nommez l'archive créée en quelque chose comme `monextension.xpi`, et voilà !

Pour tester votre extension, glissez le fichier `.xpi` dans une fenêtre de Firefox. Un petit *pop-over* devrait apparaître pour vous demander confirmation (bien sûr, acceptez).

Si vous avez une erreur, vérifiez que vous avez au moins Firefox 42. Si c'est le cas, mais que ça ne fonctionne tout de même pas, rendez vous sur la page `about:config`, et double-cliquez sur la valeur `xpinstall.signatures.required` pour l'inverser (elle autorise l'installation d'extensions non vérifiées).

Et voilà, vous avez votre extension visible dans Firefox.

FIGURE 3. – Tadam ! Votre extension est installée.

3.2. Pour Chrome et Opera

Pour tester votre extension sous Chrome, il faudra vous rendre sur la page `chrome://extensions` et cocher la case « Mode développeur » tout en haut de la liste. Cliquez ensuite sur « Charger l'extension non empaquetée... » et sélectionnez le dossier où se trouve votre code. Et voilà !

FIGURE 3. – Votre extension dans Chrome.

Pour tester les fois suivantes, vous pouvez simplement cliquer sur « Actualiser ».

> *i* Chrome et Opera ayant le même moteur de rendu, vous n'avez besoin de tester que sur l'un où l'autre. Ici, je vous propose Chrome (n'ayant pas Opera), mais vous n'êtes pas obligé de l'utiliser.

4. Une première extension

Nous allons dès à présent créer notre première extension, histoire de faire connaissance avec quelques API. Cette extension sera « Ne passez pas tout votre temps sur Internet ». Elle va nous permettre de nous familiariser avec les API **`browser action`**, **`storage`**, **`alarms`**, **`notifications`** et **`window`**.

4.1. Cahier des charges

Notre extension devra permettre à l'utilisateur de choisir un temps de navigation, puis de l'alerter quand il l'a dépassé (et même fermer le navigateur s'il persiste). On aura donc un bouton ouvrant une *popup* indiquant le temps restant, et permettant de le configurer. On aura un script en arrière-plan chargé de prévenir l'utilisateur quand il a presque fini le temps imparti (histoire qu'il sauvegarde son travail), puis de fermer ses fenêtres cinq minutes plus tard.

4.2. La *popup* d'affichage du temps

Avant même d'afficher une *popup*, il faut un moyen pour le faire. On va donc créer un petit bouton ~~trop mignon~~ qui s'affichera dans un coin du navigateur, comme ça.

FIGURE 4. – Oh ! Qu'il est mignon ce petit bouton vert !

Pour cela, il suffit d'ajouter quelques lignes à notre manifeste. On va y ajouter un objet `browser_action` (c'est le nom donné au bouton). Il devra contenir au moins une valeur (bah oui, sinon il est complètement inutile) parmi :

- `default_title`, l'infobulle affichée par défaut ;
- `default_popup`, la *popup* qui s'affiche quand on clique sur le bouton ;
- `default_icon`, l'icône par défaut ;
- `icons`, la liste des différentes icônes.

Pour notre propre extension, nous allons tous les utiliser, sauf `default_icon`, parce qu'on va laisser le navigateur choisir. On va donc rajouter ceci dans notre `manifest.json`.

```
    "browser_action": {
        "default_title": "Arrête Internet !",
        "default_popup": "popup.html",
        "icons": {
            "16": "ico16.png",
            "32": "ico32.png",
            "38": "ico38.png",
            "512": "ico512.png"
        }
    }
```

Vous remarquerez que j'ai mis plusieurs icônes. Il y a celles de 16×16 pixels et de 32×32 pixels qui sont des tailles classiques pour des icônes. J'ai aussi mis une icône de 38 pixels à cause de Chrome, qui recommande cette taille pour ne pas perdre de place[1].

Vous pouvez maintenant tester votre extension comme nous l'avons vu précédemment. Désormais, vous avez un bouton, et lorsque vous cliquez dessus...

FIGURE 4. – On dirait que Firefox a du mal à trouver nos fichiers...

Vous avez deviné d'où vient le problème, je pense. C'est tout simplement qu'on a indiqué une page `popup.html` qui n'existe pas. C'est pareil pour l'icône : bien qu'on en ait spécifié dans la clé `icons` du manifeste, on a la pièce de puzzle (l'icône par défaut) à la place. On va immédiatement remédier à ça en créant les fichiers demandés. Et si vous êtes horrifiés par GIMP et compagnie, je vous ai fait de belles icônes[2].

Contenu masqué n°1

Par contre, pour la *popup*, je ne fais pas tout pour vous ! Vous allez vous débrouiller et créer une petite page qui affichera l'heure (par défaut, 00 :00 :00, on changera ça dans le code), ainsi qu'un bouton pour régler le tout. On a un code très simple, qui peut ressembler à ceci.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Temps restant</title>
    </head>
    <body>
        <button><img src="settings.png" alt="Paramètres"></button>
        <p>Il vous reste :</p>
        <p id="temps">00:00:00</p> <!-- J'ai mis un ID pour pouvoir
            le modifier facilement en JavaScript. -->
    </body>
</html>
```

Vous pouvez rajouter un peu de CSS, histoire de rendre le tout plus beau (pensez à le lier avec votre HTML).

```css
body {
    text-align: center;
    font-family: 'Open Sans', 'Segoe UI Light', Arial;
 }

p {
    margin-top: 30px;
    color: gray;
}

#temps {
    color: red;
    font-family: 'Digital-7', monospace;
    font-size: 64px;
```

```
}

button {
    float: right;
    border: none;
    background: transparent;
}

button:hover {
    background: rgba(0, 0, 0, 0.2)
}

img {
    width: 20px;
    height: 20px;
}
```

J'utilise les polices d'écriture Segoe UI Light et Digital-7, mais vous pouvez tout à fait en utiliser d'autres, car elles ne sont pas libres de droits, et ne sont peut-être pas installées sur votre système.

Si vous testez votre extension, vous devriez avoir quelque chose ressemblant à peu près à ça.

FIGURE 4. – C'est zouli !

Maintenant, on va rendre tout cela un peu plus dynamique. Pour commencer, on va stocker le temps que l'utilisateur a le droit de passer de manière permanente (histoire qu'on n'ait pas à le reconfigurer à chaque session). Pour cela, on va avoir besoin de créer ce qu'on appelle un ***background script*** ou script en arrière-plan dans la langue de Molière. Ce script va s'exécuter dès le lancement du navigateur. On va donc y mettre une grande partie de notre code. Créez un fichier, par exemple `background.js`. Bien, maintenant il faut spécifier à notre navigateur de lancer ce fichier en même temps que l'extension. Pour cela, il faut aller dans... le manifeste ! On va y rajouter ceci.

```
"background": {
    "scripts": [ "background.js" ],
    "persistent": true
}
```

On vient d'ajouter un objet `background` contenant :

— `scripts`, la liste de nos scripts en arrière-plan ;
— `persistent`, qui indique si ces scripts doivent tourner en permanence, ou s'ils n'ont besoin de s'activer qu'occasionnellement. Ici, on va très souvent mettre à jour le temps, donc on laisse notre script tourner en permanence, même si cela prend un peu plus de mémoire.

Il va falloir ajouter autre chose dans notre manifeste : des permissions. Pour le moment nous allons seulement permettre à notre extension d'accéder à l'API `storage`. On écrit donc ceci à la suite de notre manifeste.

```
"permissions": [
    "storage"
]
```

Maintenant rendons-nous dans le fichier `background.js`, et voyons comment charger et enregistrer le temps imparti via cette fameuse API `storage`. Voici le script.

```
// Par défaut on a un temps de 1h.
var tempsRestant = 3600;

// Si l'utilisateur a choisi autre chose, on prends cette valeur.
var tempsUtilisateur = chrome.storage.local.get('temps',
    function(){});

if (tempsUtilisateur) tempsRestant = tempsUtilisateur;
```

Si on passe en argument une fonction vide, c'est parce que c'est obligatoire.

Maintenant, il va falloir mettre à jour l'affichage du temps restant chaque seconde. Pour cela, nous allons utiliser l'API `alarms`, qui permet d'exécuter une action après un certain temps. Il va tout d'abord falloir l'ajouter aux permissions de notre manifeste. Ensuite, dans `background.js`, on va créer une nouvelle alarme.

```
chrome.alarms.create ('chrono', {
    delayInMinutes : 0.0,
    periodInMinutes : 1 / 60
});
```

Ici, le paramètre `delayInMinutes` indique le temps avant que l'alarme ne s'enclenche. S'il est nul, c'est tout simplement parce que le suivant, `periodInMinutes`, précise qu'il faut faire tourner l'alarme en boucle sur un cycle de un 60^{e} de minute (une seconde quoi). Donc si on rajoutait en plus un délai, on n'aurait pas une mise à jour chaque seconde.

Vous avez sûrement vu que, pour le moment, on a beau avoir créé une alarme, elle ne déclenche rien. Pour la lier à des actions, il suffit de s'abonner à l'événement `chrome.alarms.onAlarm` de cette façon.

```
chrome.alarms.onAlarm.addListener (function() {
    // C'est là qu'on fera la mise à jour de l'interface.
});
```

Enfin, pour mettre à jour notre interface, on va devoir trouver un moyen de communiquer entre notre script d'arrière-plan et notre *popup*. Heureusement pour nous, on a l'API `runtime` qui permet notamment d'envoyer des messages entre différents scripts (on n'a pas besoin de l'ajouter aux permissions). On va donc utiliser la fonction `chrome.runtime.sendMessage` de cette façon.

```
chrome.runtime.sendMessage({temps : tempsRestant});
```

Pensez aussi à décrémenter `tempsRestant`.

Le message envoyé est toujours un objet. On a donc choisi un objet avec une propriété `temps` indiquant le temps restant à la *popup*. Pour le moment, on envoie un message dans le vide. On va gérer sa réception ainsi, dans `popup.js`.

```
chrome.runtime.onMessage.addListener(function (requete, envoyeur,
    repondre){
    document.getElementById('temps').innerHTML = requete.temps;
});
```

Vous remarquerez que l'objet `requete` est celui qu'on a envoyé via `sendMessage` quelques instant plus tôt. Avec ce code on obtient quelque chose comme ça.

: Notre *popup* s'actualise bel et bien.

Beurk ! C'est quoi cet affichage du temps. Je voudrais quelque chose de beau comme « 00 :59 :48 ».

Alors améliorons notre code pour cela. On va ajouter une fonction `beauTemps`, qui va formater le temps comme on le veut.

```
function beauTemps (temps) {
    var heures = Math.floor(temps / 3600);
    var minutes = Math.floor(temps / 60) % 60;
    var secondes = temps % 60;
    return `${heures}:${minutes}:${secondes}`;
}
```

On retourne un ***template string***, une fonctionnalité de JavaScript que vous n'avez peut-être jamais rencontrée, puisqu'elle a été ajoutée à la dernière spécification du langage. Il permet notamment d'insérer facilement des variables. Si vous voulez en savoir plus, regardez cet article (en anglais).

Il nous suffit maintenant de remplacer notre ancien code par celui-ci.

```
chrome.runtime.onMessage.addListener(function (requete, envoyeur,
    repondre){
    document.getElementById('temps').innerHTML =
        beauTemps(requete.temps);
});
```

Voilà, c'est déjà mieux. Seulement, on a beau avoir un compte à rebours, l'utilisateur peut très bien l'ignorer. On va donc devoir implémenter la notification de prévention cinq minutes avant la fin et la fermeture de la fenêtre. Pour commencer, voyons voir comment fermer la fenêtre. Pour ça, on vérifie si le temps restant est égal à zéro. Et si c'est le cas, on ferme toutes les fenêtres. Le code correspondant est celui-ci.

```
if (tempsRestant == 0) {
   // On obtient toutes les fenêtres.
   chrome.windows.getAll(function(fenetres) {
        // On les passe une à une.
        fenetres.forEach(function(fenetre) {
            // Et on les ferme.
            chrome.windows.remove(fenetre.id);
       });
    });
}
```

Bien que la fonction `chrome.windows.remove` soit apparemment implémentée dans Firefox, elle ne semble pas fonctionner. On pourrait cependant la reproduire en fermant tous les onglets un à un.

Vous pouvez tester (avec un temps plus court, peut-être), votre fenêtre se ferme toute seule ! Mais comme ça peut être embêtant de perdre tout son travail d'un coup, on va prévenir l'utilisateur via une petite notification cinq minutes avant de fermer. On va donc devoir ajouter la permission `notifications` dans notre manifeste. Après, on teste le temps restant, et on envoie la notification s'il ne reste que cinq minutes.

```
if (tempsRestant == 0) {
    // …
} else if (tempsRestant == 5 * 60) {
    chrome.notifications.create({
        type: "basic",
        iconUrl: 'ico512.png',
        title: 'Enregistrez votre travail',
        message:
            'Il ne vous reste que 5 minutes de surf, dépêchez-vous !'
    });
}
```

Et lorsqu'on teste, on voit bien notre notification s'afficher.

FIGURE 4. – La notification (c'est un peu long, attendez).

Et voilà, on a créé une petite extension, qui ne fait même pas deux cents lignes de code, mais qui nous a permis de voir comment fonctionnent certaines API. Cependant, elle peut encore être améliorée. Voici quelques pistes de choses à changer.

— Fermer la fenêtre sous Firefox, en fermant tous les onglets un à un.
— Mettre en place des paramètres qui permettraient à l'utilisateur de choisir un temps.
— Permettre un mode travail, où le compte à rebours s'arrête, mais où certains sites sont bloqués (YouTube, etc.).

Si vous avez besoin d'aide, vous pouvez retrouver la documentation en anglais ici (une version française devrait arriver sur le MDN bientôt je pense). J'ai aussi posté le code de l'extension que nous venons de faire sur Framagit , si jamais vous n'avez pas compris quelque chose (signalez-le aussi dans les commentaires, histoire que j'améliore le tout).

1. Les boutons font 19 pixels dans Chrome, mais on met 38 pixels pour une meilleure qualité d'image.
2. Moi non plus, je ne suis pas un grand maître de l'art numérique.

5. Publier votre extension

Certes, avoir une extension, c'est bien beau, mais si personne ne peut en profiter, c'est un peu inutile. Nous allons donc voir comment les publier dans les *stores* officiels.

5.1. Firefox

Les WebExtensions ne sont pas encore acceptées sur le *store* de Mozilla. En attendant que ce soit le cas, vous pouvez toujours faire télécharger votre fichier `.xpi` à vos utilisateurs, et leur faire faire une installation manuelle.

5.2. Chrome

Si vous n'avez pas de compte développeur Google, il faudra vous en créer un pour 5€.

Rendez-vous sur le Chrome Web Store et connectez-vous ou inscrivez-vous. Cliquez ensuite sur le petit engrenage en haut à droite, et choisissez l'option « Tableau de bord du développeur ». Cliquez alors sur le bouton « Ajouter un nouvel article ». Suivez ensuite les instructions. Votre extension devrait être publiée après validation. Bravo !

C'est ici que ce tutoriel s'achève : si vous avez des questions, les commentaires et les forums sont là. J'espère que bientôt, de nombreuses personnes auront installé votre extension !

Contenu masqué

Contenu masqué n°1

Retourner au texte.

Liste des abréviations

MDN Mozilla Developer Network. 11