Créer une extension Chrome

Avez-vous déjà ouvert trop d'onglets de navigateur et souhaité un outil pour les gérer automatiquement ? Dans cet article de blog, nous allons passer en revue la création d'une extension Chrome appelée **"Tabs Killer"**, qui ferme automatiquement les onglets les plus anciens lorsque le nombre d'onglets dépasse une limite définie par l'utilisateur. Je vais décomposer le code, expliquer son fonctionnement et fournir des informations pour vous aider à créer votre propre extension Chrome.

À la fin de cet article, vous comprendrez la structure d'une extension Chrome, comment travailler avec l' API Chrome et comment créer une interface popup avec des paramètres.

Que fait "Tabs Killer"?

"Tabs Killer"est une extension Chrome qui : - Surveille le nombre d'onglets ouverts. - Permet aux utilisateurs de définir une limite maximale d'onglets. - Ferme automatiquement les onglets les plus anciens lorsque la limite est dépassée. - Fournit une fonctionnalité de liste blanche pour protéger certains onglets (par exemple, en fonction de motifs d'URL) de la fermeture.

L'extension comprend une interface popup pour configurer les paramètres et un script en arrière-plan pour gérer les onglets.

Structure du projet

Voici la structure des fichiers de l'extension "Tabs Killer":

```
tabs-killer/
  manifest.json         # Configuration de l'extension
  popup.html            # Interface utilisateur popup
  popup.js             # Logique popup
  background.html       # Page en arrière-plan
  app.build.js          # Logique principale de l'application (supposée)
  js/
     lib/              # Bibliothèques externes (jQuery, Underscore, Bootstrap, RequireJS)
     tabmanager.js     # Logique de gestion des onglets (supposée)
     settings.js       # Gestion des paramètres (supposée)
  css/
     popup.css         # Styles popup
```

```
img/
    icon16.png        # Icône 16x16
    icon48.png        # Icône 48x48
    icon128.png       # Icône 128x128
```

Étape 1 : Le fichier manifeste (`manifest.json`)

Le fichier `manifest.json` est le cœur de toute extension Chrome. Il définit les métadonnées, les autorisations et les composants clés.

```json
{
  "manifest_version": 2,
  "name": "Tabs Killer",
  "description": "Ferme automatiquement les onglets les plus anciens lorsque les onglets sont trop nombreux.",
  "version": "1.0",
  "browser_action": {
    "default_icon": "img/icon128.png",
    "default_popup": "popup.html"
  },
  "icons": {
    "128": "img/icon128.png",
    "48": "img/icon48.png",
    "16": "img/icon16.png"
  },
  "background": {
    "page": "background.html"
  },
  "permissions": [
    "tabs",
    "storage"
  ],
  "content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'"
}
```

Explication :

- **`manifest_version`** : Doit être `2` (Chrome a déprécié la version 1).
- **`name`, `description`, `version`** : Métadonnées de base.

- `browser_action` : Définit l'icône de la barre d'outils de l'extension et le popup (`popup.html`).
- `icons` : Icônes de différentes tailles (utilisées dans le Chrome Web Store et la barre d'outils).
- `background` : Spécifie une page en arrière-plan (`background.html`) qui s'exécute en permanence.
- `permissions` : Demande l'accès à l'API `tabs` (pour gérer les onglets) et à l'API `storage` (pour enregistrer les paramètres).
- `content_security_policy` : Autorise `unsafe-eval` pour les bibliothèques comme RequireJS (utilisez avec prudence en production).

Étape 2 : L'interface utilisateur popup (`popup.html`)

Le popup apparaît lorsque l'utilisateur clique sur l'icône de l'extension. Il utilise Bootstrap pour le style et comprend une interface à onglets avec une section "Options".

```
<!doctype html>
<html>
<head>
  <title>Popup de l'extension Tabs Killer</title>
  <link rel="stylesheet" href="js/lib/bootstrap/css/bootstrap.css" type="text/css"/>
  <link rel="stylesheet" href="css/popup.css"/>
  <script src="js/lib/jquery.min.js"></script>
  <script src="js/lib/underscore.js"></script>
  <script src="js/lib/bootstrap/js/bootstrap.min.js"></script>
  <script src="js/lib/bootstrap/js/bootstrap-tab.js"></script>
  <script src="js/lib/require.js"></script>
  <script src="app.build.js"></script>
  <script src="popup.js"></script>
</head>
<body>
  <ul class="nav nav-tabs">
    <li><a href="#tabOptions" target="#tabOptions" data-toggle="tab">Options</a></li>
  </ul>
  <div class="tab-content">
    <div class="tab-pane active" id="tabOptions">
      <form class="well">
        <fieldset>
          <legend>Paramètres</legend>
          <p>
            <label for="maxTabs">Nombre maximum d'onglets à conserver</label>
```

```
            <input type="text" id="maxTabs" class="span1" name="maxTabs"> onglets
          </p>
        </fieldset>
        <div id="status" class="alert alert-success invisible"></div>
        <fieldset>
          <legend>Auto-verrouillage</legend>
          <label for="white-list-input">onglet avec une URL contenant la chaîne :</label>
          <input type="text" id="white-list-input"/>
          <button class="btn-mini add-on" disabled id="white-list-add">Ajouter</button>
          <table class="table table-bordered table-striped" id="white-list">
            <thead>
              <tr>
                <th>Motif d'URL</th>
                <th></th>
              </tr>
            </thead>
            <tbody></tbody>
          </table>
        </fieldset>
      </form>
    </div>
  </div>
  <script type="text/html" id="url-item-template">
    <tr>
      <td><%=url%></td>
      <td><a class="deleteLink" href="#">Supprimer</a></td>
    </tr>
  </script>
</body>
</html>
```

Explication :

- **Bibliothèques** : Utilise jQuery, Underscore, Bootstrap et RequireJS pour la fonctionnalité et le style.
- **Éléments de l'interface utilisateur** :
  - Une entrée de texte (`#maxTabs`) pour définir le nombre maximum d'onglets.
  - Une entrée de liste blanche (`#white-list-input`) et un bouton "Ajouter"(`#white-list-add`) pour protéger des URL spécifiques.
  - Un tableau (`#white-list`) pour afficher les motifs de liste blanche avec un lien "Supprimer".
  - Un message d'état (`#status`) pour afficher les retours d'information de sauvegarde.

- **Modèle** : Un modèle Underscore.js (`#url-item-template`) génère dynamiquement les lignes de tableau.

Étape 3 : Logique du popup (`popup.js`)

Ce script gère l'interactivité du popup, comme l'enregistrement des paramètres et la gestion de la liste blanche.

```
require([], function () {
  var GlobalObject = chrome.extension.getBackgroundPage().GlobalObject;

  Popup = {};
  Popup.optionsTab = {};

  Popup.optionsTab.init = function (context) {
    function onBlurInput() {
      var key = this.id;
      Popup.optionsTab.saveOption(key, $(this).val());
    }
    $('#maxTabs').keyup(_.debounce(onBlurInput, 200));
    Popup.optionsTab.loadOptions();
  };

  Popup.optionsTab.loadOptions = function () {
    $('#maxTabs').val(GlobalObject.settings.get('maxTabs'));
    var whiteList = GlobalObject.settings.get('whiteList');
    Popup.optionsTab.buildWhiteListTable(whiteList);

    var $whiteListInput = $('#white-list-input');
    var $whiteListAdd = $('#white-list-add');

    var isValid = function (pattern) {
      return /\S/.test(pattern);
    };

    $whiteListInput.on('input', function () {
      if (isValid($whiteListInput.val())) {
        $whiteListAdd.removeAttr('disabled');
      } else {
```

```
      $whiteListAdd.attr('disabled', 'disabled');
    }
  });

  $whiteListAdd.click(function () {
    if (!isValid($whiteListInput.val())) return;
    whiteList.push($whiteListInput.val());
    $whiteListInput.val('').trigger('input').focus();
    Popup.optionsTab.saveOption('whiteList', whiteList);
    Popup.optionsTab.buildWhiteListTable(whiteList);
  });
};

Popup.optionsTab.saveOption = function (key, value, hideStatus) {
  if (!hideStatus) $('#status').html('');
  GlobalObject.settings.set(key, value);
  if (!hideStatus) {
    $('#status').removeClass('invisible').css('opacity', '100')
      .html('Enregistrement...').delay(50).animate({opacity: 0});
  }
};

Popup.optionsTab.buildWhiteListTable = function (whiteList) {
  var urlItemTemplate = _.template($("#url-item-template").html());
  var $wlTable = $('table#white-list tbody');
  $wlTable.html('');
  for (var i = 0; i < whiteList.length; i++) {
    var $tr = $(urlItemTemplate({url: whiteList[i]}));
    var $deleteLink = $tr.find('a.deleteLink').parent();
    $deleteLink.click(function () {
      whiteList.splice(whiteList.indexOf($(this).data('pattern')), 1);
      Popup.optionsTab.saveOption('whiteList', whiteList, true);
      Popup.optionsTab.buildWhiteListTable(whiteList);
    }).data('pattern', whiteList[i]);
    $wlTable.append($tr);
  }
};

$(document).ready(function () {
  $('a[data-toggle="tab"]').on('show', function (e) {
```

```
      var tabId = e.target.hash;
      if (tabId === '#tabOptions') {
        Popup.optionsTab.init($('div#tabOptions'));
      }
    });
    $('a[href="#tabOptions"]').click();
  });
});
```

Explication :

- **Initialisation** : Se connecte à la page en arrière-plan `GlobalObject` pour les paramètres et la gestion des onglets.
- `init` : Configure les écouteurs d'événements, comme l'entrée débloquée pour `#maxTabs`.
- `loadOptions` : Charge les paramètres enregistrés (max onglets et liste blanche) et remplit l'interface utilisateur.
- `saveOption` : Enregistre les paramètres dans `GlobalObject.settings` et affiche une animation "Enregistrement...".
- `buildWhiteListTable` : Construit dynamiquement le tableau de liste blanche avec la fonctionnalité de suppression.
- **Écouteurs d'événements** : Gère la validation de l'entrée, l'ajout d'entrées de liste blanche et la commutation d'onglets.

Étape 4 : Logique en arrière-plan (`background.html` et scripts supposés)

La page en arrière-plan (`background.html`) s'exécute en permanence et charge la logique principale.

```
// background.js (supposé, basé sur l'extrait fourni)
GlobalObject = {};

require(['tabmanager', 'settings'], function (tabmanager, settings) {
  var startup = function () {
    GlobalObject.settings = settings;
    GlobalObject.tabmanager = tabmanager;
    settings.init();
    tabmanager.init();
  };
  startup();
});
```

Hypothèses :

- `settings.js` : Gère le stockage (par exemple, `chrome.storage`) pour les paramètres comme `maxTabs` et `whiteList`.
- `tabmanager.js` : Utilise l'API `tabs` pour surveiller et fermer les onglets en fonction de la limite `maxTabs` et de la `whiteList`.

Exemple `tabmanager.js` (hypothétique) :

```
var tabmanager = {
  init: function () {
    chrome.tabs.onCreated.addListener(this.checkTabCount);
  },
  checkTabCount: function () {
    chrome.tabs.query({}, function (tabs) {
      var maxTabs = GlobalObject.settings.get('maxTabs') || 10;
      var whiteList = GlobalObject.settings.get('whiteList') || [];
      if (tabs.length > maxTabs) {
        var tabsToRemove = tabs.filter(tab => !whiteList.some(pattern => tab.url.includes(pattern)));
        chrome.tabs.remove(tabsToRemove[0].id); // Ferme l'onglet le plus ancien
      }
    });
  }
};
```

Comment tester l'extension

1. Ouvrez Chrome et allez à `chrome://extensions/`.
2. Activez le "Mode développeur"(bouton bascule en haut à droite).
3. Cliquez sur "Charger l'extension non empaquetée"et sélectionnez le dossier `tabs-killer`.
4. Cliquez sur l'icône de l'extension pour ouvrir le popup et tester les paramètres.

Conseils pour écrire votre propre extension Chrome

1. **Commencez petit** : Commencez avec un manifeste simple et un popup ou un script en arrière-plan.
2. **Utilisez les API Chrome** : Utilisez `chrome.tabs`, `chrome.storage` et autres selon les besoins.

3. **Débogage** : Utilisez `console.log` et les outils de développement de Chrome (clic droit sur le popup > Inspecter).
4. **Sécurité** : Évitez `unsafe-eval` en production ; utilisez des politiques de sécurité de contenu plus strictes.
5. **Bibliothèques d'interface utilisateur** : Bootstrap et jQuery simplifient le développement de l'interface utilisateur mais gardez l'extension légère.

Conclusion

"Tabs Killer"montre comment combiner une interface utilisateur popup, une logique en arrière-plan et les API Chrome pour créer une extension fonctionnelle. Avec cette base, vous pouvez la personnaliser davantage - ajouter des notifications, affiner la logique de fermeture des onglets ou améliorer l'interface utilisateur.

N'hésitez pas à expérimenter avec le code et à partager vos propres idées d'extension Chrome ! Bonne programmation !