Crear una extensión de Chrome

¿Alguna vez has abierto demasiadas pestañas del navegador y deseado una herramienta que las gestionara automáticamente? En esta entrada de blog, te guiaré a través de la creación de una extensión de Chrome llamada **“Tabs Killer”**, que cierra automáticamente las pestañas más antiguas cuando el número de pestañas supera un límite definido por el usuario. Desglosaré el código, explicaré cómo funciona y proporcionaré conocimientos para ayudarte a construir tu propia extensión de Chrome.

Al final de esta entrada, entenderás la estructura de una extensión de Chrome, cómo trabajar con la API de Chrome y cómo crear una interfaz emergente con configuraciones.

¿Qué hace “Tabs Killer”?

“Tabs Killer”es una extensión de Chrome que: - Monitorea el número de pestañas abiertas. - Permite a los usuarios establecer un límite máximo de pestañas. - Cierra automáticamente las pestañas más antiguas cuando se supera el límite. - Proporciona una característica de lista blanca para proteger pestañas específicas (por ejemplo, basadas en patrones de URL) de ser cerradas.

La extensión incluye una interfaz emergente para configurar ajustes y un script de fondo para manejar la gestión de pestañas.

Estructura del Proyecto

Aquí está la estructura de archivos de la extensión “Tabs Killer”:

```
tabs-killer/
  manifest.json         # Configuración de la extensión
  popup.html            # UI emergente
  popup.js             # Lógica de la emergente
  background.html       # Página de fondo
  app.build.js          # Lógica principal de la aplicación (asumida)
  js/
     lib/              # Bibliotecas externas (jQuery, Underscore, Bootstrap, RequireJS)
     tabmanager.js     # Lógica de gestión de pestañas (asumida)
     settings.js       # Gestión de configuraciones (asumida)
  css/
     popup.css         # Estilos de la emergente
```

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

Paso 1: El Archivo Manifest (`manifest.json`)

El archivo `manifest.json` es el corazón de cualquier extensión de Chrome. Define metadatos, permisos y componentes clave.

```
{
  "manifest_version": 2,
  "name": "Tabs Killer",
  "description": "Cierra automáticamente las pestañas más antiguas cuando hay demasiadas.",
  "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'"
}
```

Explicación:

- **`manifest_version`**: Debe ser `2` (Chrome descontinuó la versión 1).
- **`name`, `description`, `version`**: Metadatos básicos.

- `browser_action`: Define el icono de la barra de herramientas y la emergente (`popup.html`) de la extensión.
- `icons`: Iconos de diferentes tamaños (usados en la Chrome Web Store y la barra de herramientas).
- `background`: Especifica una página de fondo (`background.html`) que se ejecuta persistentemente.
- `permissions`: Solicita acceso a la API `tabs` (para gestionar pestañas) y la API `storage` (para guardar configuraciones).
- `content_security_policy`: Permite `unsafe-eval` para bibliotecas como RequireJS (usar con precaución en producción).

Paso 2: La UI Emergente (`popup.html`)

La emergente aparece cuando el usuario hace clic en el icono de la extensión. Utiliza Bootstrap para el estilo e incluye una interfaz con pestañas con una sección "Opciones".

```
<!doctype html>
<html>
<head>
  <title>Emergente de la extensión 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">Opciones</a></li>
  </ul>
  <div class="tab-content">
    <div class="tab-pane active" id="tabOptions">
      <form class="well">
        <fieldset>
          <legend>Configuraciones</legend>
          <p>
            <label for="maxTabs">Máximo de pestañas para mantener</label>
```

```
            <input type="text" id="maxTabs" class="span1" name="maxTabs"> pestañas
          </p>
        </fieldset>
        <div id="status" class="alert alert-success invisible"></div>
        <fieldset>
          <legend>Bloqueo Automático</legend>
          <label for="white-list-input">pestaña con URL que contiene la cadena:</label>
          <input type="text" id="white-list-input"/>
          <button class="btn-mini add-on" disabled id="white-list-add">Agregar</button>
          <table class="table table-bordered table-striped" id="white-list">
            <thead>
              <tr>
                <th>Patrón de 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="#">Eliminar</a></td>
    </tr>
  </script>
</body>
</html>
```

Explicación:

- **Bibliotecas**: Usa jQuery, Underscore, Bootstrap y RequireJS para funcionalidad y estilo.
- **Elementos de la UI**:
  - Una entrada de texto (`#maxTabs`) para establecer el número máximo de pestañas.
  - Una entrada de lista blanca (`#white-list-input`) y un botón “Agregar”(`#white-list-add`) para proteger URLs específicas.
  - Una tabla (`#white-list`) para mostrar patrones de lista blanca con un enlace “Eliminar”.
  - Un mensaje de estado (`#status`) para mostrar retroalimentación de guardado.

- **Plantilla**: Una plantilla de Underscore.js (`#url-item-template`) genera dinámicamente filas de la tabla.

Paso 3: Lógica de la Emergente (`popup.js`)

Este script maneja la interactividad de la emergente, como guardar configuraciones y gestionar la lista blanca.

```
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('Guardando...').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();
  });
});
```

Explicación:

- **Inicialización**: Se conecta a la página de fondo `GlobalObject` para configuraciones y gestión de pestañas.
- `init`: Configura oyentes de eventos, como entrada con retardo para `#maxTabs`.
- `loadOptions`: Carga configuraciones guardadas (máximo de pestañas y lista blanca) y llena la UI.
- `saveOption`: Guarda configuraciones en `GlobalObject.settings` y muestra una animación "Guardando...".
- `buildWhiteListTable`: Construye dinámicamente la tabla de lista blanca con funcionalidad de eliminación.
- **Oyentes de Eventos**: Maneja validación de entrada, agregar entradas de lista blanca y cambiar de pestaña.

Paso 4: Lógica de Fondo (`background.html` y Scripts Asumidos)

La página de fondo (`background.html`) se ejecuta persistentemente y carga la lógica principal.

```
// background.js (asumido, basado en el fragmento proporcionado)
GlobalObject = {};

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

Asunciones:

- `settings.js`: Maneja el almacenamiento (por ejemplo, `chrome.storage`) para configuraciones como `maxTabs` y `whiteList`.
- `tabmanager.js`: Usa la API `tabs` para monitorear y cerrar pestañas basadas en el límite `maxTabs` y `whiteList`.

Ejemplo `tabmanager.js` (hipotético):

```
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); // Eliminar la pestaña más antigua
      }
    });
  }
};
```

Cómo Probar la Extensión

1. Abre Chrome y ve a `chrome://extensions/`.
2. Activa el “Modo desarrollador”(interruptor en la esquina superior derecha).
3. Haz clic en “Cargar sin empaquetar”y selecciona la carpeta `tabs-killer`.
4. Haz clic en el icono de la extensión para abrir la emergente y probar las configuraciones.

Consejos para Escribir tu Propia Extensión de Chrome

1. **Empieza Pequeño**: Comienza con un manifiesto simple y una emergente o script de fondo.
2. **Usa APIs de Chrome**: Aprovecha `chrome.tabs`, `chrome.storage` y otros según sea necesario.

3. **Depuración**: Usa `console.log` y las herramientas de desarrollo de Chrome (haz clic derecho en la emergente > Inspeccionar).
4. **Seguridad**: Evita `unsafe-eval` en producción; usa políticas de seguridad de contenido más estrictas.
5. **Bibliotecas de UI**: Bootstrap y jQuery simplifican el desarrollo de la UI, pero mantén la extensión ligera.

Conclusión

"Tabs Killer"demuestra cómo combinar una UI emergente, lógica de fondo y APIs de Chrome para crear una extensión funcional. Con esta base, puedes personalizarla aún más: agregar notificaciones, refinar la lógica de cierre de pestañas o mejorar la UI.

¡No dudes en experimentar con el código y compartir tus propias ideas de extensión de Chrome! ¡Feliz codificación!