

Chrome-Erweiterung erstellen

Hast du dich jemals gefragt, wie es wäre, wenn du zu viele Browser-Tabs geöffnet hast und ein Tool hättest, das sie automatisch verwaltet? In diesem Blogbeitrag werden wir die Erstellung einer Chrome-Erweiterung namens **“Tabs Killer”** durchgehen, die automatisch die ältesten Tabs schließt, wenn die Anzahl der Tabs eine benutzerdefinierte Grenze überschreitet. Ich werde den Code aufschlüsseln, erklären, wie er funktioniert, und Einblicke geben, um dir zu helfen, deine eigene Chrome-Erweiterung zu erstellen.

Bis zum Ende dieses Beitrags wirst du die Struktur einer Chrome-Erweiterung verstehen, wie man mit der Chrome-API arbeitet und wie man eine Popup-Oberfläche mit Einstellungen erstellt.

Was macht “Tabs Killer”?

“Tabs Killer” ist eine Chrome-Erweiterung, die:

- Die Anzahl der geöffneten Tabs überwacht.
- Benutzern ermöglicht, eine maximale Tab-Grenze festzulegen.
- Automatisch die ältesten Tabs schließt, wenn die Grenze überschritten wird.
- Ein Whitelist-Feature bietet, um bestimmte Tabs (z. B. basierend auf URL-Mustern) vor dem Schließen zu schützen.

Die Erweiterung enthält eine Popup-Oberfläche zur Konfiguration der Einstellungen und ein Hintergrundskript zur Verwaltung der Tabs.

Projektstruktur

Hier ist die Dateistruktur der “Tabs Killer”-Erweiterung:

```
tabs-killer/
  manifest.json          # Erweiterungs-Konfiguration
  popup.html              # Popup-Benutzeroberfläche
  popup.js                # Popup-Logik
  background.html         # Hintergrundseite
  app.build.js            # Hauptanwendungslogik (angenommen)
  js/
    lib/                  # Externe Bibliotheken (jQuery, Underscore, Bootstrap, RequireJS)
    tabmanager.js          # Tab-Verwaltungslogik (angenommen)
    settings.js            # Einstellungsverwaltung (angenommen)
  css/
    popup.css              # Popup-Stile
```

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

Schritt 1: Die Manifest-Datei (`manifest.json`)

Die `manifest.json`-Datei ist das Herzstück jeder Chrome-Erweiterung. Sie definiert Metadaten, Berechtigungen und Schlüsselkomponenten.

```
{
  "manifest_version": 2,
  "name": "Tabs Killer",
  "description": "Schließt automatisch die ältesten Tabs, wenn zu viele Tabs geöffnet sind.",
  "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'"
}
```

Erklärung:

- `manifest_version`: Muss 2 sein (Chrome hat Version 1 veraltet).
- `name, description, version`: Grundlegende Metadaten.

- `browser_action`: Definiert das Symbol der Erweiterung und das Popup (`popup.html`).
 - `icons`: Symbole in verschiedenen Größen (im Chrome Web Store und in der Symbolleiste verwendet).
 - `background`: Gibt eine Hintergrundseite (`background.html`) an, die dauerhaft läuft.
 - `permissions`: Fordert Zugriff auf die `tabs`-API (zur Verwaltung der Tabs) und die `storage`-API (zum Speichern der Einstellungen) an.
 - `content_security_policy`: Erlaubt `unsafe-eval` für Bibliotheken wie RequireJS (in der Produktion vorsichtig verwenden).
-

Schritt 2: Die Popup-Oberfläche (`popup.html`)

Das Popup erscheint, wenn der Benutzer auf das Symbol der Erweiterung klickt. Es verwendet Bootstrap für die Stilgestaltung und enthält eine Registerkarten-Oberfläche mit einem “Optionen”-Bereich.

```
<!doctype html>
<html>
<head>
  <title>Popup der Tabs Killer-Erweiterung</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">Optionen</a></li>
  </ul>
  <div class="tab-content">
    <div class="tab-pane active" id="tabOptions">
      <form class="well">
        <fieldset>
          <legend>Einstellungen</legend>
          <p>
            <label for="maxTabs">Maximale Anzahl der Tabs</label>

```

```

        <input type="text" id="maxTabs" class="span1" name="maxTabs"> Tabs
    </p>
</fieldset>

<div id="status" class="alert alert-success invisible"></div>

<fieldset>

    <legend>Auto-Lock</legend>

    <label for="white-list-input">Tab mit URL, die den String enthält:</label>
    <input type="text" id="white-list-input"/>

    <button class="btn-mini add-on" disabled id="white-list-add">Hinzufügen</button>

    <table class="table table-bordered table-striped" id="white-list">

        <thead>
            <tr>
                <th>URL-Muster</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="#">Entfernen</a></td>
    </tr>
</script>
</body>
</html>

```

Erklärung:

- **Bibliotheken:** Verwendet jQuery, Underscore, Bootstrap und RequireJS für Funktionalität und Stilgestaltung.
- **Benutzeroberflächen-Elemente:**
 - Ein Text-Eingabefeld (#maxTabs) zum Festlegen der maximalen Anzahl von Tabs.
 - Ein Whitelist-Eingabefeld (#white-list-input) und eine “Hinzufügen”-Schaltfläche (#white-list-add), um bestimmte URLs zu schützen.
 - Eine Tabelle (#white-list), um Whitelist-Muster mit einem “Entfernen”-Link anzuzeigen.

- Eine Statusmeldung (#status), um Rückmeldungen zum Speichern anzuzeigen.
 - **Vorlage:** Eine Underscore.js-Vorlage (#url-item-template), die Tabellenzeilen dynamisch generiert.
-

Schritt 3: Popup-Logik (popup.js)

Dieses Skript verarbeitet die Interaktivität des Popups, wie das Speichern von Einstellungen und das Verwalten der Whitelist.

```
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())) {
        $whiteListAddremoveAttr('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('Speichern...').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();
});
}
);

```

Erklärung:

- **Initialisierung:** Verbindet sich mit dem Hintergrundskript GlobalObject für Einstellungen und Tab-Verwaltung.
 - **init:** Richtet Ereignis-Listener ein, wie z. B. verzögerte Eingabe für #maxTabs.
 - **loadOptions:** Lädt gespeicherte Einstellungen (maximale Tabs und Whitelist) und füllt die Benutzeroberfläche.
 - **saveOption:** Speichert Einstellungen in GlobalObject.settings und zeigt eine “Speichern...”-Animation an.
 - **buildWhiteListTable:** Erstellt dynamisch die Whitelist-Tabelle mit Löschfunktion.
 - **Ereignis-Listener:** Verarbeitet Eingabevalidierung, Hinzufügen von Whitelist-Einträgen und Tab-Wechsel.
-

Schritt 4: Hintergrundlogik (background.html und angenommene Skripte)

Die Hintergrundseite (background.html) läuft dauerhaft und lädt die Kernlogik.

```

// background.js (angenommen, basierend auf bereitgestelltem Snippet)
GlobalObject = {};

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

```

Annahmen:

- `settings.js`: Verwalten des Speichers (z. B. `chrome.storage`) für Einstellungen wie `maxTabs` und `whiteList`.
- `tabmanager.js`: Verwenden der `tabs-API`, um Tabs basierend auf der `maxTabs`-Grenze und `whiteList` zu überwachen und zu schließen.

Beispiel `tabmanager.js` (hypothetisch):

```
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); // Entferne ältesten Tab
      }
    });
  }
};
```

So testest du die Erweiterung

1. Öffne Chrome und gehe zu `chrome://extensions/`.
 2. Aktiviere den “Entwicklermodus”(oben rechts).
 3. Klicke auf “Unpacked load” und wähle den `tabs-killer`-Ordner aus.
 4. Klicke auf das Symbol der Erweiterung, um das Popup zu öffnen und die Einstellungen zu testen.
-

Tipps zum Schreiben deiner eigenen Chrome-Erweiterung

1. **Klein anfangen**: Beginne mit einem einfachen Manifest und einem Popup oder Hintergrundskript.
2. **Chrome-APIs verwenden**: Nutze `chrome.tabs`, `chrome.storage` und andere nach Bedarf.
3. **Debugging**: Verwende `console.log` und Chrome’s DevTools (Rechtsklick auf Popup > Inspectieren).

4. **Sicherheit:** Vermeide `unsafe-eval` in der Produktion; verwende strengere Content Security Policies.
 5. **UI-Bibliotheken:** Bootstrap und jQuery erleichtern die UI-Entwicklung, halte die Erweiterung aber leichtgewichtig.
-

Fazit

“Tabs Killer” zeigt, wie man eine Popup-Oberfläche, Hintergrundlogik und Chrome-APIs kombiniert, um eine funktionale Erweiterung zu erstellen. Mit dieser Grundlage kannst du sie weiter anpassen – Benachrichtigungen hinzufügen, die Tab-Schließlogik verfeinern oder die Benutzeroberfläche verbessern.

Fühl dich frei, mit dem Code zu experimentieren und deine eigenen Chrome-Erweiterungsideen zu teilen!
Viel Spaß beim Codieren!