Building a Chrome Extension

Have you ever opened too many browser tabs and wished for a tool to automatically manage them? In this blog post, we'll walk through the creation of a Chrome extension called **"Tabs Killer"**, which automatically closes the oldest tabs when the tab count exceeds a user-defined limit. I'll break down the code, explain how it works, and provide insights to help you build your own Chrome extension.

By the end of this post, you'll understand the structure of a Chrome extension, how to work with the Chrome API, and how to create a popup interface with settings.

What Does "Tabs Killer"Do?

"Tabs Killer"is a Chrome extension that: - Monitors the number of open tabs. - Allows users to set a maximum tab limit. - Automatically closes the oldest tabs when the limit is exceeded. - Provides a whitelist feature to protect specific tabs (e.g., based on URL patterns) from being closed.

The extension includes a popup interface for configuring settings and a background script to handle tab management.

Project Structure

Here's the file structure of the "Tabs Killer"extension:

```
tabs-killer/
  manifest.json        # Extension configuration
  popup.html           # Popup UI
  popup.js            # Popup logic
  background.html      # Background page
  app.build.js         # Main app logic (assumed)
  js/
     lib/              # External libraries (jQuery, Underscore, Bootstrap, RequireJS)
     tabmanager.js     # Tab management logic (assumed)
     settings.js       # Settings management (assumed)
  css/
     popup.css         # Popup styles
  img/
     icon16.png        # 16x16 icon
```

```
    icon48.png        # 48x48 icon
    icon128.png       # 128x128 icon
```

Step 1: The Manifest File (`manifest.json`)

The `manifest.json` file is the heart of any Chrome extension. It defines metadata, permissions, and key components.

```json
{
  "manifest_version": 2,
  "name": "Tabs Killer",
  "description": "Automatically kill the oldest tabs when tabs are too many.",
  "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'"
}
```

Explanation:

- **`manifest_version`**: Must be `2` (Chrome deprecated version 1).
- **`name`, `description`, `version`**: Basic metadata.
- **`browser_action`**: Defines the extension's toolbar icon and popup (`popup.html`).
- **`icons`**: Icons for different sizes (used in the Chrome Web Store and toolbar).

- `background`: Specifies a background page (`background.html`) that runs persistently.
- `permissions`: Requests access to the `tabs` API (to manage tabs) and `storage` API (to save settings).
- `content_security_policy`: Allows `unsafe-eval` for libraries like RequireJS (use cautiously in production).

Step 2: The Popup UI (`popup.html`)

The popup appears when the user clicks the extension icon. It uses Bootstrap for styling and includes a tabbed interface with an “Options”section.

```
<!doctype html>
<html>
<head>
  <title>Tabs Killer Extension's Popup</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>Settings</legend>
          <p>
            <label for="maxTabs">Maximum tabs to stay on</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 with url which contains string:</label>
          <input type="text" id="white-list-input"/>
          <button class="btn-mini add-on" disabled id="white-list-add">Add</button>
          <table class="table table-bordered table-striped" id="white-list">
            <thead>
              <tr>
                <th>Url Pattern</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="#">Remove</a></td>
    </tr>
  </script>
</body>
</html>
```

Explanation:

- **Libraries**: Uses jQuery, Underscore, Bootstrap, and RequireJS for functionality and styling.
- **UI Elements**:
  - A text input (`#maxTabs`) for setting the maximum number of tabs.
  - A whitelist input (`#white-list-input`) and "Add"button (`#white-list-add`) to protect specific URLs.
  - A table (`#white-list`) to display whitelist patterns with a "Remove"link.
  - A status message (`#status`) to show saving feedback.
- **Template**: An Underscore.js template (`#url-item-template`) dynamically generates table rows.

Step 3: Popup Logic (`popup.js`)

This script handles the popup’s interactivity, such as saving settings and managing the 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())) {
        $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('Saving...').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();
  });
});
```

Explanation:

- **Initialization**: Connects to the background page's `GlobalObject` for settings and tab management.
- **`init`**: Sets up event listeners, like debounced input for `#maxTabs`.
- **`loadOptions`**: Loads saved settings (max tabs and whitelist) and populates the UI.
- **`saveOption`**: Saves settings to `GlobalObject.settings` and shows a "Saving…"animation.
- **`buildWhiteListTable`**: Dynamically builds the whitelist table with delete functionality.
- **Event Listeners**: Handles input validation, adding whitelist entries, and tab switching.

Step 4: Background Logic (`background.html` and Assumed Scripts)

The background page (`background.html`) runs persistently and loads the core logic.

```
// background.js (assumed, based on provided snippet)
GlobalObject = {};

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

Assumptions:

- **`settings.js`**: Manages storage (e.g., `chrome.storage`) for settings like `maxTabs` and `whiteList`.
- **`tabmanager.js`**: Uses the `tabs` API to monitor and close tabs based on the `maxTabs` limit and `whiteList`.

Example `tabmanager.js` (hypothetical):

```
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); // Remove oldest tab
      }
    });
  }
};
```

How to Test the Extension

1. Open Chrome and go to `chrome://extensions/`.
2. Enable "Developer mode"(top-right toggle).
3. Click "Load unpacked"and select the `tabs-killer` folder.
4. Click the extension icon to open the popup and test the settings.

Tips for Writing Your Own Chrome Extension

1. **Start Small**: Begin with a simple manifest and a popup or background script.
2. **Use Chrome APIs**: Leverage `chrome.tabs`, `chrome.storage`, and others as needed.
3. **Debugging**: Use `console.log` and Chrome's DevTools (right-click popup > Inspect).
4. **Security**: Avoid `unsafe-eval` in production; use stricter Content Security Policies.
5. **UI Libraries**: Bootstrap and jQuery simplify UI development but keep the extension lightweight.

Conclusion

"Tabs Killer"demonstrates how to combine a popup UI, background logic, and Chrome APIs to create a functional extension. With this foundation, you can customize it further—add notifications, refine the tab-closing logic, or enhance the UI.

Feel free to experiment with the code and share your own Chrome extension ideas! Happy coding!