


Chrome Extension

Ashwini Kemshtetty

Agenda

- What is chrome extension
 - How are they useful
 - Architecture and Project structure
 - Core concepts
 - Development and Debugging
 - Deployment and Distribution
 - Conclusion and Q&A
- 



Let's Think

What is a chrome extension?

Chrome extensions enhance the browsing experience by customizing the user interface, observing browser events, and modifying the web.



How are they useful?

- Ease of use
- Enhancing productivity
- Automate repetitive task and save time
- Customize the browsing experience
- Web Accessibility
- And many more.....





Demo



Let's Reflect

Our findings

- Extension Icon
- Extension Popup and Options page
- Webpage
- Extension and webpage are two different entities
- Injection of scripts on webpage
- Permissions to access contents of webpage
- Communication Mechanism






Let's connect the . . .

Architecture & Project structure

```
└─ extension-sample/  
  └─ manifest.json  
  └─ service-worker.js  
  └─ scripts/  
    └─ content-script.js  
  └─ popup/  
    └─ popup.css  
    └─ popup.js  
    └─ popup.html  
  └─ options/  
    └─ options.css  
    └─ options.js  
    └─ options.html  
  └─ icons/  
    └─ 16.png  
    └─ 32.png  
    └─ 48.png  
    └─ 128.png
```

The Manifest file

- Manifest file is a JSON formatted configuration file of a Chrome extension
 - Name of the file must be manifest.json. It should be in the root directory of project
 - Provides metadata of the extension. Minimum chrome version is 88 for manifest v3
 - The name of the extension, a description of what it does, the current version number, and what icons to use
 - The Chrome API keys and permissions that the extension needs
 - The files assigned as the extension service worker, the popup HTML file, the options page, the content scripts, resources, etc
- 

Sample Manifest file

```
// Required
"manifest_version": 3,
"name": "My Extension",
"version": "1.0.1",

// Recommended
"action": {...},
"default_locale": "en",
"description": "A plain text description",
"icons": {...},

// Optional
"author": "developer@example.com",

"background": {...},
"chrome_settings_overrides": {...},
"chrome_url_overrides": {...},
"commands": {...},
```

Popup page

Hello React play

Update Title

Update Description

Update Image

Content scripts

- Content scripts execute Javascript in the context of a web page
- Can read and modify the DOM of the pages they're injected into and send information to its parent extension.
- Content Scripts can only use a subset of the Chrome APIs but can indirectly access the rest by exchanging messages with the extension service worker.



Communication - Message Passing

- Communication between extensions and their content scripts works by using message passing
- Either side can listen for messages sent from the other end, and respond on the same channel
- A message can contain any valid JSON object (null, boolean, number, string, array, or object)
- There are 2 types of connections to exchange messages
 - Simple one-time requests (`runtime.sendMessage()` or `tabs.sendMessage()`)
 - Complex long-lived requests (`runtime.connect` or `tabs.connect`)



The Service Worker in general

- A service worker is a script that runs in the background of a web application, separate from the main web page, and is designed to perform tasks that don't require direct user interaction.
- Offline support, Faster page loads, Push notifications, Background synchronization, Resource caching.



The Service Worker in extension

- Handles and listens for browser events.
- Is loaded when it is needed, and unloaded when it goes dormant
- Once loaded, it generally runs as long as it is actively receiving events, though it can shut down
- To register the service worker, add background field in the manifest file
- To update the service worker, publish a new version of your extension to the Chrome Web Store.



Registration of service worker in the manifest

```
{  
  "name": "Awesome Test Extension",  
  ...  
  "background": {  
    "service_worker": "service-worker.js"  
  },  
  ...  
}
```

Web Accessible Resources

- Web-accessible resources are files inside an extension that can be accessed by web pages or other extensions
- Extensions typically use this feature to expose images or other assets that need to be loaded in web pages, but any asset included in an extension's bundle can be made web accessible.
- By default no resources are web accessible



Web Accessible Resources in manifest

```
{  
  ...  
  "web_accessible_resources": [  
    {  
      "resources": [ "test1.png", "test2.png" ],  
      "matches": [ "https://web-accessible-resources-1.glitch.me/*" ]  
    }, {  
      "resources": [ "test3.png", "test4.png" ],  
      "matches": [ "https://web-accessible-resources-2.glitch.me/*" ],  
      "use_dynamic_url": true  
    }  
  ],  
  ...  
}
```

Options page

- Options page enables customization of the extension
- Use options to enable features and allow users to choose what functionality is relevant to their needs
- There are two types of extension options pages, full page and embedded. The type of options page is determined by how it is declared in the manifest



Externally connectable

- Declares which extensions and web pages can connect to your extension via `runtime.connect` and `runtime.sendMessage`.
- If the `externally_connectable` key is not declared in your extension's manifest, all extensions can connect, but no web pages can connect
- As a consequence, when updating your manifest to use `externally_connectable`, if `"ids": ["*"]` is not specified, then other extensions will lose the ability to connect to your extension



Manifest V3 vs V2

- Security, Privacy and Performance
- Replacing background pages with service workers
- Support for Promises. Callbacks are still backward compatible
- Host_permissions key in manifest file
- All the code should be within the package submitted to chrome and removed support for remotely hosted code



Debugging & Storage



Let's Understand Deployment & Distribution Process

Publish in the Chrome Web Store

- Create your item's zip file.
- Create and setup a developer account.
- Upload your item.
- Add assets for your listing.
- Submit your item for publishing.





Questions ?



Thank you!