

Cross-Browser Extension

The cross browser extension is an extension that can be run on any web browser, means we can share same piece of code across all the web stores.

Below are the few challenges faced while creating a cross browser extension.

1. API namespace - There are 2 main API namespaces that multiple browsers supports.

1. browser.* , it is used by the firefox, safari.
2. chrome.* , it is used by chrome, edge, etc.

It is preferable to use browser.* namespace. This is because it supports promises and asynchronous functions.

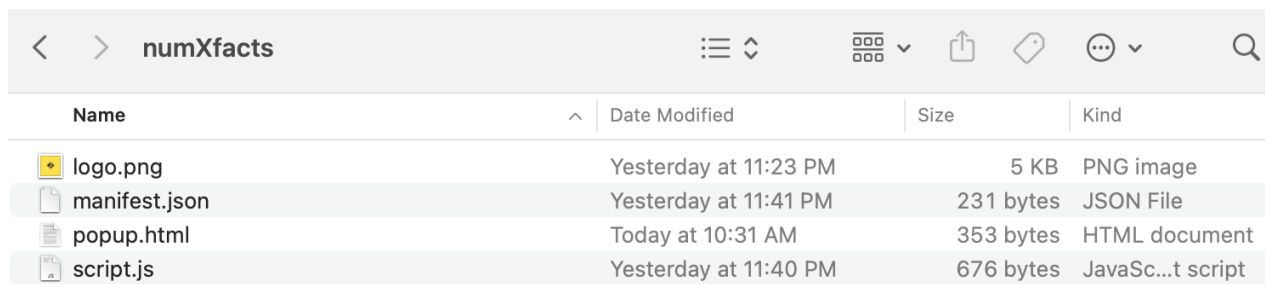
2. API asynchronous event handling - Below 2 approaches are used among the main browsers.

1. Promises, the proposed std for the extensions API used by Firefox.
2. Callbacks, used by Chrome.

3. Web extension browser API polyfill - It solves the issues like API namespace and API asynchronous event handling across chrome and firefox. I'll read more about it and later I'll update more info about it.

Folder Structure:

The basic folder structure of a browser extension looks like -



Name	Date Modified	Size	Kind
logo.png	Yesterday at 11:23 PM	5 KB	PNG image
manifest.json	Yesterday at 11:41 PM	231 bytes	JSON File
popup.html	Today at 10:31 AM	353 bytes	HTML document
script.js	Yesterday at 11:40 PM	676 bytes	JavaSc...t script

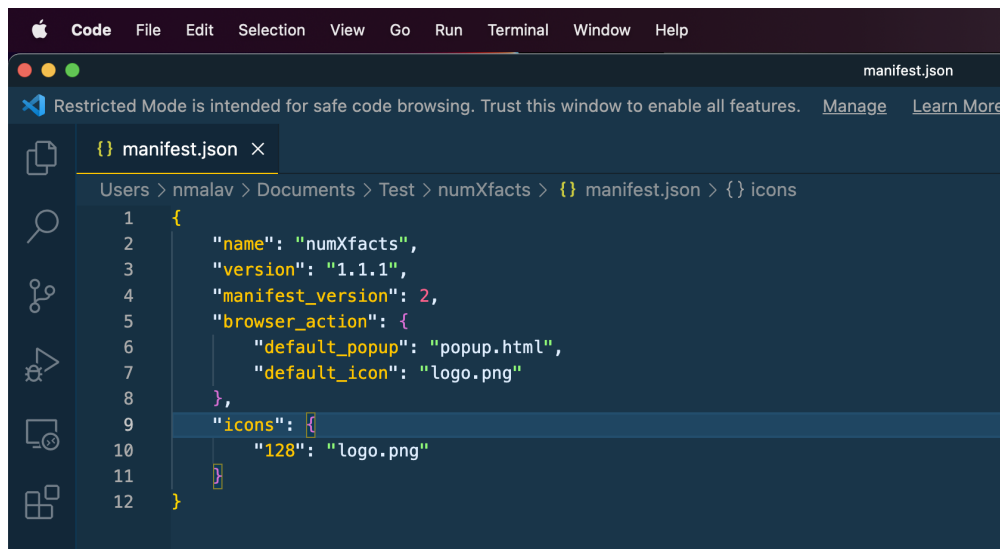
where, manifest.json contains info about the extension, like name, version number, manifest version, etc.

popup.html is the display window of the extension.

script.js contains what's needed to be performed in the background of extension, like API calls, etc.

Manifest.json:

Every web extension must include the manifest.json file. It specifies the basic metadata about extensions such as name, version number. It can also specify the extension functionality such as browser actions, content scripts, etc.

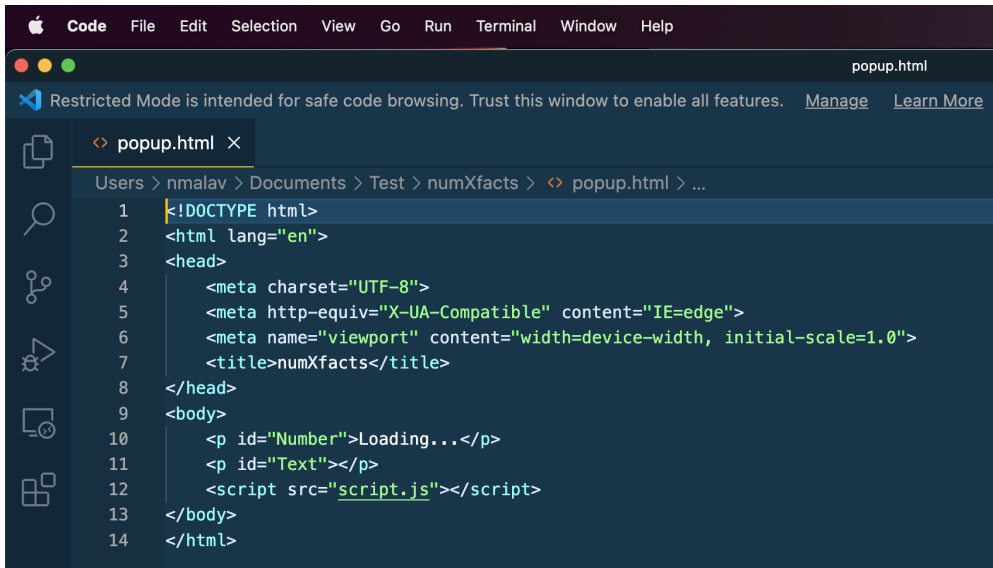


```
{} manifest.json x
Users > nmalav > Documents > Test > numXfacts > {} manifest.json > {} icons
1  {
2    "name": "numXfacts",
3    "version": "1.1.1",
4    "manifest_version": 2,
5    "browser_action": {
6      "default_popup": "popup.html",
7      "default_icon": "logo.png"
8    },
9    "icons": {
10     "128": "logo.png"
11   }
12 }
```

Detail description about all the manifest keys supported in different browsers by following [link](#).

Popup.html:

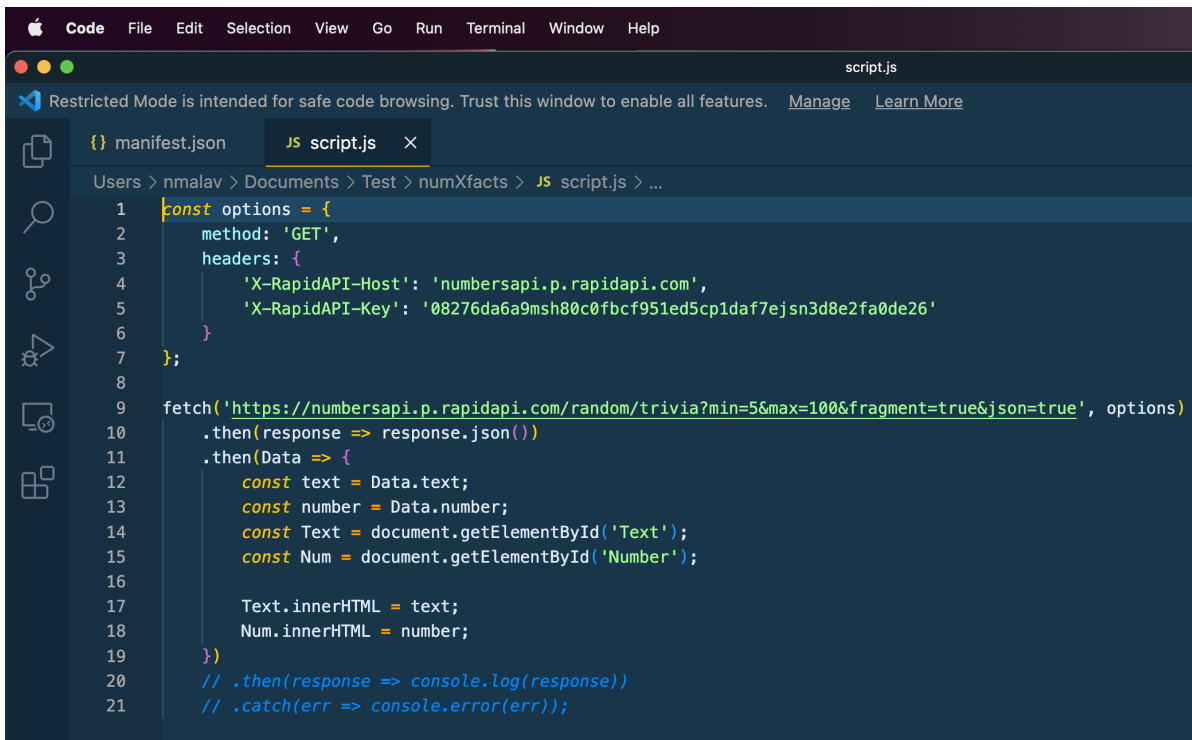
It will make the UI of the popup as well the logic being executed.



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>numXfacts</title>
8 </head>
9 <body>
10  <p id="Number">Loading...</p>
11  <p id="Text"></p>
12  <script src="script.js"></script>
13 </body>
14 </html>
```

Script.js:

For our sample cross web extension - numXfacts, the script.js made an api call to get the fact associated with the random number.



```
1 const options = {
2   method: 'GET',
3   headers: {
4     'X-RapidAPI-Host': 'numbersapi.p.rapidapi.com',
5     'X-RapidAPI-Key': '08276da6a9msh80c0fbcf951ed5cp1daf7ejsn3d8e2fa0de26'
6   }
7 };
8
9 fetch('https://numbersapi.p.rapidapi.com/random/trivia?min=5&max=100&fragment=true&json=true', options)
10  .then(response => response.json())
11  .then(Data => {
12    const text = Data.text;
13    const number = Data.number;
14    const Text = document.getElementById('Text');
15    const Num = document.getElementById('Number');
16
17    Text.innerHTML = text;
18    Num.innerHTML = number;
19  })
20  // .then(response => console.log(response))
21  // .catch(err => console.error(err));
```

