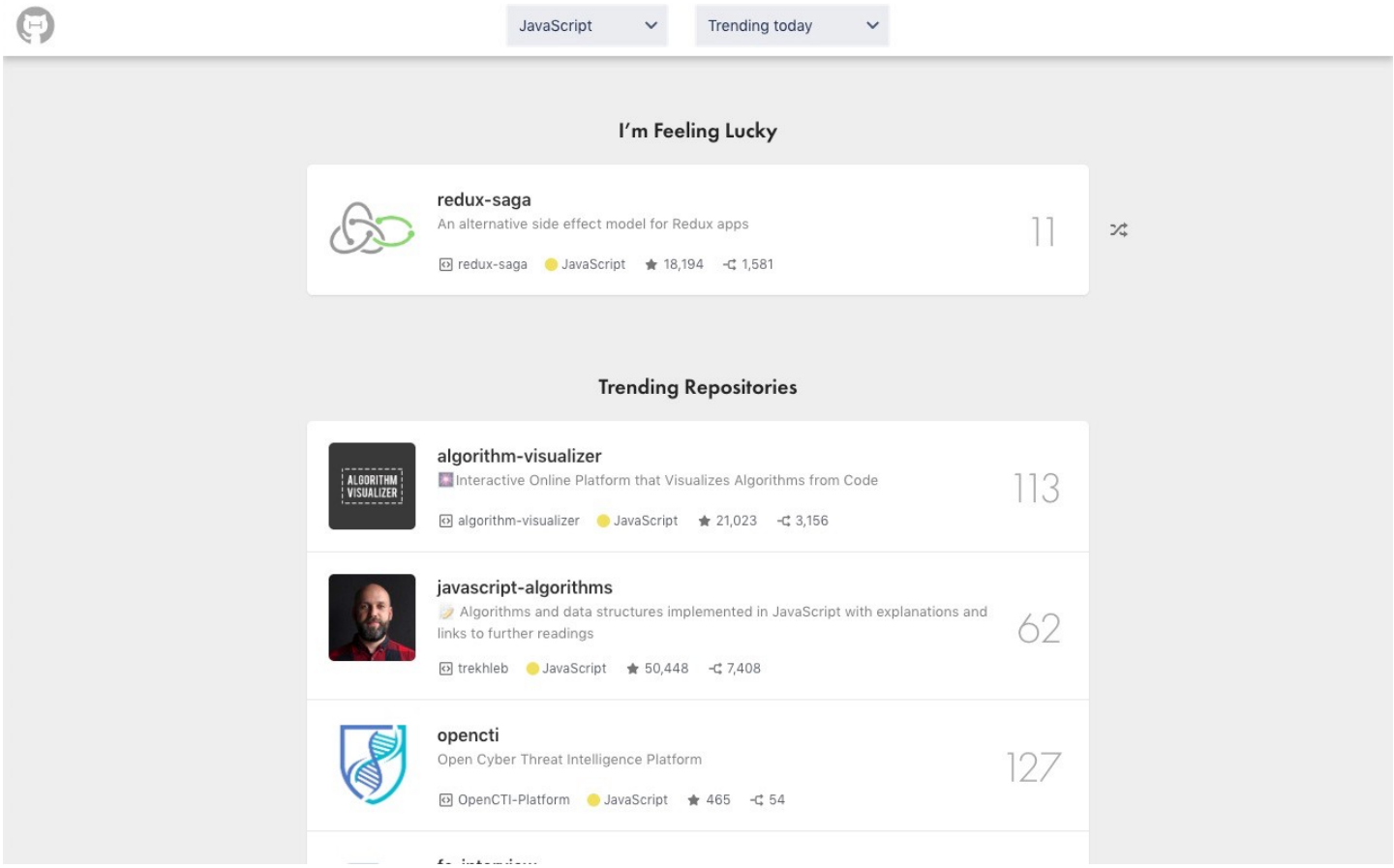
How to use background script to fetch data in a Chrome extension

Hu Chen
Jul 8 · 6 min read



Hacker Tab: Chrome extension to view GitHub trending projects on new tab

If you are not sure how to write a chrome extension, you may first have a look at my previous story **How to use React.js to create a cross-browser extension in 5 minutes**. It teaches you how to use create-react-app to build a browser extension for the new tab.

The extension we built was good, but every time the user opens a new tab we will reload the latest trending repositories from API which could take a few seconds. Even if

it just takes a second, it is still slow for a new tab opening.

> What if we fetch data in background periodically, save the data in localStorage, and read it on open a new Tab?

I have spent a few days to research about it and discovered that it is totally feasible. In this article, I will share with you what I have learned during the process.

1. Adding background script to manifest.json

Background script can react to browser events and perform certain actions, the background page is loaded when it is needed, and unloaded when it goes idle. To use the background file, declare it as follows in the `manifest.json` file:

```
{
  "name": "Awesome Test Extension",
  ...
  "background": {
    "scripts": ["background.js"],
    "persistent": false
  },
  ...
}
```

You will notice the `persistent` parameter, a persistent background page exists during the lifetime of the extension and only one instance of it actively running in the background of the Chrome browser waiting for the user to interact with the extension.

According to Google developer documentation, it is more recommended to use non-persist mode and it could help reduce the resource cost of your extension. A non-persistent background script is purely event-based, it stays dormant until an event they are listening for fires, react with specified instructions, then unload. Some examples of events include:

- The extension is first installed or updated to a new version or browser restarted.

- The background page was listening for an event, and the event is dispatched. (e.g. We will create `alarm` to dispatch event periodically)
- A content script or other extension sends a message.
- Another view in the extension, such as a popup, calls `runtime.getBackgroundPage`.

2. Adding alarms to trigger actions periodically

You might be familiar with the code below to run something periodically:

```
window.setInterval(function() {
  console.log('Hello, world!');
}, 1000 * 60 * 3);
```

It prints “Hello, world!” every 3 minutes. To change it to event-based alarm, we will use alarms API instead.

```
chrome.alarms.create('refresh', { periodInMinutes: 3 });
```

Then add a listener.

```
chrome.alarms.onAlarm.addListener((alarm) => {
  alert("Hello, world!");
});
```

We will need to create an alarm when the extension is installed so that the alarm will be triggered every 3 minutes in the background, this is our `background.js` so far.

```
1  chrome.runtime.onInstalled.addListener(() => {
2    console.log('onInstalled...');
3    // create alarm after extension is installed / upgraded
4    chrome.alarms.create('refresh', { periodInMinutes: 3 });
5  });
6
7  chrome.alarms.onAlarm.addListener((alarm) => {
8    console.log(alarm.name); // refresh
9    helloWorld();
10 });
```

```
11
12  function helloWorld() {
13    console.log("Hello, world!");
14  }
```

background.js hosted with ♡ by GitHub view raw

To use alarms, you will also need to add `alarms` permission in `manifest.json`

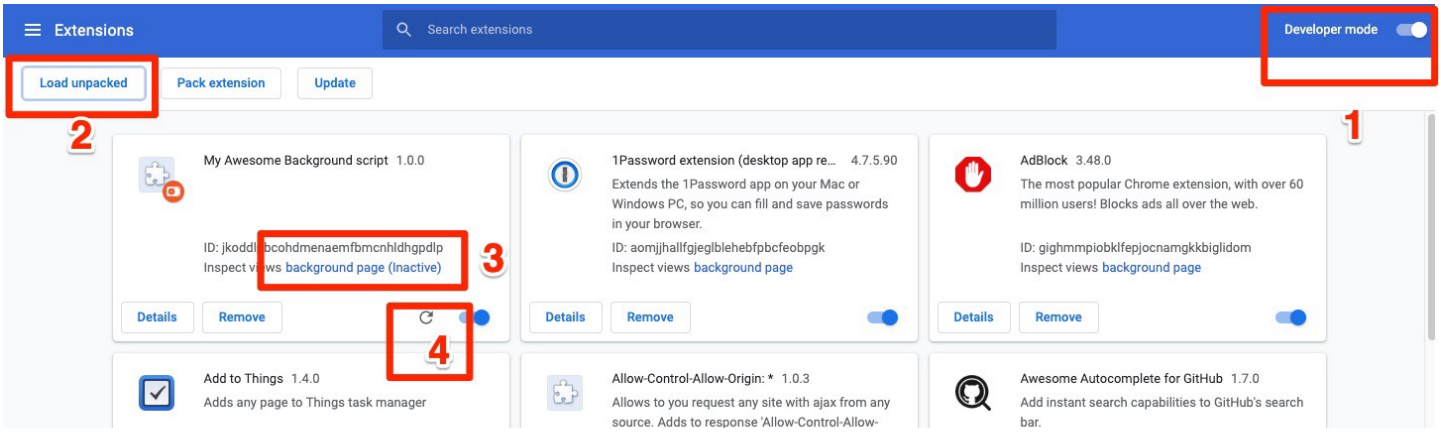
```
1  {
2    "manifest_version": 2,
3    "name": "My Awesome Background script",
4    "version": "1.0.0",
5    "permissions": ["alarms"],
6    "background": {
7      "scripts": ["background.js"],
8      "persistent": false
9    }
10 }
```

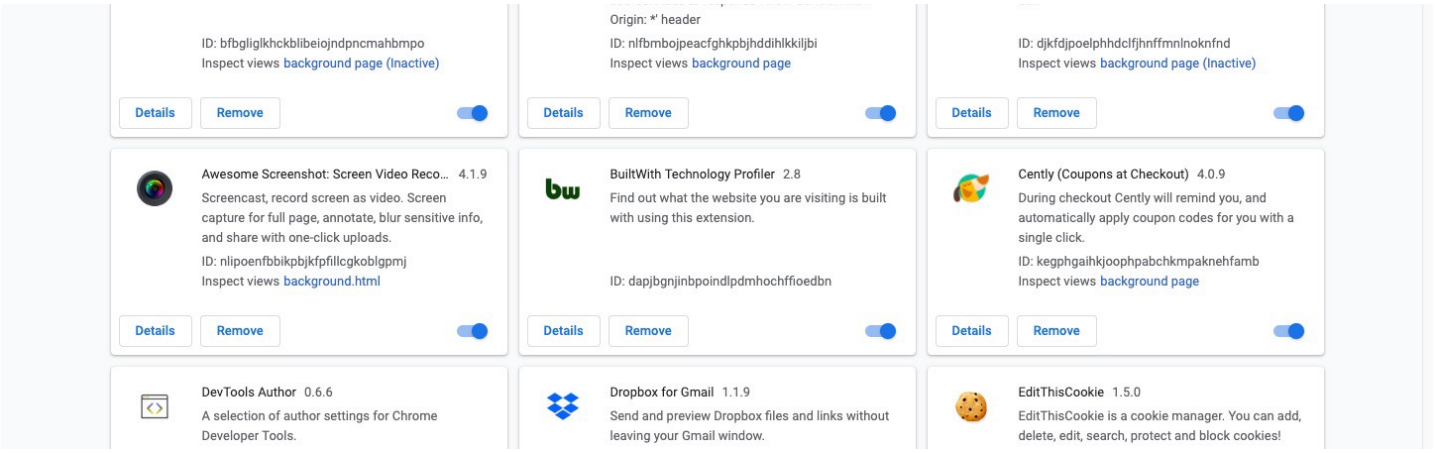
manifest.json hosted with ♡ by GitHub view raw

3. Debugging background script

To test whether the above scripts could run as expected, we will need to test it locally. This is very similar to the steps in previous story.

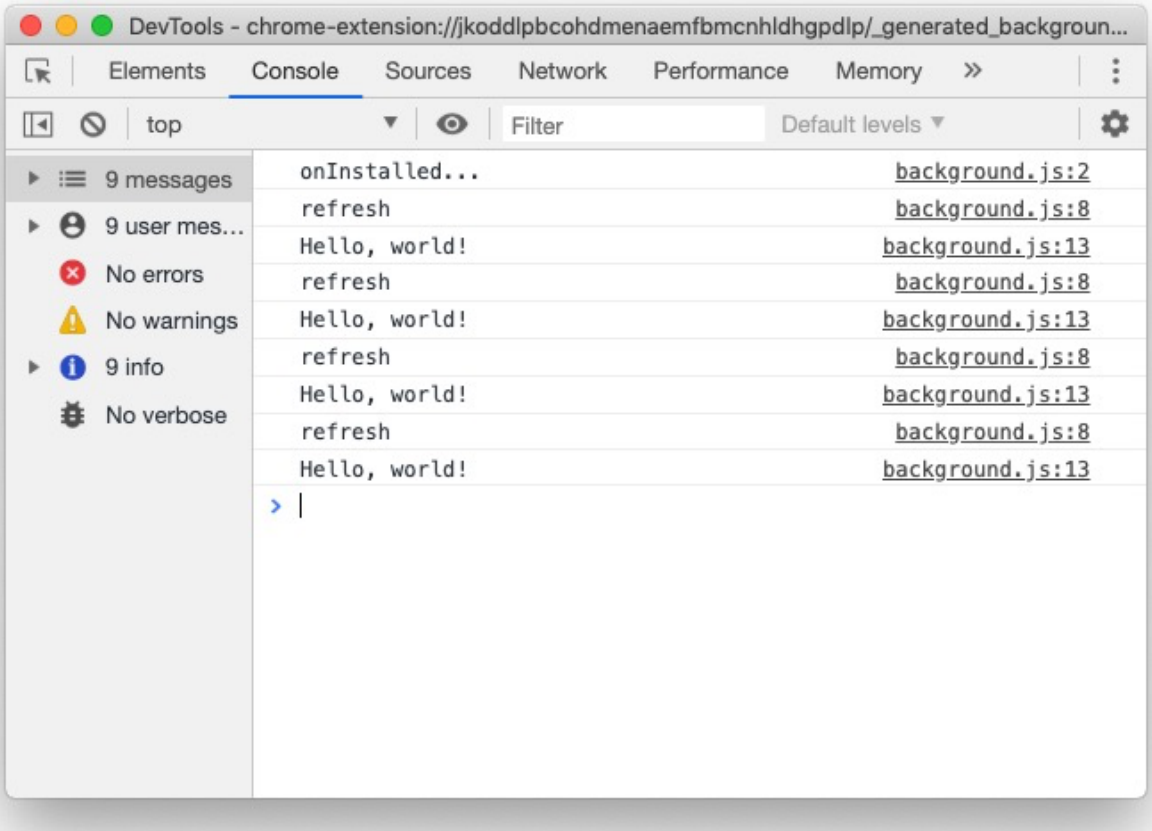
1. Make sure developer mode is on as shown on the image above.
2. Click “Load unpacked” and target folder contains your `manifest.json` .
3. Instead of inspecting on the new tab, you should click `background page`, as background page is running in a separate process.
4. If you have changed code locally, you will need to click reload.





Test extension locally

As you can see from the image below, the alarms are running every 3 minutes.



Inspect background script

And you could check **network**, **localStorage** here as well. If your extension modified new tab, the localStorage is shared across your extension, that is how we communicate data between your background script and the new tab extension.

4. Fetch data and save to localStorage

Now we have a basic alarm working, let's fetch data and save to local storage. Later in our React extension, we could direct read from local storage on loaded instead of fetch from API.

1. We will create a "**refresh**" alarm to fetch and save data every 30 minutes when the extension is first installed or upgraded.
2. We will fetch data when chrome is restarted so that the user will get the latest data.
3. We will also create another "**watchdog**" alarm every 5 minutes to check whether the refresh alarm is still available, if not, we will create it. (It might not be necessary for it here)

```
// create alarm for watchdog and fresh on installed/updated, and start fetch data
chrome.runtime.onInstalled.addListener(() => {
  console.log('onInstalled....');
  scheduleRequest();
  scheduleWatchdog();
  startRequest();
});

// fetch and save data when chrome restarted, alarm will continue running when chrome is restar
chrome.runtime.onStartup.addListener(() => {
  console.log('onStartup....');
  startRequest();
});

// alarm listener
chrome.alarms.onAlarm.addListener(alarm => {
  // if watchdog is triggered, check whether refresh alarm is there
  if (alarm && alarm.name === 'watchdog') {
    chrome.alarms.get('refresh', alarm => {
      if (alarm) {
        console.log('Refresh alarm exists. Yay.');
      } else {
        // if it is not there, start a new request and reschedule refresh alarm
        console.log("Refresh alarm doesn't exist, starting a new one");
        startRequest();
        scheduleRequest();
      }
    });
  } else {
    // if refresh alarm triggered, start a new request
    startRequest();
```

```
    }
  });

  // schedule a new fetch every 30 minutes
  function scheduleRequest() {
    console.log('schedule refresh alarm to 30 minutes...');
    chrome.alarms.create('refresh', { periodInMinutes: 30 });
  }

  // schedule a watchdog check every 5 minutes
  function scheduleWatchdog() {
    console.log('schedule watchdog alarm to 5 minutes...');
    chrome.alarms.create('watchdog', { periodInMinutes: 5 });
  }

  // fetch data and save to local storage
  async function startRequest() {
    console.log('start HTTP Request...');
    const data = await fetchRepositories();
    saveToLocalStorage(data);
  }
```

background.js is hosted with ♡ by GitHub view raw

5. Reuse logic and transpile to ES5

There are two problems with the code above and we will need to solve them now:

1. Notice that we did not write the implementation for `fetchRepositories` and `saveToLocalStorage`, they are most likely already available in your React app, it is fine to copy the implementation to `background.js` but it will be better if we could reuse the functions to keep code DRY.
2. We have written the `background.js` using ES6 syntax like `=>` and `async await`, older browsers might not be able to run it.

Here we will use `webpack` to help bundle and transpile the code to ES5.

5.1 Install dependencies

```
yarn add --dev webpack-cli npm-run-all rimraf
```

Notice we should not install `webpack`. (`react-scripts` already have it in dependency and it will complain about another instance of `webpack`). Feel free to install it if you are not using `create-react-app` .

5.2 Change build script

Now change your `build` script to build both your app and background script:

```
"prebuild": "rimraf build",
"build": "npm-run-all build:*",
"build:app": "INLINE_RUNTIME_CHUNK=false react-scripts build",
"build:bg": "webpack --mode production ./src/background.js --output
./build/background.js",
```

We have already covered `INLINE_RUNTIME_CHUNK=false` in a previous tutorial. After the change, if you run `npm run build` , it will perform

- Clean the `build` folder
- Bundle the React extension using `react-script`
- Bundle the `src/background.js` using webpack and export to `build/background.js`

5.3 Refactor ./src/background.js

In your `src/background.js` , you are free to import any external libraries like `lodash` or import constants/functions from other files, such as:

```
import {
  fetchRepositories,
  saveToLocalStorage
} from './lib/helpers';

chrome.runtime.onInstalled.addListener(() => {
...
```

Your code is much cleaner now.

5.4 Update ESlint config

If you are using ESLint in your editor, it might be complaining about `chrome` is not defined in your `./src/background.js` . That is because `chrome` API is only available in

extensions, we will need to tell ESLint that we are developing extension and please ignore those.

Add/update the following lines in your `package.json` .

```
"eslintConfig": {
  "extends": "react-app",
  "env": {
    "browser": true,
    "webextensions": true
  }
}
```

After adding `webextensions` to `env` , editor will no longer complain about it.

5.5 Add .babelrc

Webpack will need a `.babelrc` file to compile, and we will need to add it manually. Since `react-scripts` have already installed `babel-presets-react-app` , we will just need to use it.

```
{
  "presets": ["react-app"]
}
```

Now run `npm run build` will have your complete extension together with compiled background script in your `build` folder.

That's it!

If you are curious, you could check all the file changed in the commit of my extension on GitHub.

. . .

Thank you for reading this far. You could check the source code in Github or download the extension in Chrome Web Store and Firefox Add-ons Hub, happy hacking, and let me know what you have built!

Chrome Extension Developer Tools Github Create React App Reactjs

Medium About Help Legal