Wiki Translator: A Google Chrome Extension for Translating the Longest Language of a Wikipedia Article

吳秉倫

廖翊廷

陳立峰

楊立宇

CED

https://github.com/palapapa/wiki-translator

Abstract

In a Wikipedia article, it is often possible that some language versions of it are longer than others, typically when the subject of the article is about someone or something in a different country that uses another language. Without prior knowledge about the origin of the subject, it can be difficult to know which language version contains the most information.

To solve this problem, we decided to develop a Google Chrome extension that can automatically detect which language version of an article has the most content, decided by comparing the length of their respective HTML source after translating each of them into a "target language", usually the native language of the user, and replace the on-screen article with the translated one that has the longest length.

Keywords: Chrome Extension, Browser Extension

Wiki Translator: A Google Chrome Extension for Translating the Longest Language of a Wikipedia Article

Contents

List of Figures

Conception

The idea for this project originated from Chrome Extension Ideas (2023) where user @synanthropy asked for a browser extension that automatically detects which language version of a Wikipedia article is the longest and shows you the translated version of it. We thought this was a good idea and decided to implement it.

Technologies Used

1. HTML
2. CSS
3. TypeScript
4. Webpack

Implementation

The implementation of this project can be divided into the following parts:

1. Check if the user is on a Wikipedia article.
2. Get the title of the article.
3. Get the HTML sources of all language versions of the article with the MediaWiki API.
4. Translate the HTML sources with the Google Translate API.
5. Compare the length of the translated HTML sources.
6. Replace the original article on-screen with the longest translated one.

These steps are further explained below:

Check if the user is on a Wikipedia article

This is done by using the `checkIfOnWikipedia` function in `background.ts`, and make 5 different Chrome events subscribe to them, namely `chrome.runtime.onInstalled`, `chrome.runtime.onStartup`,`chrome.tabs.onActivated`, `chrome.tabs.onUpdated`, and `chrome.windows.onFocusChanged`. The function checks whether the substring from the next character after the first dot to the end of the string of the current URL's hostname is `wikipedia.org`, and enables/disables the extension's popup accordingly.

Get the title of the article

This is done by the `getTitle` function in `articleFetcher.ts`. This function takes the current URL and attempts to split it by the following delimiters: `/wiki/`, `/zh-tw/`, `/zh-cn/`, `/zh-hk/`, `/zh-mo/`, `/zh-my/`, and `/zh-sg/`. The first one in that order that successfully splits the URL will take effect and the function will return the substring after that delimiter. This is done because in any URL of a given Wikipedia article, you can find the title of that article, which acts like a unique identifier in the database of articles from Wikipedia. The title is needed to make requests to the MediaWiki API for data about a certain article. Usually, the title is the substring after the substring `/wiki/` in the URL. However, when the Chinese conversion system "使用說明: 中文維基百科的繁簡、地區詞處理" (n.d.) is in effect, that is not the case, and the title is the substring after one of the 6 delimiters mentioned above, based on the Chinese variant used.

Get the HTML sources of all language versions of the article with the MediaWiki API

This is done by the `fetchAllLanguages` function in `articleFetcher.ts`, which calls two other functions, `getLanglinks`, and `getHtml`. `getLanglinks` calls the MediaWiki API with the title of the current article, and the API will return the URLs of all language versions of that article. `getHtml` then uses that list of URLs and requests their respective HTML sources from the MediaWiki API.

Translate the HTML sources with the Google Translate API

This is done by the `translateArticles` function in `articleTranslator.ts`. It takes an array of `WikiArticle` objects and send them to the Google Translate API for translation, and returns an array of translated `WikiArticle` objects. If Google Translate does not support translating from some language version's original language to the target language, then its translated `WikiArticle` object will contain a `null` HTML source.

Compare the length of the translated HTML sources

This is simply done by comparing the lengths of the translated HTML sources. We originally thought to compare the word counts of the text nodes of the HTML sources, but later realized that wasn't easy, because there wasn't a language-agnostic way to define what a "word" was, especially in languages that don't separate words with spaces. Moreover, directly counting the lengths of the translated HTML sources isn't such a bad idea, because in theory, once the HTML sources have all been translated into a single language, they should have roughly the same information density, which is not the case if they are in different languages.

Replace the original article on-screen with the longest translated one

This is done by `translationReplacer.ts`. It finds the `div` with a `class` of `mw-parser-output` in the user's current page and replaces its `innerHTML` with the translated language version the user selected. It also removes the `div` with a `class` of `siteNotice` if it exists on the current page, because when that element is present, then after the replacement is done, and the site notice element starts cycling to the next notice, it will cause a strange bug

where the translated article does not replace the original one, but is rather concatenated before the original one, and then the site notice element will become very large and completely white and covers the translated article.

Demonstration

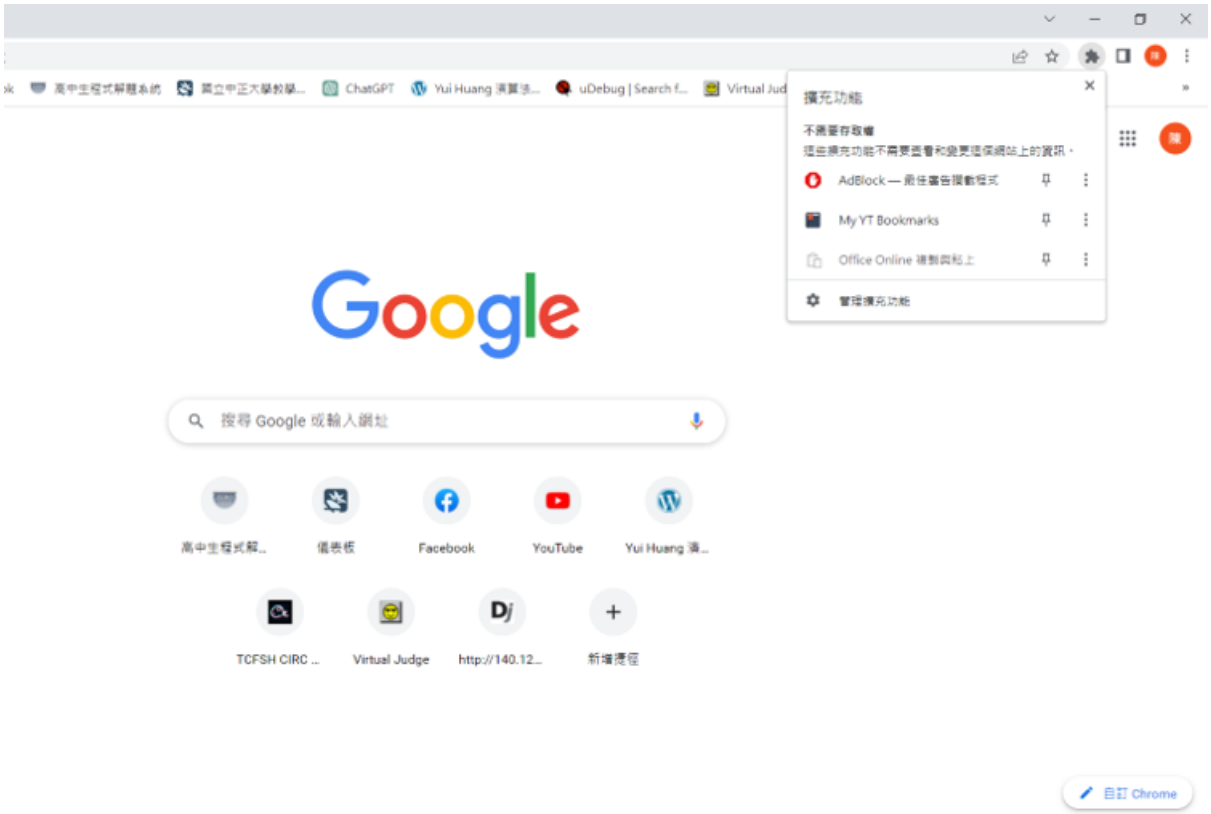
Installation

1. Run `npm i; npm install;` in the extension's root directory.

Figure 1. Installation step 1

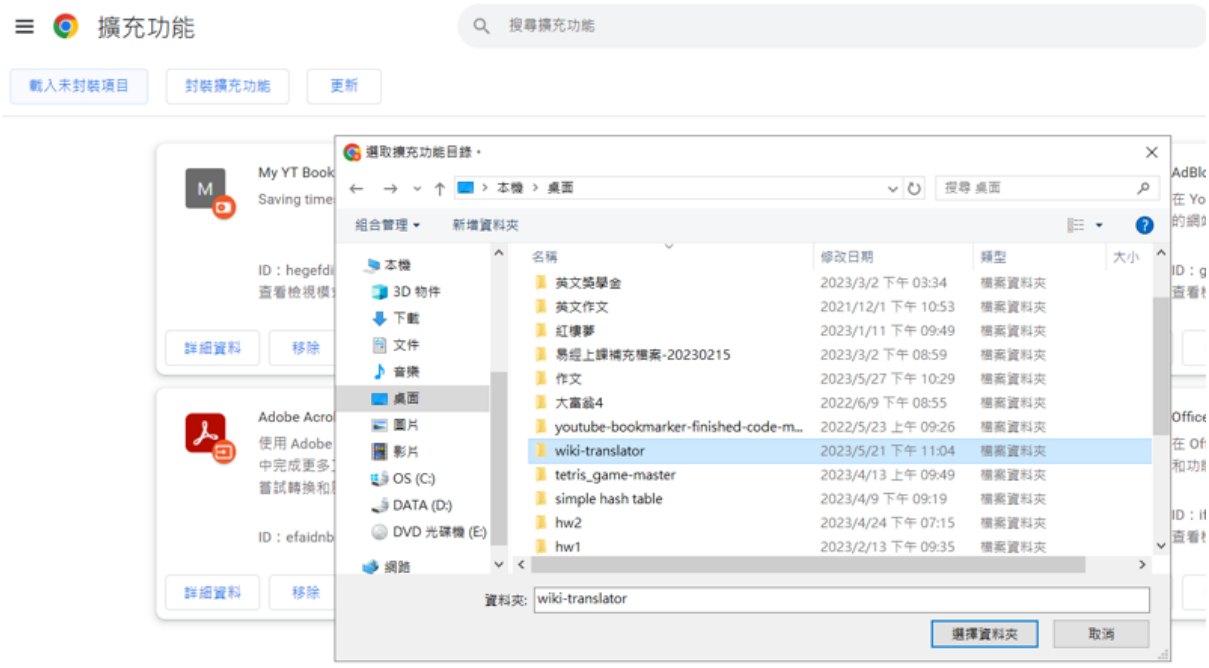
2. Open "Manage extensions".

Figure 2. Installation step 2



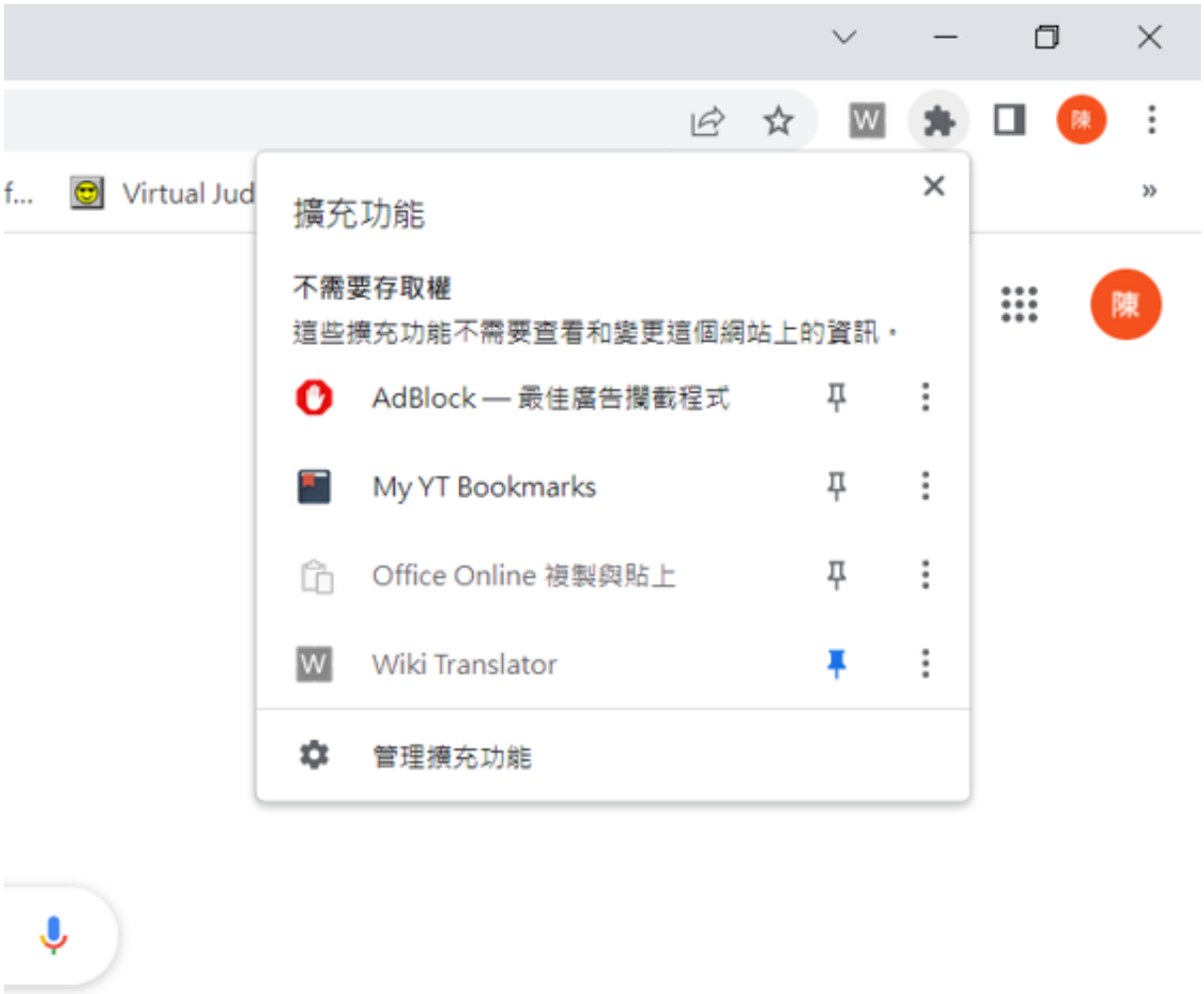
3. Click “Load unpacked” and select the extension’s folder.

Figure 3. Installation step 3



4. Pin the extension to the toolbar.

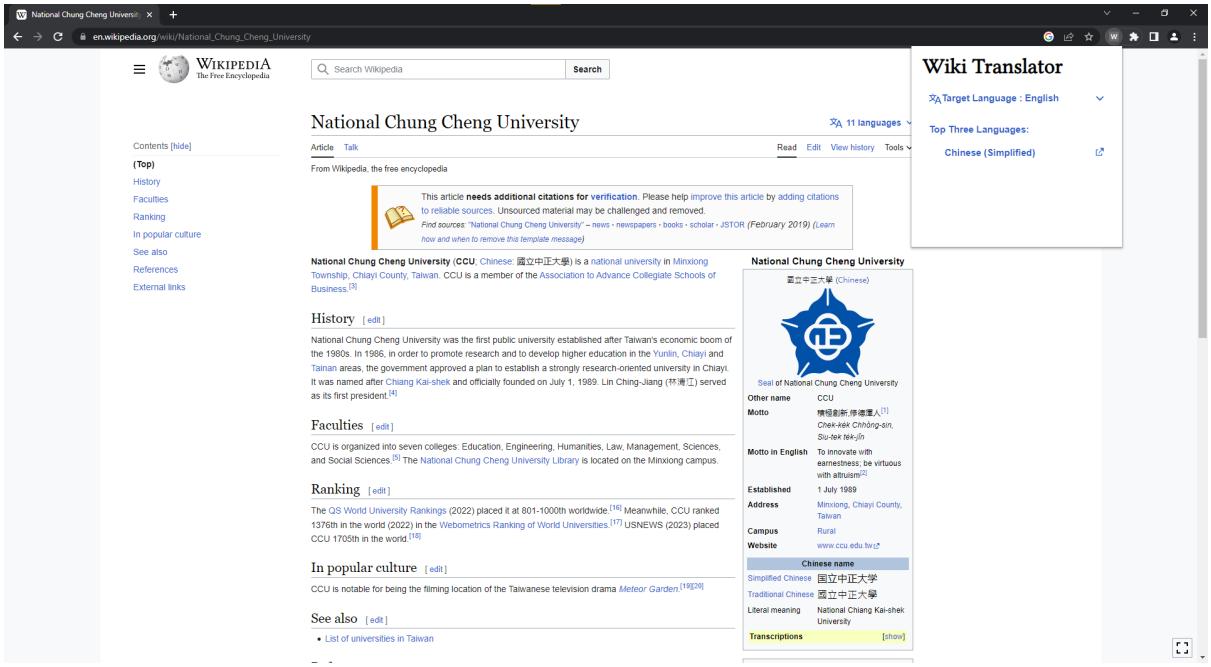
Figure 4. Installation step 4



Usage

1. Go to any Wikipedia article and open the extension's popup. The target language will be automatically selected to be the language of the current article, but you can change it yourself.

Figure 5. Usage step 1

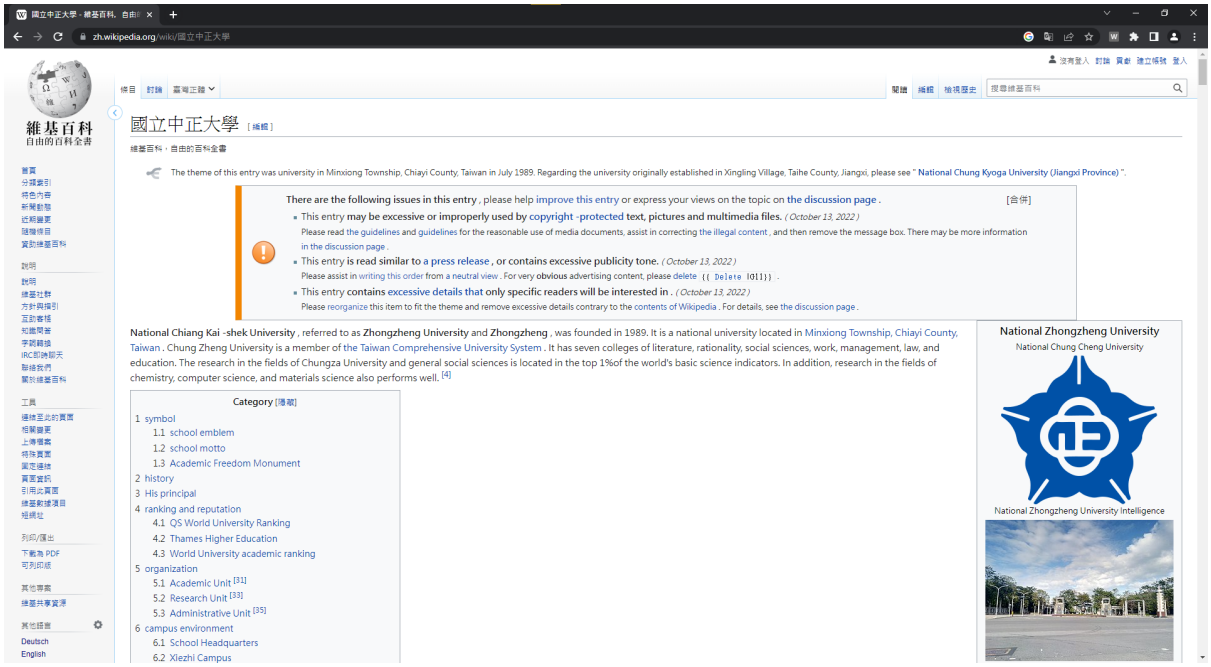


2. The “Top Three Languages” list will show the top 3 longest language versions(when they are translated into the target language). This list does not include the target language version nor those language versions that, after translation, become shorter than the current version. Note that in this case, “Chinese (Simplified)” is displayed in the list. In fact, this list will never contain “Chinese (Traditional)”, because Wikipedia only supports “Chinese”, as in, you cannot store separate Wikitexts for Traditional Chinese and Simplified Chinese. There is only one kind of “variant-agnostic” Chinese, and the conversion between different Chinese variants is handled by “使用說明: 中文維基百科的繁簡、地區詞處理” (n.d.). The choice to show “Chinese (Simplified)” in the UI is simply an arbitrary one.

Figure 6. Usage step 2

3. After clicking on the language version that you want to read, you will be redirected to the translated version of that version. In this case, the Chinese version that is translated into English.

Figure 7. Usage step 3



Problems Encountered

Google Translate API costs money

Google has deprecated its old v1 translation API, which was free. However, after some research, we found that by setting the `client` parameter in the query string of the request to Google Translate to either `gfx` or `chrome-dict-ex`, the API becomes free. For example, `https://translate.googleapis.com/translate_a/t?client=gfx&format=html` is the "base" URL we use when making requests to the Google Translate API, in the sense that it doesn't include the `sl` and `tl` parameters.

Google Translate API gives weird 411 errors

You can put the string to translate in the `q` parameter when making requests to the Google Translate API. However, we quickly found that after that string exceeded some 11000 characters, the API would issue the `411 Length Required` error, even though we did include `Content-Length` in the header. Later we found that if we put the string to translate in the request body as `x-www-for-urlencoded`, then the API would translate strings up to 1 MiB in length.

Webpack repeatedly packs already-packed files

We used TypeScript for this project, and also ESM, so to build to project, we had to first compile the TypeScript sources to JavaScript, and then bundle them with Webpack. At first, we made `tsc` compile all files under `src/` and put the output in `dist/`, and made `webpack` bundle all the compiled files under `dist/` and put them back into `dist/`, replacing the old unbundled files. After some time, we noticed that sometimes, the source maps would stop working, and the bundled files seemed to be getting a deeply nested structure, and were also increasing in size at the same time. We eventually found that this was because we were using the `--watch` mode of `tsc`. This mode automatically recompiles source files that have been changed and doesn't recompile the unchanged ones. But `webpack` always bundled all files under `dist/`. This caused the files that were not recompiled to get repeatedly bundled. This meant that the source maps didn't stop working, they just showed the bundled source code, because they had been bundled 2 or more times. The fix to this was to separate the output directories of TypeScript and Webpack. We made it so that `tsc` would output to `distUnbundled/`, and `webpack` would bundle the files under `distUnbundled/` and output to `dist/`.

Changing the target language too fast causes duplicated top language buttons

When the user selects a target language, the top languages list will be updated with an `async` function. However, if the user chooses another language before the top-language buttons show up, then the list will show duplicated languages. Even though the function responsible for updating the list clears the list, it does so before making requests to Wikipedia and Google Translate, so if that function is called multiple times too fast, they will all clear the list at roughly the same time, make the requests, which are asynchronous, and add entries to the list at roughly the same time, thus causing this bug. To fix this, there had to be a way to cancel that function if the user chose another language before it had time to finish. We used `AbortController` to add the cancellation feature.

Styles incorrect after replacement

If an article's original language is L_1, but is replaced into an article whose original language was L_2, and $L_1 \neq L_2$, then the styles for that article break. This doesn't happen, for example, if the user is on the English version of the page, selects Chinese as the target language, and clicks on English, because in this case, $L_1 = L_2 = \text{English}$. We tried to import the stylesheets of the translated article into the current page ("API:Styling content", n.d.), but it didn't fix the problem. We thought maybe we could redirect the user to the language that is the original language of the translated article, and then replace the translated article into the new page. However, Chrome doesn't provide an easy way to wait for the new page to finish loading, so our content script would immediately try to replace the article even before the new page was loaded. We ended up having to use `setTimeout` to wait for 5 seconds after changing to the new page and then replace. This felt like an ugly hack and I am sure there are better ways to do this.

Thunder Client doesn't support Unicode

When we were making test requests to Google Translate to make sure our method worked, we were quite worried because Google Translate seemed to refuse to translate any non-Latin characters. It turned out that it was just because Thunder Client, a Visual Studio Code extension that I used for testing the API, didn't seem to support Unicode characters, and would just remove them from the request. We ended up using Postman.

Prospect

Remembers last selected target language

It would be nice if the extension could remember the last selected target language instead of always choosing the language of the current article.

Put the most used languages at the top of the target language selector

Right now the target language selector sorts the languages alphabetically. Wikipedia, for example, doesn't do this, and puts some of the most used languages in the world at the top. It

would be nice if our extension could do this, too.

Translate articles longer than 1 MiB

If you request the Google Translate API to translate a string longer than 1 MiB, it will return an empty string(but with a status code of 200). One way to circumvent this is to recursively extract each text node from the HTML source and translate them individually. In fact, this is what we originally planned, but later discarded because we thought it was too difficult.

Division of Labor

1. 吳秉倫: Programmed the main logic and made this report.
2. 廖翊廷: Programmed the UI.
3. 陳立峰: Prepared the presentation slides and made the presentation.
4. 楊立宇: Prepared the presentation slides.

References

API:Styling content. (n.d.). Retrieved from https://www.mediawiki.org/wiki/API:Styling_content

Chrome Extension Ideas. (2023, February 18). A chrome extension that detects which version of a given wikipedia page is the longest. Retrieved from https://twitter.com/ExtensionIdeas/status/1626892782534283264

使用說明: 中文維基百科的繁簡、地區詞處理. (n.d.). Retrieved from https://zh.wikipedia.org/zh-tw/Help:%E4%B8%AD%E6%96%87%E7%BB%B4%E5%9F%BA%E7%99%BE%E7%A7%91%E7%9A%84%E7%B9%81%E7%AE%80%E3%80%81%E5%9C%B0%E5%8C%BA%E8%AF%8D%E5%A4%84%E7%90%86