A) Our first job was to extract the username from the LeetCode ranking page. I searched the internet for whether LeetCode has any api that I can use to extract live contest ranks of all users

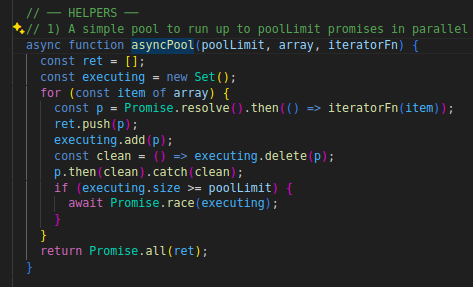
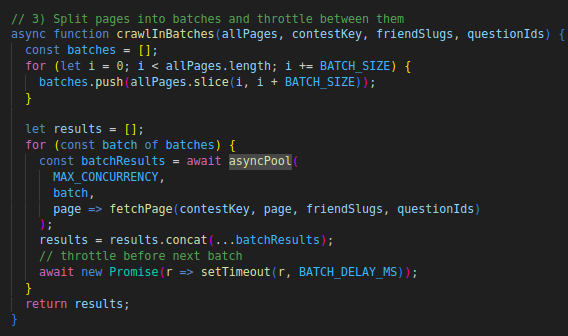
i) We heard that using leetcode.com/graphql we can fetch some important info, but that didn't include global rankings

ii) We thought of another option of directly making the fetch to LeetCode rank webpage(1st rank page to 500 etc, rank page) and among the different users, save the friends element html and directly use that to display the results, but This approach was not feasable since leetcode's frontend is a spa made most probably using react. Therefore, using fetch(https://leetcode.com/contest/biweekly-contest-156/ranking/2/?region=global\_v2) will return only the static part of index.html, and hence, dynamically rendered parts like the rank table, etc, will not get returned.

iii) Then we went to inspect->network section✅ for the leetcode global ranking webpage for a particular contest, and observed that it made api request to "https://leetcode.com/contest/api/ranking/weekly-contest-449/?pagination=${page}&region=global\_v2" for this particular webpage. Moreover, we above API didn't require api keys, etc.

iv) Each page had a global ranking of 25 users. We observed that it takes around 10 minutes to fetch the ranking of 500 pages (i.e, 12,500 users) if we request pages one by one. 10 minutes is a very long time, no user will wait for 10 minutes to get the ranks of their friends.

v) Therefore, instead of making the next request once the previous fetch request is completed, concurrently executing multiple asynchronous tasks, ensuring that we can reduce the time to fetch all the pages significantly. The portion of code that I used for the same is attached below:

Moreover, in order to make sure the total time to fetch all pages doesn’t get increased by rate limit errors. We divided all the total pages into batches, and between 2 subsequent batches, we had a gap of BATCH\_DELAY\_MS. Also we made sure that if request to a particular page is rejected due to certain reasons then we will not go to the fetching next page, rather we will wait for 1sec and then again make a fetch request for the same page, This ensures that at the end we have the valid responses for all the pages. Moreover, since most of the serious LeetCode contestants whom we would like to follow generally have contest ranks of less than 12,500, we fetched the results for the first 500 pages (i.e, 12,500) users only.

In the above code, when we set poolLimit(=6) and batch\_size(=250), BATCH\_DELAY\_MS(=6sec), we found the time to fetch all the pages to be minimum(=90sec). Therefore, in our extension, we used this value only.  
Note: i) poolLimit in the above code is the maximum number of promises that can run concurrently.

Later, we work on improving the time to fetch all the results using techniques like exponential backing, using multiple IP addresses via proxies to bypass rate limit issues, and making requests via [background.js](http://background.js) instead of content.js

Now we can move to the main part, i.e, building the full-fledged extension. The requirements from our side are:

I) The code inside [Content.js](http://content.js) should include:

1. Save friends' usernames, user\_slugs in Chrome local storage whenever their profile is star-clicked.(exact name of classes of div tag used for each of these will be given when we will implement this specific part)

2. Fetching the ranks of users for the given contest as soon as the link satisfying the below format is opened: [https://leetcode.com/contest/api/ranking/${contestKey}/\*](https://leetcode.com/contest/api/ranking/$%7BcontestKey%7D/). Then save the details in Chrome's local storage corresponding to each contest, so that whenever the page is reopened then the stored details are shown before ranking is calculated

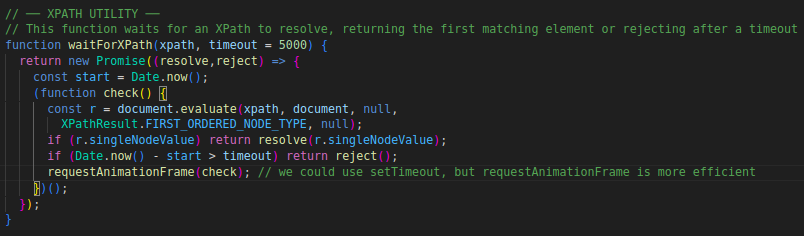
3. Verifying whether the given user is a friend during the ranking page checking, check whether the user exists via "user\_slug" instead of username, since it is the unique identifier

4. Create a button between the global and LLM button named "Friends", and when the user clicks on it then it will show rows of friends' performance who participated in this contest.

5. While displaying the rank of friends in the friends section, display the username, rank, along with other details like total score, finish time(including penalties), submission time for each question that are visible to any other user in the rank page style present in the global section.

Now, some important points regarding the implementation of these features in our extension for the [content.js](http://content.js) part:

A) For the injecting star for friends part, the challenge that we faced here was that styling for the div class was done by LeetCode using Tailwind CSS, due to which the div element can't be accessed via their IDs. To obtain the username, userslug, we observed that there is only one span element with text exactly "Rank". So we used a custom waitForXPath() function to return the first matching element or reject after the timeout. Since, directly using document.evaluate() will not work properly for a dynamic Application like leetcode.com.

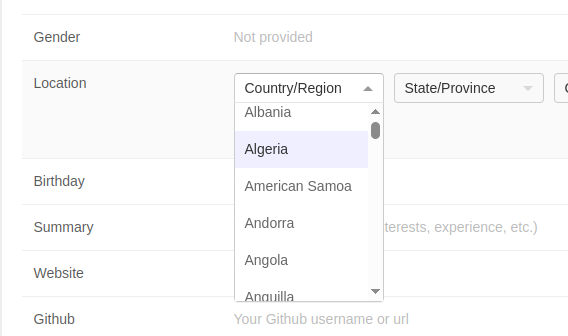


Note: we could have used the overall path present in the inspect ->”copy\_overall\_path to the element” as well to locate the element, but since that is more prone to any change in html structure of the webpage, we stuck with the current way

B) While displaying the rank, username, etc, for each row in the friends’ section of the webpage. Columns like rank, username, and score were directly obtained from Chrome local storage, but

i) Time for finishing different questions was in Unix format, so we converted that to IST format and subtracted it from the start of contest time to get time in hr:mm:ss format

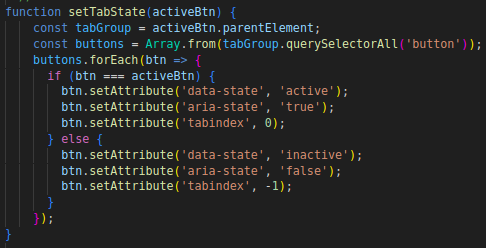
ii) while adding programming language icon next to submission time for each submission, leetcode has coded SVG images from scratch code to represent coding languages images for global rankings page, But we will simply used SVG images downloaded and change size in case if we need it, because there are over 15+ languages allowed by leetcode therefore it is not worth it to code svg image for each language.

iii) Next to the username, we had provided the flag, the user’s country. The saved friends ranking part of Chrome local storage had a “country” key, which had the country name, but that didn’t include the country emoji. We needed to explicitly map country names with their corresponding emoji. We went to the LeetCode user profile page, and went to inspect->network to find the GraphQL query response, which included all the countries' names. (Properly mapping name to emoji was important because, for eg, in LeetCode, countries like the USA are named as “United States”, but if we didn’t know about this, then we could have a USA flag with United States of America)  


iv) Make sure that consecutive rows have a different color contrast, similar to the standard LeetCode global rank table

v) We added 1 to the rank given by Chrome local storage since the rank given by api was 0-based.

C) When the Friends button is used to get pressed, then the corresponding div elements of the LLM button and the Global button are used to get hidden. Along with that, the properties, namely data-state, aria-state, and tabIndex, are also updated accordingly to make sure the current page’s button is highlighted



II) The code inside [popup.js](http://popup.js), popup.html, and popup.css should make sure:

a) If there is no current friend present in the Chrome local storage, then it will display "There is no current friend."

b) Else, the username, user\_slug, contest rating, and last submission date of the user are displayed row-wise. Also, if we click on the row, then it will redirect us to the user's LeetCode account(URL for his LeetCode account is: "https://leetcode.com/u/user\_slug/")

Some important points regarding the implementation of these features in our extension inside the popup.js:

i) The styling for popup.html is done using popup.css, and the content is rendered dynamically using code written in popup.js

ii) The latest rating and last submission of the user for [popup.js](http://popup.js) was obtained using the GraphQL request to leetcode.com/graphql API. Corresponding query:  
