IERG4999J Final Year Project II

Topics in Internet Content Exploration

BY

LO, Siu Kit
1155147592

A FINAL YEAR PROJECT REPORT
SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF BACHELOR OF INFORMATION ENGINEERING
DEPARTMENT OF INFORMATION ENGINEERING
THE CHINESE UNIVERSITY OF HONG KONG

April, 2023

Table of Contents

Abstract

Searches we do on the internet with mobile devices are often unintuitive and inefficient. To overcome this problem, we have built a cross-platform mobile application that integrates multiple search platforms, enables searching by images, and intuitively displays the results. Webpage contents are extracted for search query refinements, delivering more accurate, related, and useful results for users. The mobile application can provide users with a more convenient, intuitive, and efficient searching experience that returns optimal results.

1. Introduction

1.1. Background

We always search on the internet, especially on Google, so much that it has been the most visited website since 2010 [1]. At the same time, mobile devices account for almost 60% of the network traffic [2]. It is no doubt that most of us conduct countless searches on the internet using mobile devices instead of traditional desktop computers nowadays. Intuitively, if ones cannot find their desired results from a single search, they try to change the search query or go for another search engine. However, it might be hard to perform these actions on touch-based mobile devices in an efficient manner. Moreover, we usually mess with the original search queries attempting to know more about a specific thing, but often make it worse. It is also common for people to have no idea on what the search query should be just by looking at something they are interested in. In short, we need a way to conduct searches in a more intuitive and efficient way on mobile devices, preferably with specific yet concise keywords. At the same time, the search input should not be limited to text only.

1.2. Description of Problem

Searching on the internet sounds simple and easy. We type in the keywords and click on the results that interest us. If we want to view another result, we go back and click on another result. On desktop computers we can easily open multiple browser tabs with a single mouse click. However, this is not the case for mobile devices. The most intuitive and efficient way of interacting with mobile devices is certainly swiping and at least clicks as possible.

It is also often that we want to drill-down more on the current topic. The usual way we do this is to perform a new search with lengthened query. Problem with this approach is that sometimes the query is unnecessarily lengthened while the usefulness or relatedness of the search results drop. Leading us back to the problem of unspecific and unconcise search queries.

The above are based on the assumption that the users know how to construct a search query. In fact, it is not always the case. It certainly would be convenient if we can search with image such that no keyword is required. In fact, searching with images is nothing new. Google Images has been there for years. Till now, the process of searching with images on mobile devices is still neither satisfying nor user-friendly, users have to download a dedicated app for that as it is not available within the mobile browser, at least not without some little tricks.

1.3. Motivation and Project Idea

In light of the aforementioned problems, we would like to design a solution that provides a better searching experience for users with touch-based mobile devices. Specifically, a mobile application that integrates different search platforms, comes with auto search query refinements, and allows searching with images.

Users can perform searches in their usual ways, with the addition of being able to search on multiple platforms with ease, enhancing their efficiency and productivity. Moreover, users can search with auto-refined queries without going through the hassle of manually tuning them. Last but not least, everyone can perform searches with images much more conveniently.

1.4. Related Works

It is not an entirely new problem. Previous attempts to solve the problem exist, some of which are available on the market.

1.4.1. Existing Mobile Applications

Mobile applications that partially solve the problems exist on the market. For instance, *Search-It* and *Search All* on App Store, *Smart Search & Web Browser* on Google Play. Yet, these apps only simplify the process of manually searching on multiple search engines. The results are still not being extracted and displayed directly, the search queries do not go through any refinement, and the search input is still limited to text only.

1.4.2. Results Post-processing

Modifying the search query is not the only way to provide better search results for users. Directly modifying the results could be another way to solve the problem. In fact, it is suggested that the results of search engines can be optimized by returning the more relevant and user desired pages on the top of the search result list [3].

This approach involves four steps. Firstly, a similarity analyzer calculates the similarities based on the search query and user feedback. Secondly, a query clustering tool groups queries into same cluster if their similarity value is above pre-defined threshold. Thirdly, a pattern generator takes as input the query clusters and finds the sequential patterns in each cluster. Finally, a rank updater that takes its input from the query processor, i.e., the matched documents of a user query, and then modify the rank score of the returned pages. After all these steps, we get an updated search results list that should fit user's requirements better.

However, there are limitations to the approach. Lots of data are required to perform the analysis and clustering, such as user search records and view records. In other words, we must collect and store user activities, which is not economically feasible and might raise privacy concerns.

2. Methodology

To solve the previously mentioned problems, a mobile application that "truly" integrates multiple search platforms is built. Specifically, users can see the results from different search engines directly after they have entered the keyword or uploaded the photo, instead of having to pick. To minimize the overall development costs and to push the final product to more people, the application is built as a cross-platform mobile application.

2.1. Use Cases and Overview of Implementations

There are multiple ways and reasons we search on the internet. Below are some of the most common use cases and an overview of the proposed implementation.

2.1.1. General Searches

This is the main focus of the report. Under current search method (in browser), users first input some queries, the results are then displayed as a list. Users then click into any result they find interesting or related. If they want to go to another result, they first need to go to the previous page (list of results), then click on another result. To search on different platforms, they must repeat the searching procedures again.

Instead of repeating these steps, our application searches the queries on multiple general search platforms, including Google, Bing, and DuckDuckGo. Then, merges and re-ranks the results into a single list.

Rather than displaying the results as a list for users to pick from, our application loads the result (webpage) directly, users go through different results and platforms simply by moving the joystick-like controller, which is essentially swiping. The number of clicks is reduced to minimal.

2.1.2. Search for Videos

This is essentially the same as general searches, except the search platforms used are video-oriented, including YouTube, Bing Video, and Vimeo.

2.1.3. Search for Social Networking Sites (SNS) Contents

Similar to video searches, this is essentially the same as general searches, except the search platforms used are more SNS-oriented, including Facebook, Instagram, Twitter, and LinkedIn.

2.1.4. Drill-down Searches

Drill-down search refers to searches that users perform based on current search result. For instances, a user searched XYZ and is looking at a webpage that contains information about **XYZ_1**. S/he wants to know more about **XYZ_1**. Then, s/he manually searches "XYZ **XYZ_1**" so that results about **XYZ_1** based on XYZ are returned. We call this whole process a drill-down search.

Instead of having users to manually come up with new search query, our application refines the query for users semi-automatically. The content of currently displaying result (webpage) is analyzed to produce or suggest possible new search query.

2.1.5. Search by Image

Other than manually coming up with a search query, users can also make one by image. It could be pictures on device or photos taken at the moment. This search method makes use of the Bing Image Search API, which returns a list of related images and their corresponding label. Then, users can select the one they want. Then, the selected label will be used as the search query to perform searches.

2.2. Different Search Modes

Two search modes are implemented in the application, namely *plain* and *drill-down*.

2.2.1. Plain

As the name suggests, this is the plain way of performing searches. This is also the default mode when users enter their own search queries. Moreover, search queries under this mode do not go through any refinement.

2.2.2. Drill-down

This mode can be used after the initial search. New search is performed upon the trigger of a drill-down. The search query is refined with information of the currently displaying webpage. For instance, title of the webpage, context of the webpage currently displaying within the device's screen, or whatever content on screen that the users have selected. The detailed implementations will be covered later in *Search Procedures* section.

During the refinement, these are analyzed to look for possible related keywords. The refined search query is expected to return more accurate and related results.

2.3. Current UI & Flows of the Cross-platform Mobile Application

The targeted users of the app are every one that searches online, that includes children, teenagers, adults, elderlies, etc. Thus, the user interfaces should be as intuitive as possible. The application currently consists of four main pages, home page, search page, result page, and settings page.

On the home page (figure 2.1), users can tap on the little magnifying glass icon in the top right corner to open the search page. They can also access the settings page or take a tutorial via the menu drawer (figure 2.2), which can be opened by tapping on the 3-lines button in the top left corner.

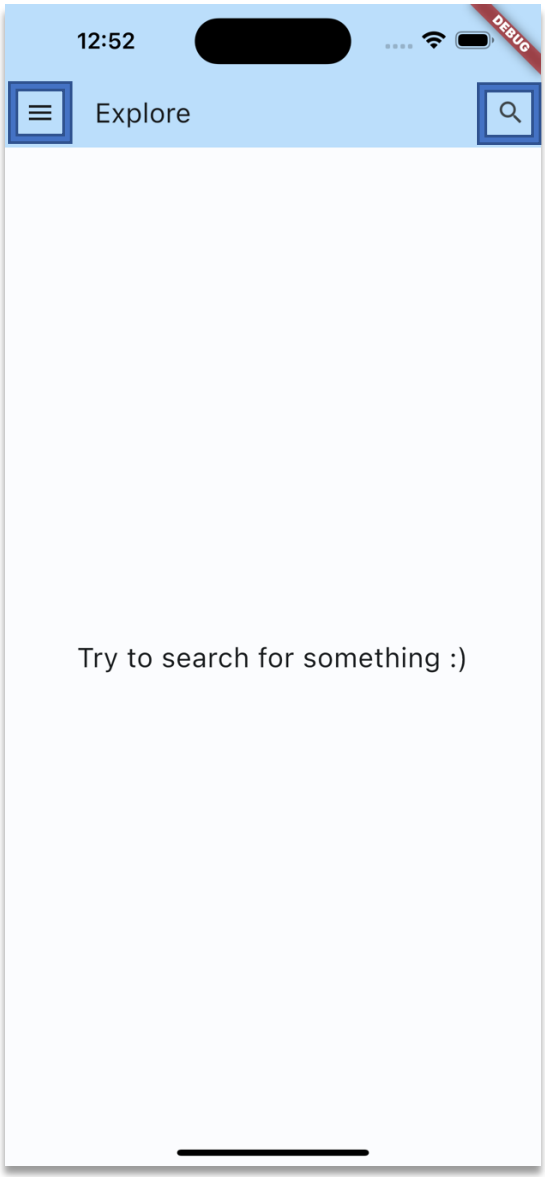


Figure 2.1. Home Page.

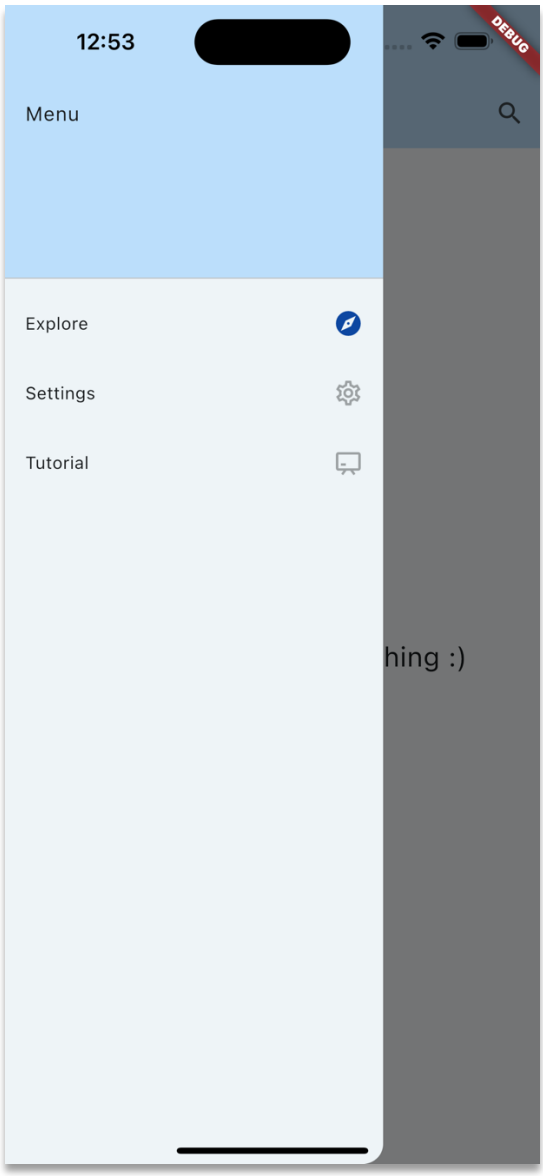


Figure 2.2. Menu Drawer.

Then, in the search page (figure 2.3), users can select the search type. They can then either enter their search query or upload an image from device's library or camera. If an image is uploaded, a list of related images with their corresponding label is shown (figure 2.4). Users can then select any one of the labels to be the search query. After that, the search procedures begin. Results will be displayed on the results page when the search is done.

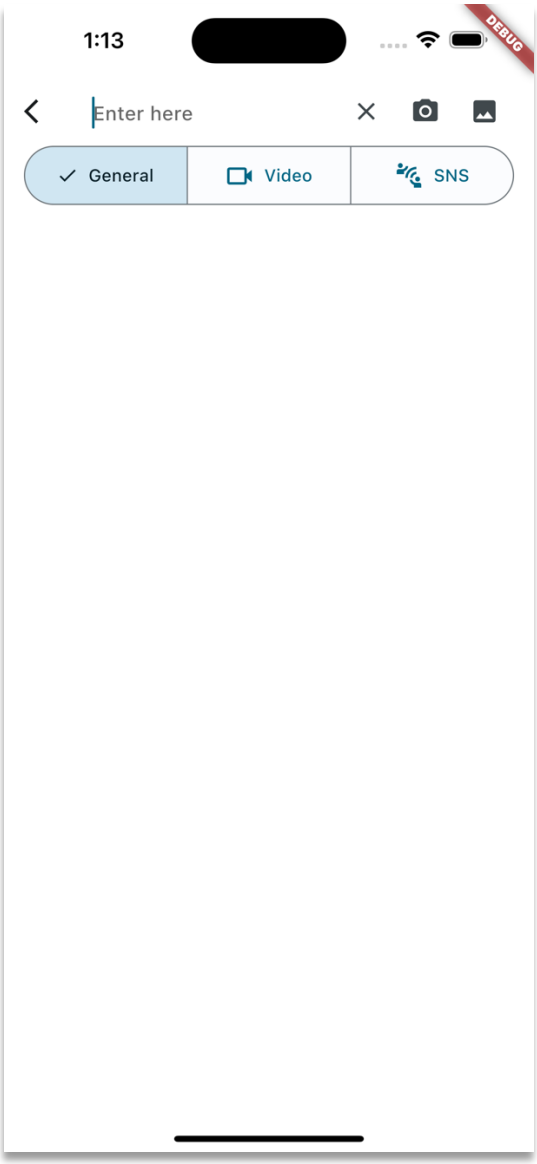


Figure 2.3. Search Page.

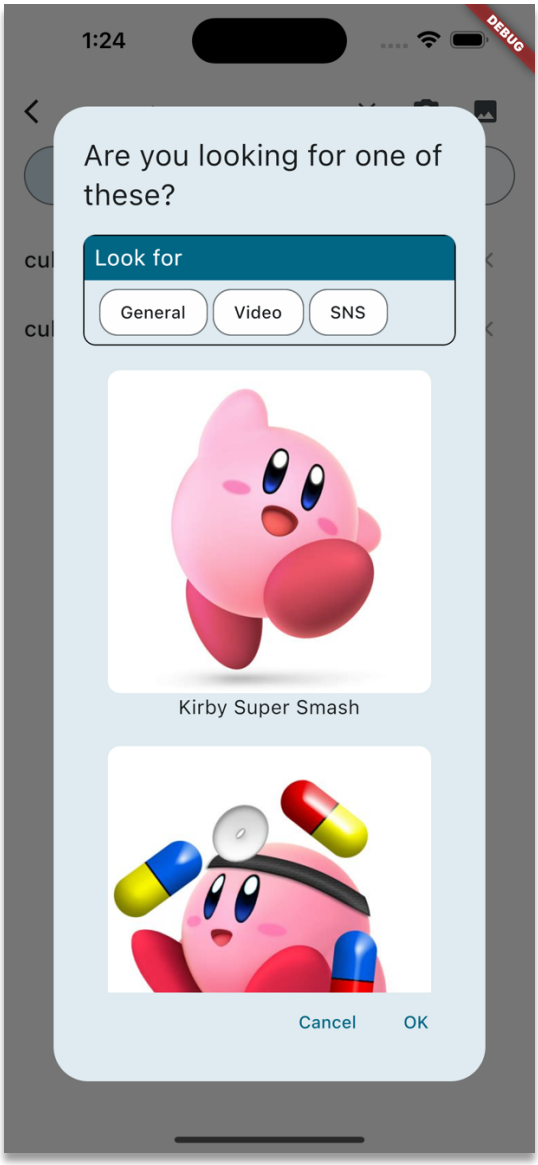


Figure 2.4. Search Page (Selecting Search Query).

The results page (figure 2.5) consists of a InAppWebView widget (in-app browser) for loading the URL from the search results. The bar below the in-app browser is for quick controls, such as opening the drill histories modal, going to the first result, going back, and sharing current result.

There is a joystick-like controller (blue arrow in figure 2.5) above the middle of the bottom bar, which can be dragged horizontally and vertically to navigate through results and search type respectively (figure 2.6).

There is also a draggable floating action button (FAB) with a drill icon near the bottom right corner (green arrow in figure 2.5), which is used for pinpointing the content that users want to drill on. When dragging the FAB, an arrow is shown for precise content selection (figure 2.7). Users can place the arrow wherever they prefer within the webpage. Dropping the arrow in red zones cancels the drill action.

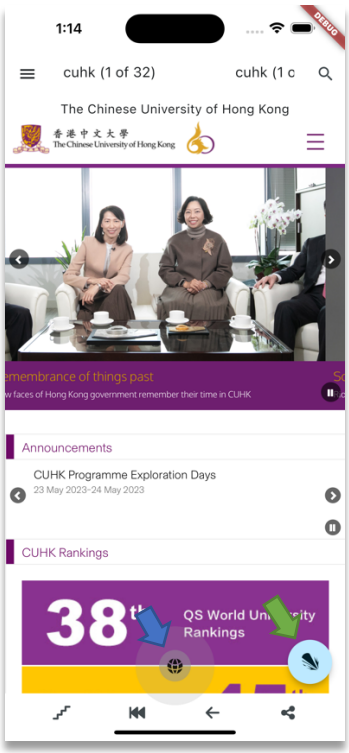


Figure 2.5. Result Page.

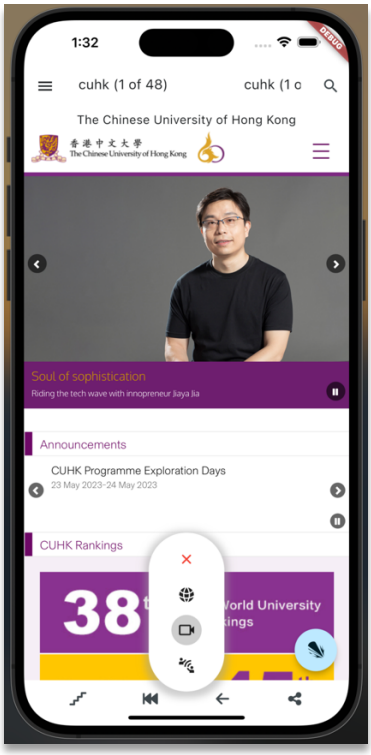


Figure 2.6.
Switching Search Type.

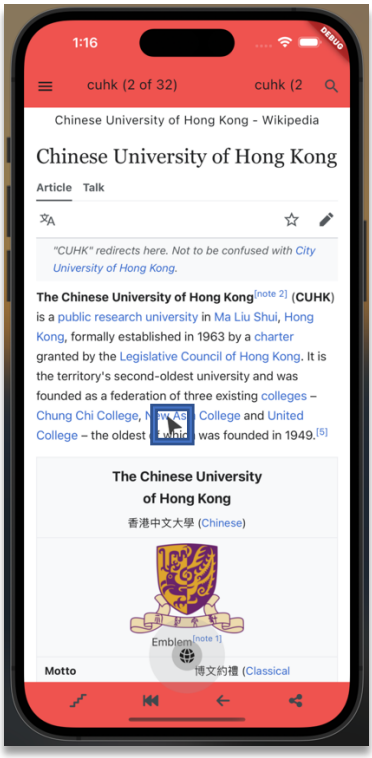


Figure 2.7.
Drag-and-drop Drilling.

The extracted content by drag-and-dropping might not be suitable for direct searching, e.g., too long. Some pre-refinement will be done, such as possible keywords extraction, which will be covered later in the *Searching Procedures* section.

After the extraction is completed, users can select one or more keywords from the popup (figure 2.8), confirm the search type and begin the search.

Depending on the search type and merge algorithm, users may be able to filter the results by their uniqueness (figure 2.9).

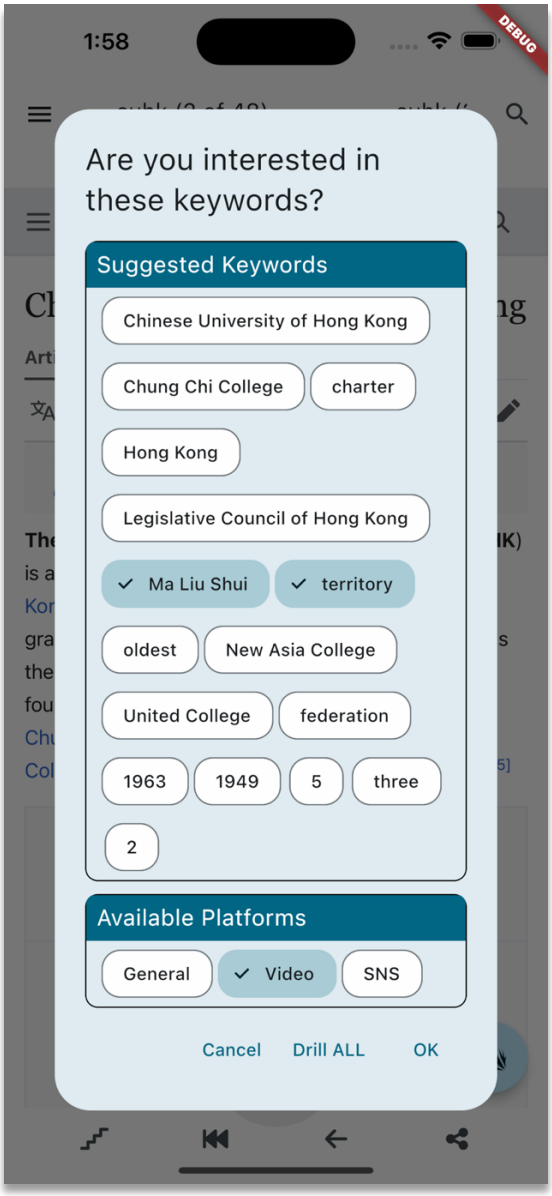


Figure 2.8.
Possible Keywords Selection Popup.

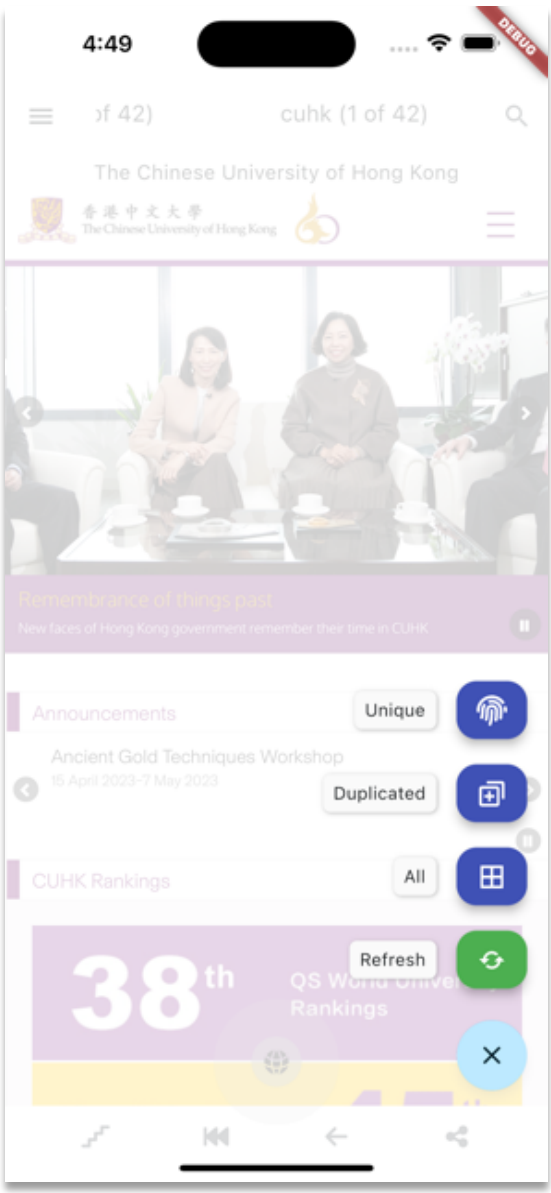


Figure 2.9. Toggle for Unique Results.

The search histories modal (figure 2.10) lists all searches performed by the users in the current search session. A new session is started when users re-open the app. Drill-down searches are prefixed with a special icon. Users can go back to a particular search by clicking on the item.

The settings page (figure 2.11) allows users to tweak settings, including enablement of result preloading, reverse joystick control, method of extracting the search query for drill-down, and algorithm for merging and ranking the search results from different platforms.

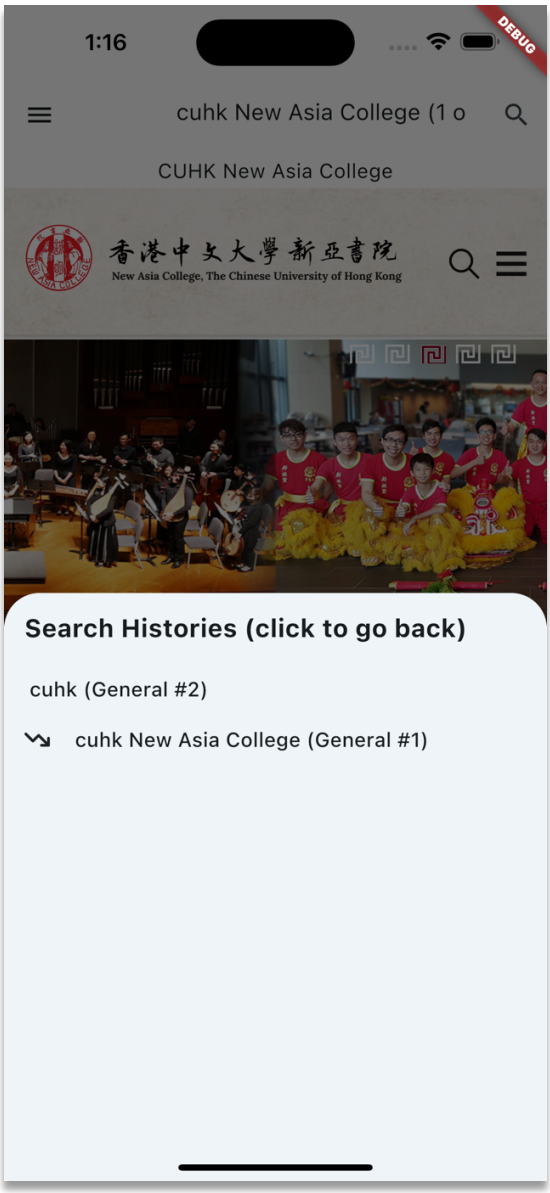


Figure 2.10. Search Histories Modal.

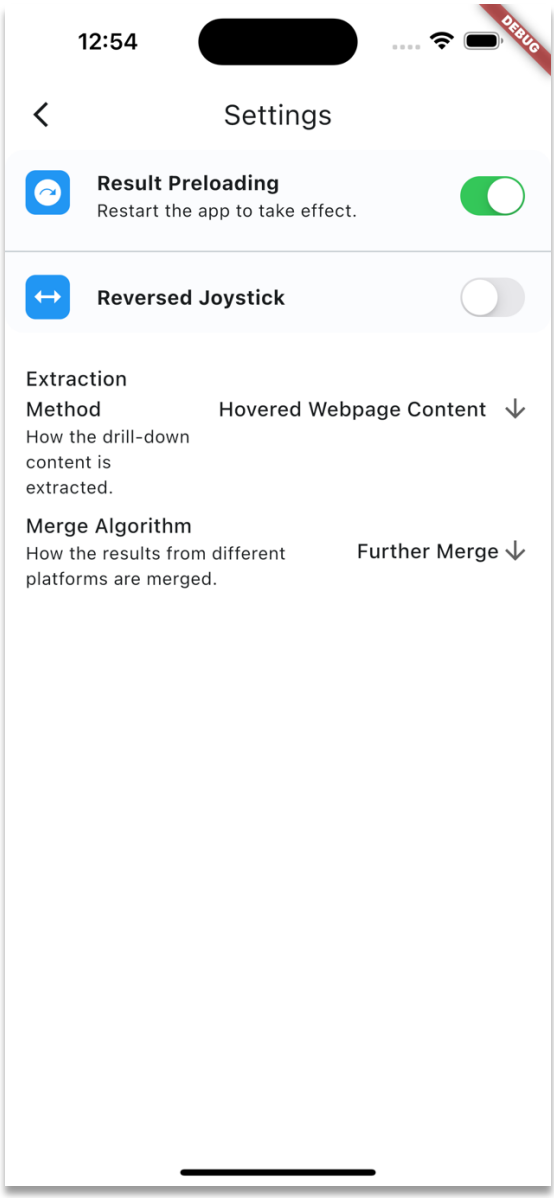


Figure 2.11. Settings Page.

The flow chart below illustrates the basic flow of the mobile application.

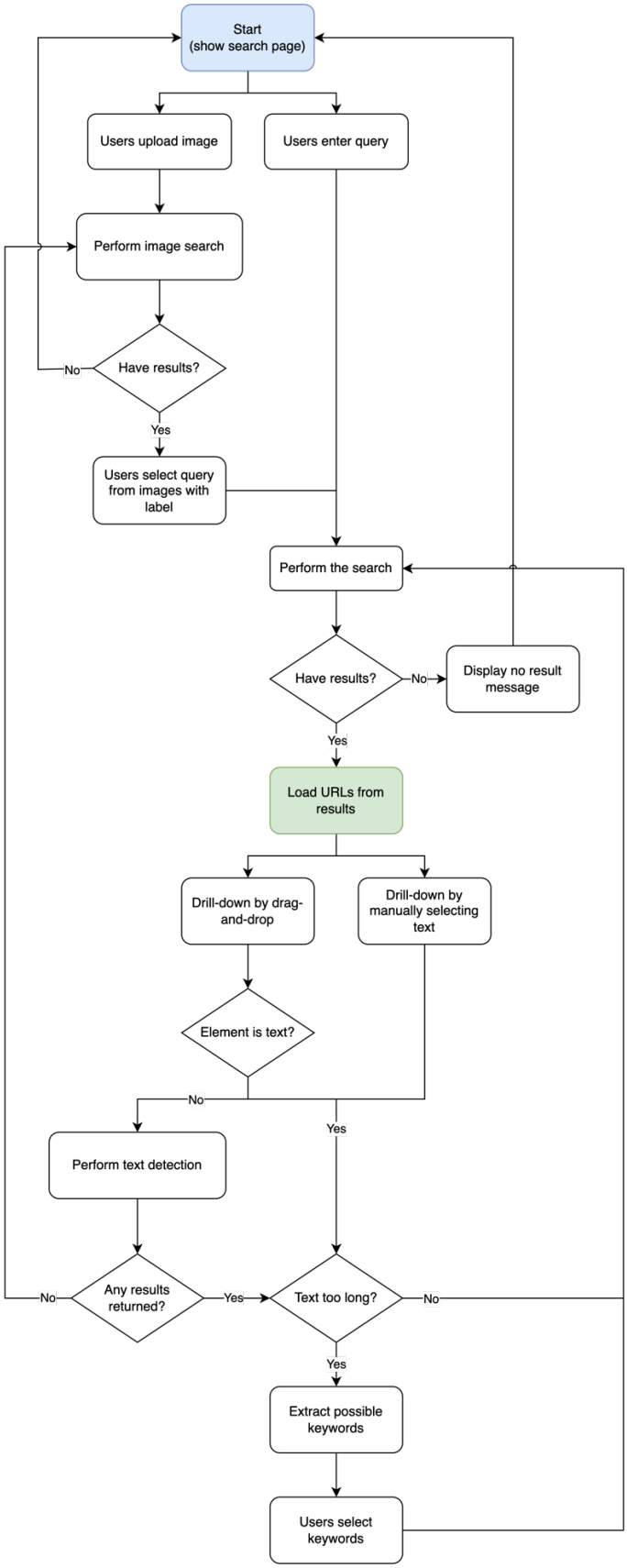


Figure 2.12. Flow Chart of the Mobile Application.

2.4. Framework Used

Several commonly used frameworks are available, such as Ionic, React Native, and Flutter. All of them come with many pre-built UI components and the ability to compile the same codebase into Android and iOS apps. React Native is better than Ionic when building cross-platform applications [4]. After careful consideration and thorough testing, Flutter is chosen to be used in this project because of its better performance [5] [6].

2.5. Searching Procedures

The searching procedures can be separated into three parts, constructing the search query, sending out the search request, and decoding the response.

2.5.1.Constructing the Search Query

For plain searches, the search query is simply what the users have entered. For drill-down searches, the search query varies depending on the extraction method chosen by users.

There are multiple ways to construct search queries based on the currently displayed webpage. Most methods require running JavaScript within the webpage; thus, JavaScript handlers have been set up for communication between Flutter code and the webpage.

2.5.1.1. Title

Only the title of the current webpage is used as it is supposed to be the most concise description of a webpage [7].

2.5.1.2. Webpage Content

Webpage content refers to the inner text of the HTML element currently in the middle of the visible portion of the webpage. The rationale behind is that if an element is within the screen, it must be visible to the users. We believe it makes sense to assume that users are more interested in what is on the screen than what is not.

2.5.1.3. Title With Webpage Content

This mode combined the two abovementioned modes to construct a more sophisticated search query.

2.5.1.4. Hovered Webpage Content

With the draggable FAB at the lower right corner on the result page, users can pin-point the exact content they want to drill-down. The positional information of the FAB will be converted to match up with the in-app browser. Then, the inner text of the exact HTML element that users were aiming at can be extracted.

2.5.1.5. User Selected Text

The above auto-extraction methods might include information that the users might find unnecessary. Thus, they can also long-press to precisely highlight whatever text they want for a more granular selection.

2.5.1.6. Safeguard Measures

Different search platforms have their own limitations for search query length, for instance, Google limits it to 32 words. It is possible for the extracted text to be too long, e.g., it might be a long paragraph. This happens especially frequent if users use the auto-extraction methods.

If that is the case, the app will try to extract possible keywords using Google Natural Language API, which returns a list of keywords ranked by their respective salience within that piece of text. Users can then select one or more keywords to perform the search.

Sometimes it might not be possible to extract text directly, as the element could be an image, or the webpage might have deliberately disabled highlighting. In this case, the app will take a screenshot of the currently visible portion of the webpage and perform text detection with Google Vision API, which returns a list of text it recognizes from the provided image. If the detected text is too long, it will be further processed with the abovementioned Google Natural Language API. In the end, users can then select one or more keywords to perform the search.

2.5.2. Sending Out the Search Request

After having the search query, the application sends out a HTTP request that contains the search query.

Depending on the platforms, requirements for constructing the request might vary. Ideally, we would make use of the API provided by each platform. However, not all platforms have their own API for our usage. In those case, we either uses the Google Search API with specified domain names to mimic the process of searching on different platforms with their own APIs or get the results by web scraping.

2.5.2.1. Google

A typical Google Search API request is constructed by a pre-defined API key, a pre-defined search engine ID, the search query, and the page number that the results should start from.

We have applied and set up a unique search engine ID for each platform beforehand, such that when the specific search engine ID is used, we are essentially "searching on that platform". This is mainly used for platforms that do not provide their own API.

```
uri = Uri.https('www.googleapis.com', '/customsearch/v1', {
  'key': API_KEY,
  'cx': SEARCH_ENGINE_ID_GOOGLE,
  'q': value,
  'start': _start.toString(),
});
response = !_gg ? await http.get(uri) : null;
```

▲ Codes for sending out the search request with Google Search API

2.5.2.2. Bing

Bing provides a Web Search API similar to that of Google. It is constructed by search query, custom configuration pre-configurated, market, and authentication.

Here, the market is set to "zh-HK", which represents Hong Kong. By doing so, the search results should be similar to searching on Bing's website in Hong Kong. This could potentially be changed according to users' location to enhance to the relatedness of the results.

```
response = !_gg
  ? await http.get(
    Uri.parse(
'https://api.bing.microsoft.com/v7.0/custom/search?q=$value&customconfig=6c099879-5079-4eb0-be77-694fffc16ddc&mkt=zh-HK'),
    headers: {
      'Ocp-Apim-Subscription-Key':
      'f22755e26efb48f9ad6fa930df61f1cc'
    },
  )
: null;
```

▲ Codes for sending out the search request with Bing Web Search API

2.5.2.3. DuckDuckGo

DuckDuckGo does not provide web search API like Google and Bing do. Mimicking searches on DuckDuckGo using Google Search API does not seem to be working well. Considering the simple layout of DuckDuckGo website, we decided to get its results by web scraping.

The request URL is the same as when we were to use DuckDuckGo on browser, which consists of the search query.

```
response =
  await http.get(Uri.parse('https://duckduckgo.com/html/?q=$value'));
```

▲ Codes for sending out the search request to DuckDuckGo

2.5.2.1. Other Platforms

It is worth mentioning that the app also includes other platforms other than the general search platforms above.

For video search, YouTube API, Bing Video API, and Vimeo API are used.

For SNS contents search, Facebook, Instagram, Twitter, and LinkedIn are mimicked with Google Search API with different search engine IDs.

2.5.3. Decoding the Response

Upon receiving the response, it is decomposed and extracted into a list of results. Each result consists of a title, link, snippet, and their rank on that platform. Below shows how responses from different platforms are decoded respectively.

2.5.3.1. Google

For Google, the decoding process is relatively simple as all results are stored in an array. The app simply adds all results with required information into a new list.

```
jsonResponse['items'] != null
  ? jsonResponse['items'] as List<dynamic>
  : [];

for (int i = 0; i < results.length; i++) {
  items.add({
    "title": results[i]['title'],
    "link": results[i]['link'].toString(),
    "snippet": results[i]['snippet']
        .toString()
        .trim()
        .replaceAll(RegExp(r'(\.\.\.)$'), ""),
    "rank": i + 1,
  });
}
```

▲ Codes for decoding the response from Google Search API

2.5.3.2. Bing

For Bing, the results are similar to Google's, except that there could be links nested under a search result, which are called deep links. We treat these deep links as having the same rank as their parent does.

The app first adds the result with require information into the new list, then check if there is any deep link associate with the current result. If yes, those deep links are added to the new list as well.

```
int rank = 1;
if (jsonResponse['webPages'] != null) {
  if (jsonResponse['webPages']['value'] != null) {
    for (int i = 0; i < jsonResponse['webPages']['value'].length; i++) {
      items.add({
        "title": jsonResponse['webPages']['value'][i]['name'],
        "link": jsonResponse['webPages']['value'][i]['url'],
        "snippet": jsonResponse['webPages']['value'][i]['snippet']
            .toString()
            .trim()
            .replaceAll(RegExp(r'(\.\.\.)$'), ""),
        "rank": rank++,
      });

      var deepLinks = jsonResponse['webPages']['value'][i]["deepLinks"];
      if (deepLinks != null) {
        for (int j = 0; j < deepLinks.length; j++) {
          items.add({
            "title": deepLinks[j]['name'],
            "link": deepLinks[j]['url'],
            "snippet": deepLinks[j]['snippet']
                .toString()
                .trim()
                .replaceAll(RegExp(r'(\.\.\.)$'), ""),
            "rank": rank,
          });
        }
      }
    }
  }
}
```

▲ Codes for decoding the response from Bing Web Search API

2.5.3.3. DuckDuckGo

For DuckDuckGo, the response would be an HTML document. We look for related elements from the HTML document and extract their text, then add them to the new list to be returned later.

It is worth noting that the URL extracted with this method contains parameters that DuckDuckGo uses for internal redirection, which are not needed and are removed during the process.

```
String getActualUrl(String url) {
  // Extract the actual URL from the intermediate URL
  final start = url.indexOf('/l/?uddg=') + '/l/?uddg='.length;
  final end = url.indexOf('&rut=');
  return Uri.decodeFull(url.substring(start, end));
}

// Parse the HTML response
final document = parse(response.body);

// Extract title and URL elements
final titleElements = document.querySelectorAll('.result__title');
final urlElements = document.querySelectorAll('.result__url');
final snippetElements = document.querySelectorAll('.result__snippet');

// Extract title and URL data
final titles = titleElements
    .map((element) => element.text.toString().trim())
    .toList();
final urls = urlElements
    .map((element) =>
        getActualUrl(element.attributes['href'].toString()))
    .toList();
final snippets = snippetElements
    .map((element) => element.text.toString().trim())
    .toList();

for (int i = 0; i < titles.length; i++) {
  items.add({
    "title": titles[i],
    "link": urls[i],
    "snippet": snippets[i]
        .toString()
        .trim()
        .replaceAll(RegExp(r'(\.\.\.)$'), ""),
    "rank": i + 1
  });
}
```

▲ Codes for decoding the response from DuckDuckGo

2.5.3.5. Safety Measures

Though search platforms are becoming more and more powerful, chances are that they might not be able to return anything because of bad search queries, request errors, and etc. In these cases, the app would mark their responses as null, hoping that other platforms would return something useful.

2.5.3.6. Initial Post-processing

The snippet of a search result could be useful for comparison during the merging process, but search platforms often have length limitations on snippets. Thus, the snippets we get are not the complete version and they are often appended with "…" as an indication of an ellipsis. To facilitate the future merging process, the app would check if the snippet ended with "…" using regular expression `(\.\.\.)$` and then replace it with "".

2.6. Results Post-processing

Each platform returns their own list of results, which needed to be further process before the results can be displayed to users. Thus, we need to combine and re-rank these lists. Several algorithms are proposed as follows.

2.6.1.ABAB

This is the simplest ways to merge results. Let say there are three lists of results, A, B, and C. This method first creates a new list, then append a result from A, a result from B, a result from C, a result from A, and so on, until all results are added into the new list.

2.6.2.Frequency

This is an extension of the ABAB method. When appending a result to the new list, it checks if the result already exists in the list by comparing their URL. If yes, the frequency of that result is increased. After appending all results into the new list, it is sorted by the frequency in descending order.

2.6.3.Original Rank

This method takes the original ranking of a result in their own platform into account as well. Before the merging process, a base score is calculated by

$$Base\ Score = \frac{Number\ of\ Total\ Results}{Number\ of\ Platforms}$$

For each result, there will be a score calculated by

$$Result\ Score = \frac{Base\ Score}{Rank\ of\ the\ Result\ in\ Original\ Platform}$$

Then, similar to the *Frequency* method, when appending a result to the new list, it checks if the result already exists in the list by comparing their URL. If yes, the scores of the two results will be added up.

2.6.4. Further Merge

This algorithm takes the *Original Rank* method a step up by further merging similar results. Multiple comparison methods are used to facilitate the process.

The first method is to compare URL regardless of their protocols. Sometimes the same result would have different URL on different platform. For instance, http://admission.cuhk.edu.hk vs https://admission.cuhk.edu.hk.

The second method is to compare the URL in a loose way, then verify if the two results are indeed the same with other information. Specifically, when comparing two URL, instead of looking for exact match, we look for partial match. If a URL is a substring of another URL, it is likely that the longer URL is a more detailed URL that leads to the same or related webpages.

To verify that, we further compare their snippet. As mentioned before, the snippets have been appropriately trim for comparison. Thus, we could loosely compare their snippets. Even though snippet can be generated differently by different search platforms, it is mainly generated by the webpage content or even metadata [8], which is embedded into the HTML and should be the same regardless of platforms.

If snippet of a result is the substring of another result, they are highly likely to be the same result that leads to the same webpage. In that case, we keep the result with longer (more precise) URL, add up their score, and drop the shorter (less precise) result.

2.7. Storing Users Data

This app depends heavily on data returned by the APIs as well as users' own input, it is crucial to store the data in a way that can be utilized efficiently. Different approaches are used for different kinds of data with different purposes.

2.7.1.Fetched Results

The results returned by APIs are used for displaying the results and for users to go back to a particular search. We believe that users would not search for the same thing again within a small period of time. Thus, these data are stored temporarily and gone once the app is closed.


```
"FETCHED RESULTS": {
  "search query 1": {
    "last viewed platform": "platform name 2",
    "platform name 1": {
      "last viewed index": "0",
      "list of results": [
        {
          "title": "title", "link": "url", "snippet": "snippet", "rank": 1
        },
      ]
    },
    "platform name 2": {
      "last viewed index": "1",
      "list of results": [
        {
          "title": "title", "link": "url", "snippet": "snippet", "rank": 1
        },
        {
          "title": "title", "link": "url", "snippet": "snippet", "rank": 2
        }
      ]
    }
  },
  "search query 2": {
    "last viewed platform": "platform name 1",
    "platform name 1": {
      "last viewed index": "0",
      "list of results": [
        {
          "title": "title", "link": "url", "snippet": "snippet", "rank": 1
        },
      ]
    }
  }
}
```


▲ An example of the data structure after multiple searches and platform switching

2.7.2. Search Records

However, we do believe it would be convenient to keep the search records so that users can search them again after a while. Thus, these data should be stored permanently, until deleted by users.

A database is necessary for storing these data. Due to privacy reasons, we choose to store the data locally instead of running a server that stores the data of all users. This also helps with the app's performance and cost-saving. Traditionally, there are SQL and NoSQL databases available, whereas NoSQL databases are generally faster [9]. A Flutter database, Isar, is being used in this application. It is a NoSQL database that is easy to use and flexible, the data stored in it persist even after the application is closed. It also comes with a handy in-browser inspector tool that makes development easier.

Currently, the application stores only the users' search records. Each record includes an automatically generated ID, search count, search text, and last search time.

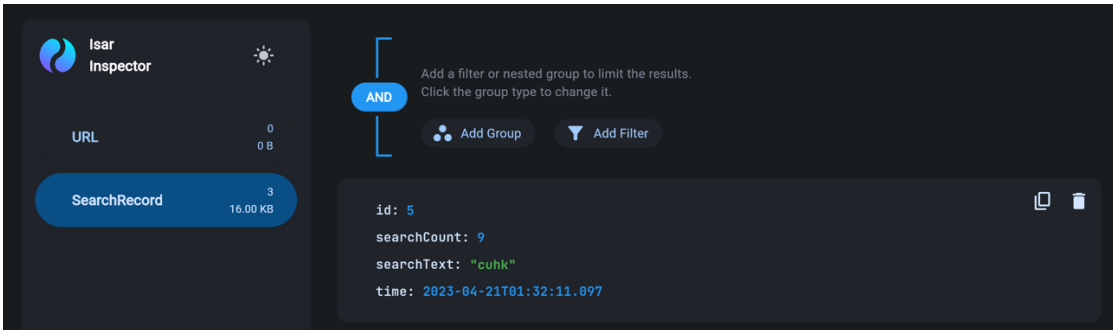


Figure 2.13. In-browser Inspector provided by Isar

2.7.3. Search Histories in Current Search Session

As mentioned previously, users are able to go back to a particular search and drill-down searches should be indicated. Difference between search histories and search records is that search histories refer to searches in the current search session, i.e., from opening the app to closing it; while search records refer to all search users have performed, without time limitation.

It is possible to add a field to the temporarily stored fetched results mentioned above, but it would be difficult fully extract the drill-down layers, i.e., the sequences and parent-child relationship of the search queries.

Thus, a simple data structure is created for storing these data.

```
"SEARCH HISTORIES": {
  "search query 1": false,
  "drill search query 1": "initial search query",
  "drill search query 2": "drill search query 1",
  "search query 2": false,
}
```

▲ An example of the data structure of search histories after multiple searches

3. Results

We are not able to evaluate whether the application provides users with a better searching experience entirely based on users' feedback as it requires an enormous amount of time, manpower, and effort. Alternatively, we measured the effectiveness of our advanced features, including results merging and re-ranking.

Below illustrates the current capabilities of the mobile application and some testing on the effectiveness of our advanced features.

3.1. Different Search Types

The mobile application is now able to perform different types of searches, namely General, Video, and SNS. General search performs searches on Google, Bing, and DuckDuckGo. Video search performs searches on YouTube, Bing, and Vimeo. SNS search performs searches on Facebook, Instagram, Twitter, and LinkedIn. We will mainly focus on General search in the following sections.

Search results from different platforms are merged and re-ranked, then loaded directly as a webpage as expected. Users can browse through different results by swiping the joystick-like controller horizontally (green arrow).

To switch platforms, users simply swipe up the joystick-like controller. To prevent accidental switching, users can cancel the action by dropping the controller on the top of the platform selection menu.

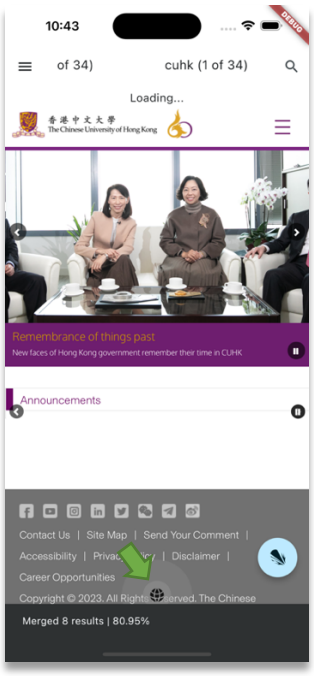


Figure 3.1.
Initial search result.

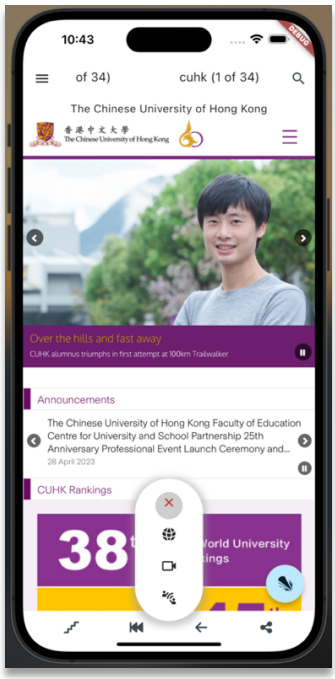


Figure 3.2. Platform switching action can be cancelled.

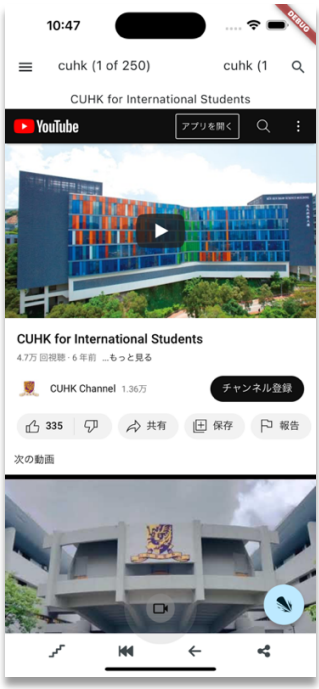


Figure 3.3. New search is performed and new results are loaded immediately.

3.2. Filtering Unique Results

When conducting General search, users can filter out duplicated results.

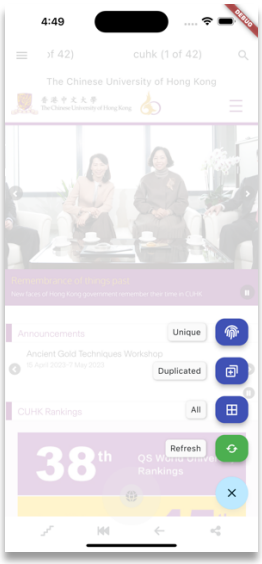


Figure 3.4.
Users can select

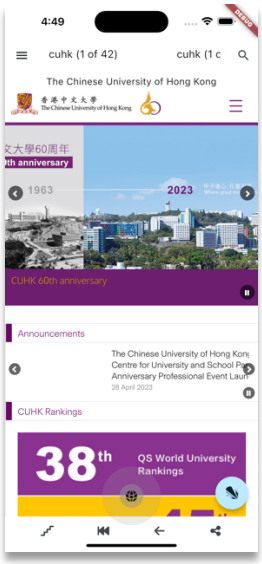


Figure 3.5.
Original results.

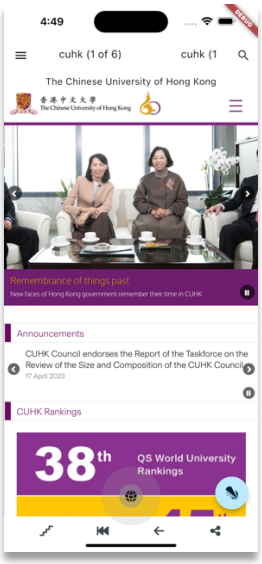


Figure 3.6.
Duplicated results only.

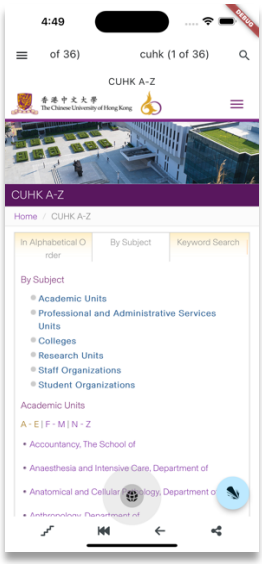


Figure 3.7.
Unique results only.

3.3. Drill-down Searches

Users can trigger a drill-down search either by manually selecting some text within the webpage or drag-and-dropping the FAB. The colour of the top bar and FAB changes accordingly to provide a visual indicator of searching progress.

The first result from the drill down is immediately shown on the screen.

Below is a demo of a drill-down search with hovered content mode selected in action.

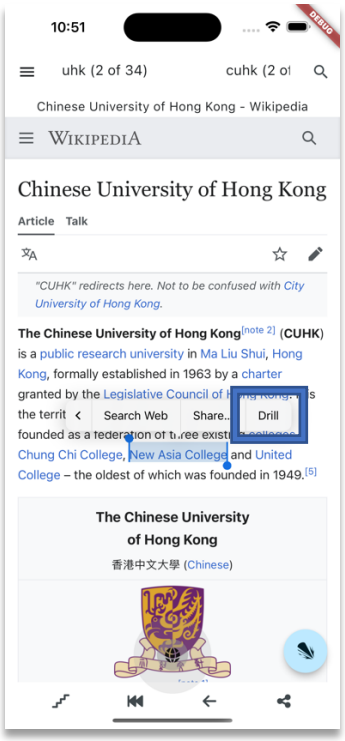


Figure 3.8.
Users select text to drill.

Figure 3.9. Users drag the FAB to trigger a drill-down (hovered content mode used).

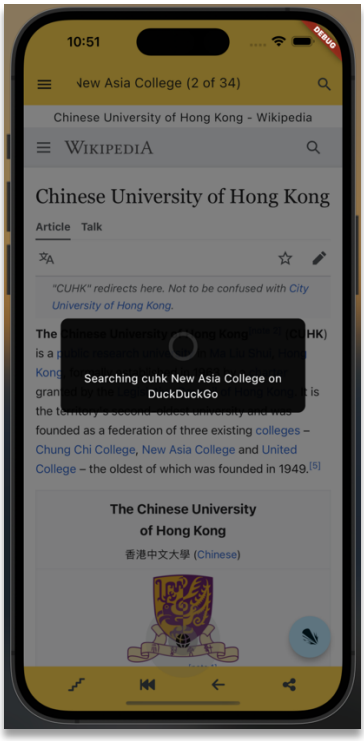


Figure 3.10. Colour the app changes to indicate the drill-down is in progress.

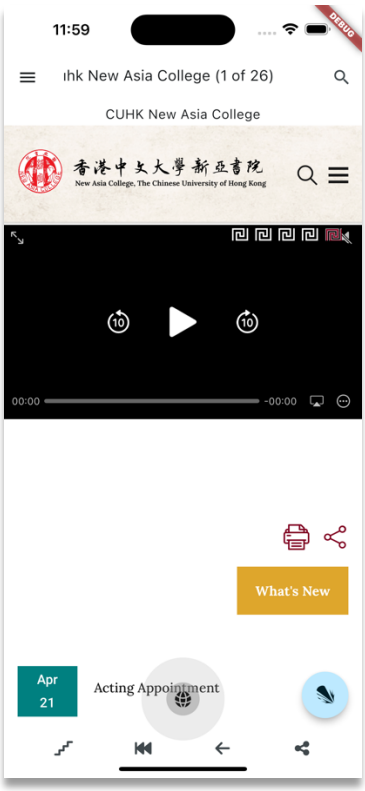


Figure 3.11. The first result from the drill-down is displayed immediately.

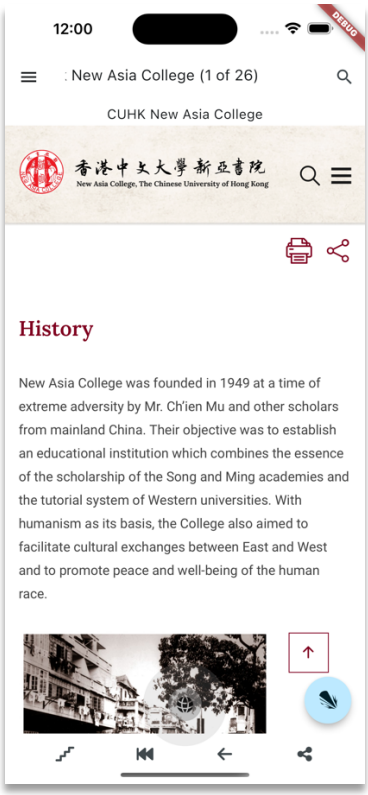


Figure 3.12. Go to another link within the current webpage.

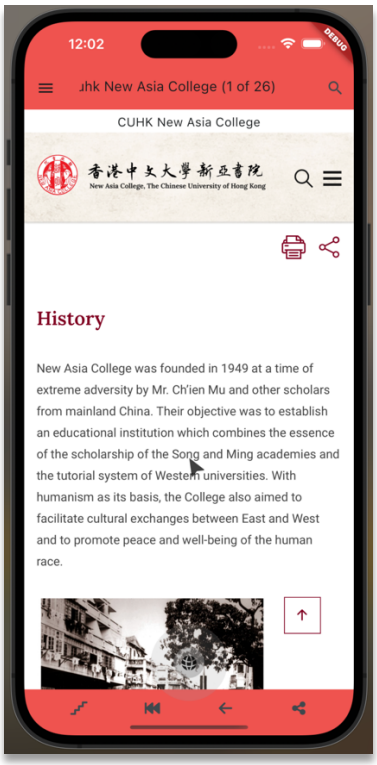


Figure 3.13.
Try to drill on a paragraph.

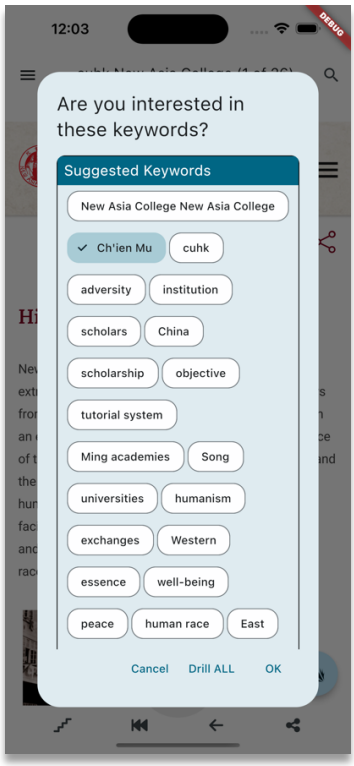


Figure 3.14.
Keywords extraction is performed, a list of suggested keywords appeared.

Figure 3.15. The first result from the drill-down is displayed immediately.

Figure 3.16. Go to another result and try to drill on an image.

Figure 3.17.
Text recognition and keywords extraction are performed, a list of suggested keywords appeared.

Figure 3.18.
Related result appeared.

3.4. Going Back to Previous Search

The last viewed platform and last viewed result of each platform are stored in the app. These data are updated immediately after user goes to another result or switches to another platform.

Below is a demo of a series of searches that include going back to previous search in action.

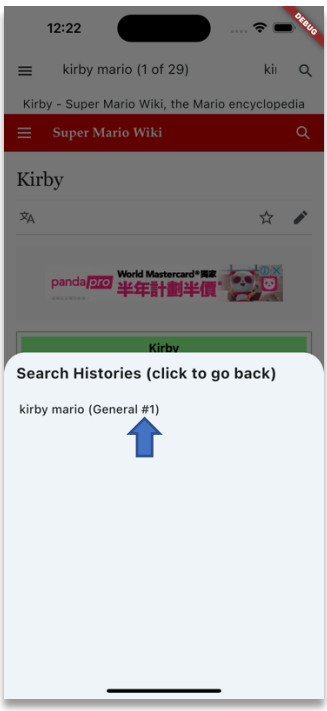


Figure 3.19. The drill histories after the initial search.

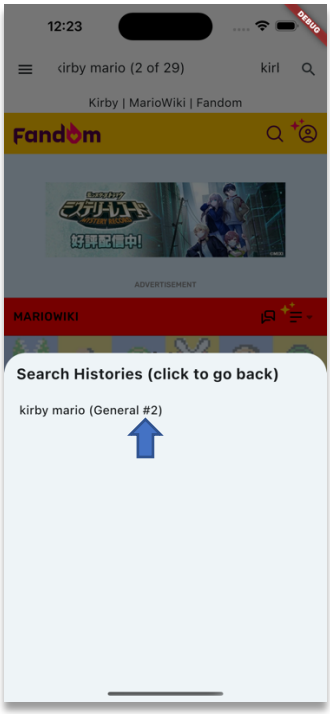


Figure 3.20. The last viewed result is updated immediately upon going to another results.

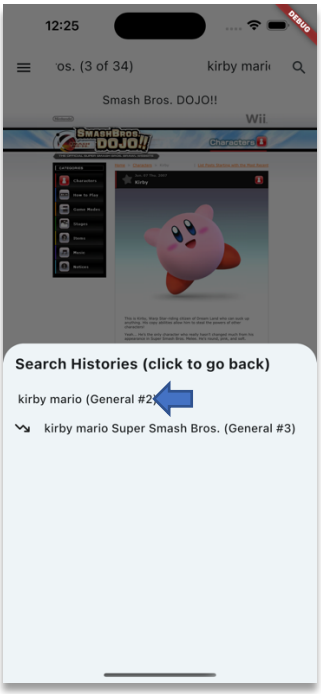


Figure 3.21. After drilling, information of the previous search persists.

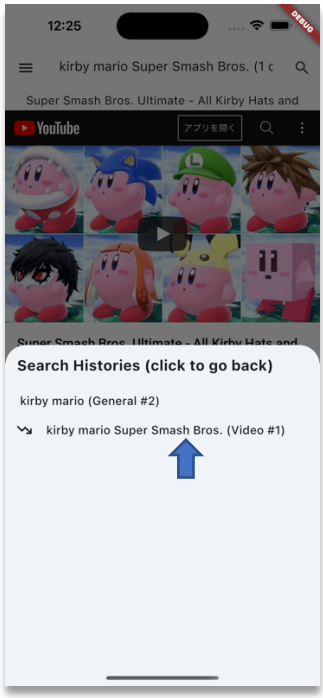


Figure 3.22. The last viewed platform is updated immediately upon switching.

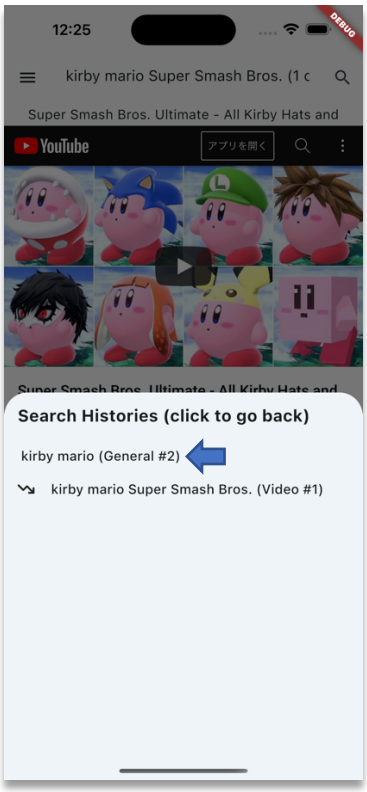


Figure 3.23.
User clicks on the previous search.

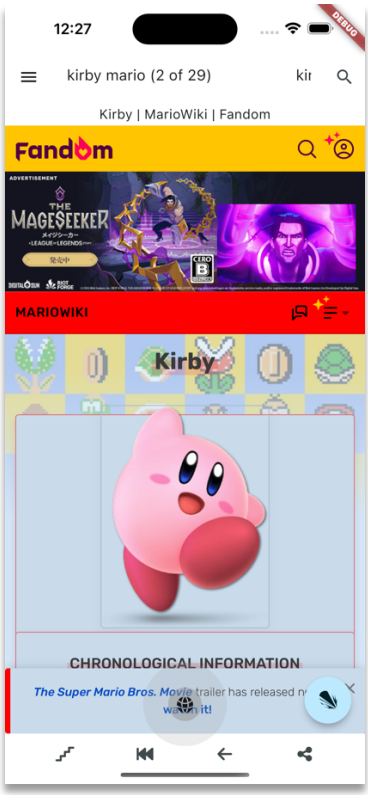


Figure 3.24.
The exact result that the user left at is loaded.

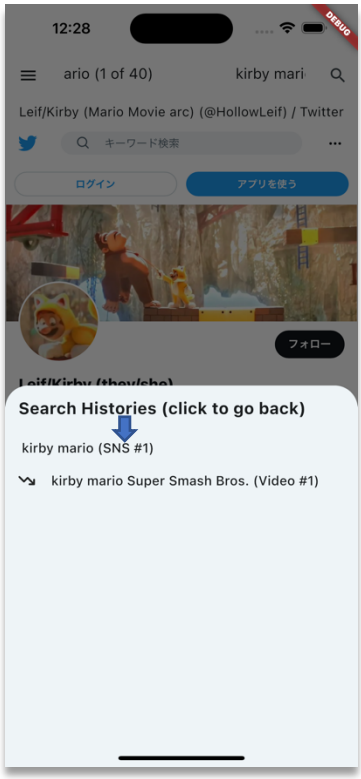


Figure 3.25. The last viewed platform can still be updated.

3.5. Search by Image

The app also allows searching by images. Users can either take a picture or choose an existing image from device's library. Upon submission of the image, the image will be sent to Bing Image API for searching. During the process, visual indicator is provided to notify users of the search process. After the search is completed, users will be prompted to select a related image with label. The selected label will then be used as the search query to perform the search.

Below is a demo of searching by image in action.

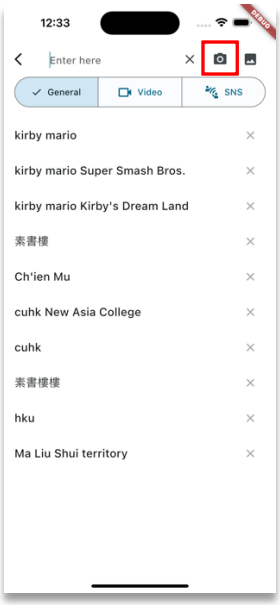


Figure 3.26. Users choose to select image from library.

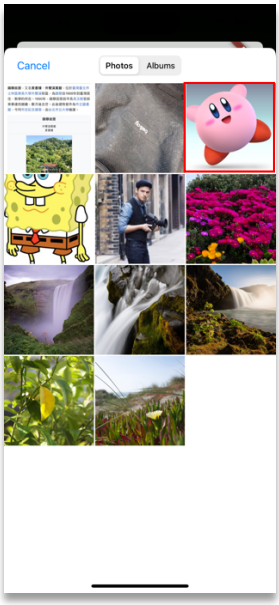


Figure 3.27. Users choose a photo (red box) (or take a picture if they choose to).

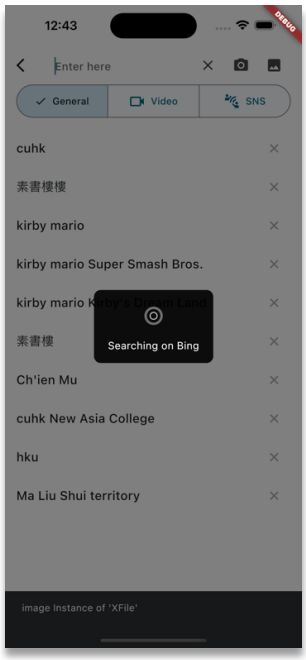


Figure 3.28. Visual indicator of the search progress.

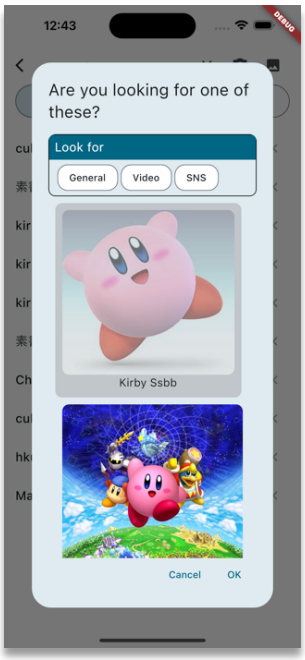


Figure 3.29. A list of related images with label appeared.

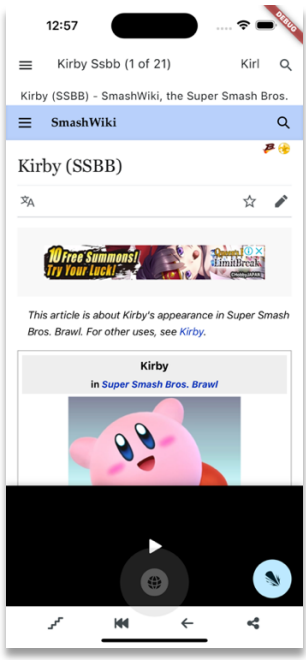


Figure 3.30. First result based on user's selected keyword.

3.6. Results Post-processing

Merging and re-ranking results from different platforms is one of the most important features of this mobile application. Different algorithms have their own merits, we will try to compare them below.

3.6.1.Effectiveness Comparison

To evaluate the performance and effectiveness of the merging algorithms, we have searched the same set of search queries for each algorithm except ABAB. The results are as follows.

Frequency

Search Query	**Total # of Results (Before Merge)**	**Total # of Results (After Merge)**	**Merge Ratio**
cuhk	51	45	88.24%
hku	38	33	86.84%
hkust	43	35	81.40%
iphone 14 pro	46	41	89.13%
flutter	44	36	81.82%
		Overall	*85.49%*

Original Rank

Search Query	**Total # of Results (Before Merge)**	**Total # of Results (After Merge)**	**Merge Ratio**
cuhk	51	45	88.24%
hku	38	33	86.84%
hkust	43	35	81.40%
iphone 14 pro	46	41	89.13%
flutter	44	35	**79.55%**
		Overall	*85.04%*

Further Merge

Search Query	**Total # of Results (Before Merge)**	**Total # of Results (After Merge)**	**Merge Ratio**
cuhk	51	42	**82.35%**
hku	37	32	86.49%
hkust	43	35	81.40%
iphone 14 pro	46	41	89.13%
flutter	44	35	**79.55%**
		Overall	*83.78%*

In terms of effectiveness, these three algorithms seem to be performing similarly, with Further Merge leading with a slight edge. However, looking at the numbers is not enough. In reality, these little differences affect the user experience a lot, which will be discussed in the following sections.

3.6.2.ABAB

As mentioned before, this algorithm only merge results one-by-one without any further processing. Based on multiple tests with different search queries, there is a high chance that the first few results are the same as different search platforms tend to have the same webpage ranked on top. It is also common that a result exists in multiple platforms but ranked differently. In this case, the result will still repeat but less noticeable unless users browse through many results.

Below is one of the test cases that demonstrates the issue.

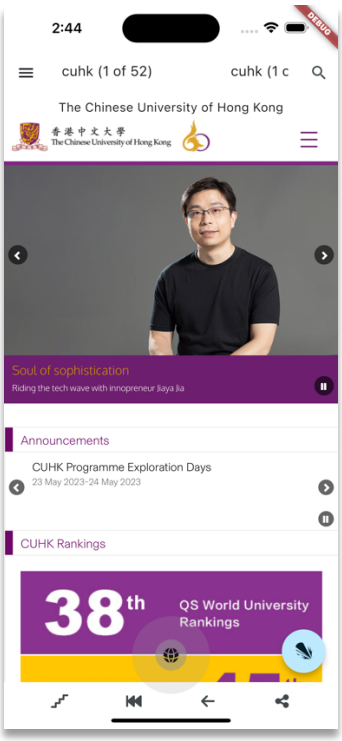


Figure 3.31. First result.

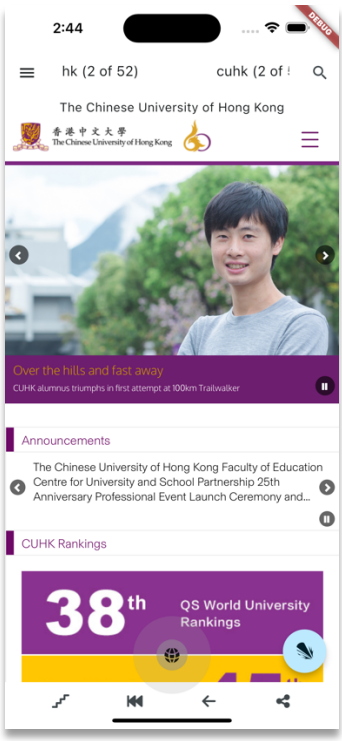


Figure 3.31. Second result.

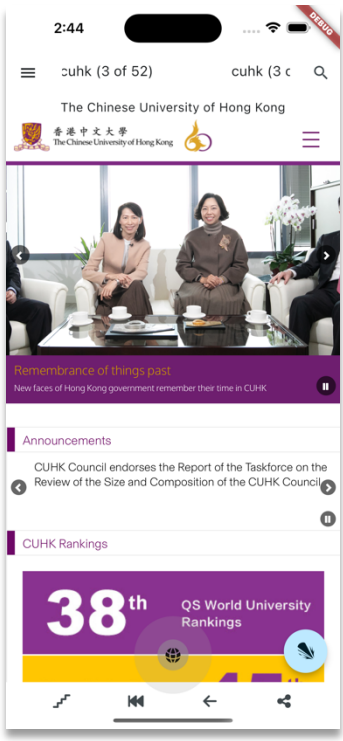


Figure 3.33. Third result.

3.6.3.Frequency

This algorithm takes webpage's frequency into account as well, which does solve the similar links issues to some extent. The chance of seeing the same result significantly decreased.

However, the sequence of the results did not quite match up with the search platforms. Specifically, when searching for "cuhk", the official website of CUHK is ranked first at all three platforms we used. It is safe to assume that it should be the first result to be displayed in our mobile application as well, but obviously this is not the case here.

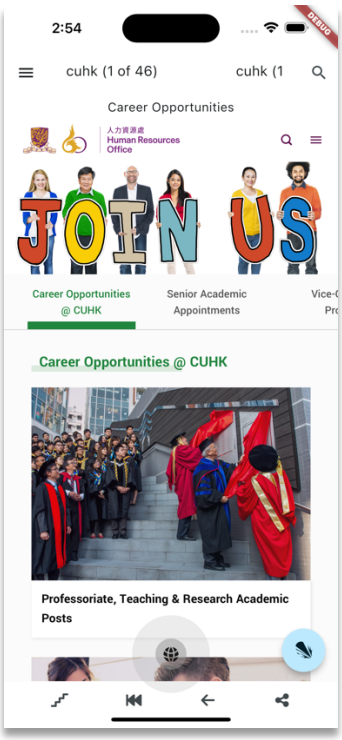


Figure 3.34. First result.

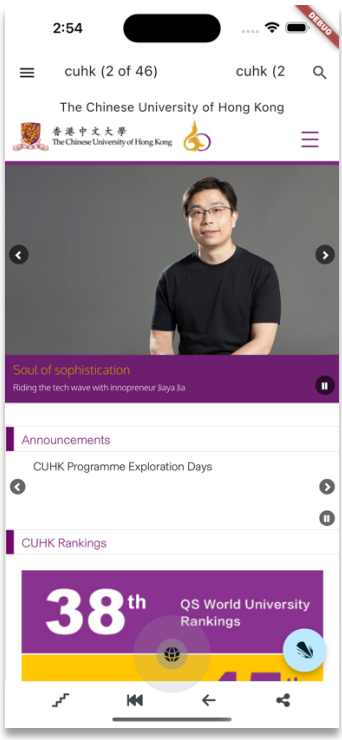


Figure 3.35. Second result.

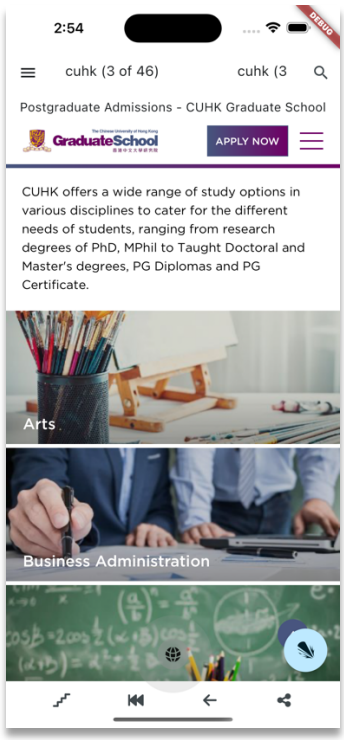


Figure 3.36. Third result.

3.6.4. Original Rank

The reason why the above situation exists is that the rank is not considered during the merging process. Thus, we came up with a scoring system that score every result based on their number of appearance and rank in their original platform, which has been thoroughly discussed above.

The results seem better than using frequency only, as the official website of CUHK is displayed first. However, another problem aroused as the CUHK official website appears again in later result.

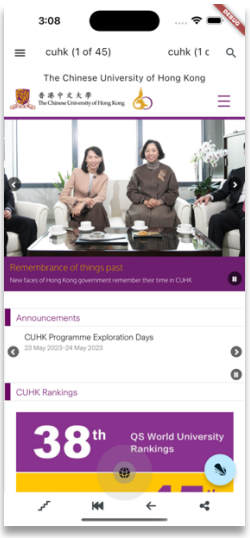


Figure 3.37.
First result.

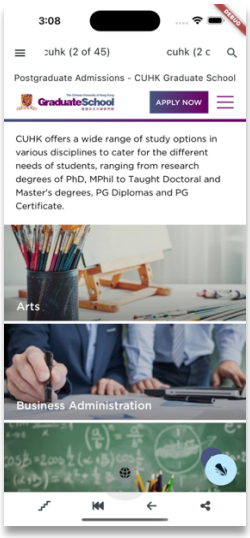


Figure 3.38.
Second result.

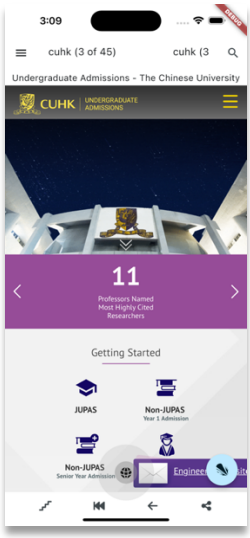


Figure 3.39.
Third result.

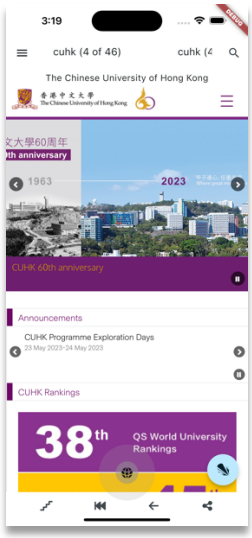


Figure 3.40.
Fourth result (repeated).

3.6.5.Further Merge

The abovementioned problem exists because different platforms do not necessarily return the same URL for the same result. For instance, when searching for “cuhk”, Google returns https://www.cuhk.edu.hk/, while Bing and DuckDuckGo both returns https://www.cuhk.edu.hk/english/index.html. Similar situation includes http://admission.cuhk.edu.hk vs https://admission.cuhk.edu.hk.

This algorithm aimed at further merging similar results. Based on the different test cases, the results are relatively satisfying. Chances of repeating results are reduced even more. The re-ranked results also seem reasonable compared to searching on those platforms directly.

Although there still exist search results that cannot be merged, these kinds of results differ in almost every aspect, such as URL, title, and snippet. There is no easy way to merge them unless having access to their actual content. A possible way to do implement that is to fetch the webpage content, but consider the number of results, it could greatly increase the processing time.

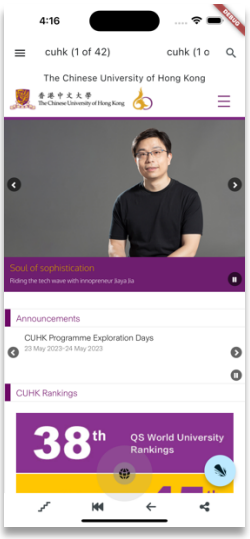


Figure 3.41.
First result.

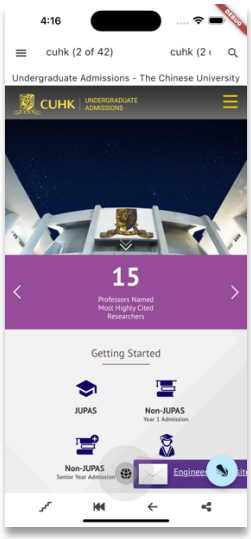


Figure 3.42.
Second result.

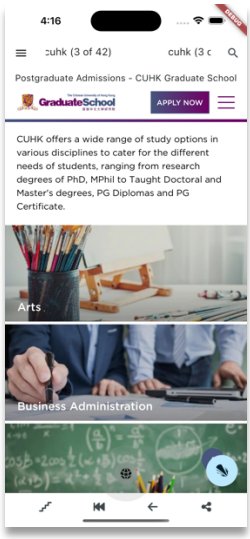
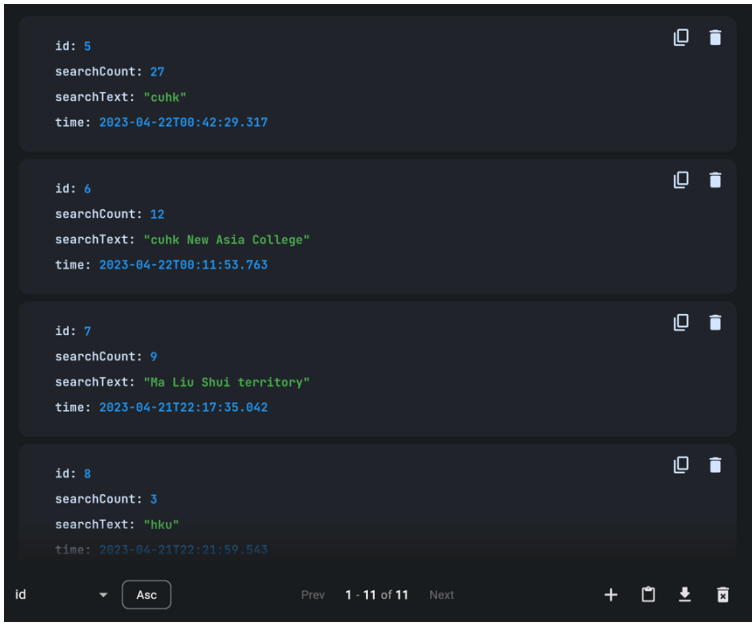


Figure 3.43.
Third result.

Figure 3.44.
Fourth result.

3.7. Database

The database currently stores users search records. Each record consists of a unique ID generated by Isar automatically, search count, search text, and last search time.



▲ **Figure 3.45.** Inspector showing the data being stored.

These data are used to generate the search records in the search page for users to view and quickly search again.

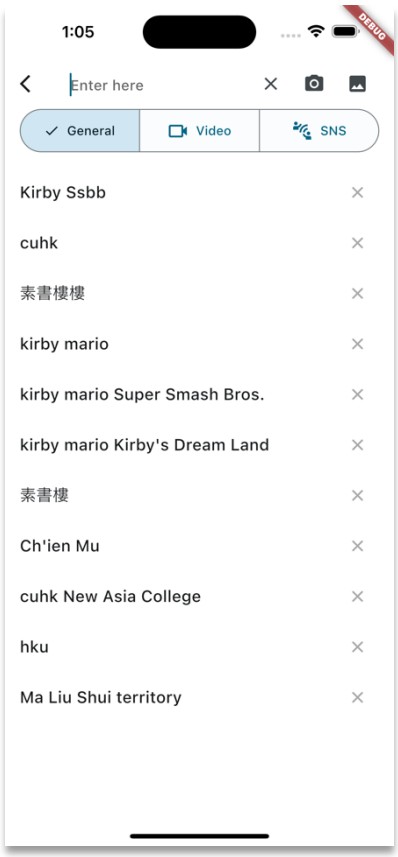


Figure 3.46. Search page showing all the search records. ▲

4. Conclusion

To conclude, we have successfully built a functional cross-platform mobile application that works as a proof of concept. Concepts involved in the project are indeed feasible and are possible to be implemented, such as the multiple platforms integration, automatic keyword refinements, search results merging and re-ranking, and searching with images. With these features integrated into a single app, users can search on their mobile devices more efficiently and conveniently.

We used Flutter to build our mobile application, with flutter_inappwebview and preload_page_view libraries being the core components. Throughout the development process, we have explored different approaches to building a cross-platform mobile application, things to consider when building a mobile application, as well as how to enhance the user experience by implementing intuitive UI designs. We believe we have successfully built a mobile application that is intuitive to use and requires a low learning curve.

The mobile application is certainly a big part of this project, but the (many different) APIs involved are no less important. These APIs are useful and powerful to be used on their own, especially Google Vision API and Google Language API. It is xxx that they can be used together to create a more useful and powerful solution to the existing problems, such as extracting keywords from an image.

This project also shows that search results from different search platforms could vary a lot. In other words, we might be missing out on potentially related information if we stick to a single search platform. Thus, there are indeed benefits and needs to gather results from multiple platforms.

Overall, this mobile application successfully provides a more convenient, intuitive, and efficient searching experience that returns optimal results for everyone that searches on the Internet.

5. Future Directions

Even though the development of the mobile application has come to an end, there are still improvements that can be made to further polish the user experience and effectiveness.

5.1. UI Revision

The current UI of the app works fine, but it could be improved. For instance, the searching option can be displayed at the welcome screen, allowing quicker access to searching functionalities.

5.2. Enhanced Results Post-processing

As mentioned above, there are limitations to current approach of merging results. Different results might be linking to the same webpage while their URL, title, and snippet are all different, making them impossible to be compared.

A possible way to overcome this problem is to fetch the actual webpage content, similar to web scraping results from DuckDuckGo website. To minimize the performance loss, this could be done in the background while showing some results to users first.

5.3. Search Images / Videos

Instead of webpages, images or videos are directly shown/played. To do so, we need to look for a way to display images or play videos within the app as well as to make sure only URLs of images and videos are being returned from the APIs.

6. References

[1] J. Clement, "Most popular websites worldwide from 1993 to 2020, by highest number of monthly visits," August 2020. [Online]. Available: https://www.statista.com/statistics/1175389/most-popular-websites-monthly-visits/. [Accessed 22 October 2022].

[2] J. Clement, "Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 2nd quarter 2022," 20 July 2022. [Online]. Available: https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/. [Accessed 25 October 2022].

[3] A. Sharma, N. Aggarwal, N. Duhan and R. Gupta, "Web search result optimization by mining the search engine query logs," 10 February 2011. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5706716/authors#authors. [Accessed 28 November 2022].

[4] S. Borick, "Comparing the Usage of React Native and Ionic," 2018. [Online]. Available: https://ideaexchange.uakron.edu/honors_research_projects/620/. [Accessed 24 October 2022].

[5] A. E. Fentaw, "Cross platform mobile application development: a comparison study of React Native Vs Flutter," 2020. [Online]. Available: https://jyx.jyu.fi/handle/123456789/70969#. [Accessed 24 October 2022].

[6] W. Wu, "React Native vs Flutter, cross-platform mobile application frameworks," 1 March 2018. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf?sequence=1. [Accessed 24 October 2022].

[7] D. Ingale and M. M. Kshirsagar, "A review on an approach to infer user search goals for optimize result," 6 October 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7583941. [Accessed 24 October 2022].

[8] A. Turpin, Y. Tsegay, D. Hawking and H. E. Williams, "Fast generation of result snippets in web search," in *The 30th Annual International SIGIR Conference*, New York, 2007.

[9] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6625441. [Accessed 25 October 2022].

[10] V. P. Abbott, "Web page quality: can we measure it and what do we find? A report of exploratory findings," *Journal of Public Health Medicine,* vol. 22, no. 2, pp. 191-197, 2000.

7. Appendix

7.1. Consolidated Weekly Logbook

Date	Work done in the past week	Challenges / problems faced	Tasks planned to be done in the coming week
30-01-2023	The results can now be preloaded before user scroll to it. For instance, when user is looking at the first webpage, the second webpage has already been loaded. Also, the search results were originally limited to 10 only. The app can now extend it automatically upon reaching the end of result.	The drag-and-drop feature did not work after adding the preload feature, but by changing the target widget got capturing the location, I have successfully solved the problem.	The bottom bar is now buggy due to the above changes, I planned to fix it in the coming week.
06-02-2023	I have fixed the bottom bar and updated the UI a bit. Previously, some basic functionalities such as drag-and-dropping wasn't working properly because of a new package for preloading the result. It is also fixed now.	It was tricky to keep track of the current webview controller, which is crucial for the drag-and-drop (position). As each result (webpage) has their own controller, and the controller doesn't seem to have anything related to the result that is accurately identifiable. Thus, I am using the position of the result to keep track of the current controller.	I planned to implement a new way of switching result and platform, preferably by swiping.
13-02-2023	I have implemented a new way of switching platform. Instead of selecting from a dropdown menu, users now tap to loop through different platforms, and long-press to search on that platform. I have also investigated possible new ways for drilling, a possible way is to add a menu item in existing text selection menu.	It is extremely complicated and might not be feasible to be done in Flutter, some packages are available, but they are for Android only, but the app should work on iOS too.	I planned to switch to new package, Flutter InAppWebView, it is similar to the official WebView, but it seems to offer easy text selection menu modification.

20-02-2023	I have switched to a new package for loading the web content, i.e., InAppWebView. The text selection menu is also modified, users can manually select text and drill on them.	There were some differences between the new and old packages, but I managed to keep the functionalities even using the new package.	I planned to try to provide some visual feedback when selecting content for drilling.
27-02-2023	I have implemented a new way to change platform. The app can now do search automatically on new platform if there is no more change for a certain amount of time. Also, the InAppWebViews are now in 2D structure, which the search results on each platform can be preserved.	There were some difficulties when attempting to do the search automatically after certain amount of time, mainly due to the sequence of the platform being activated.	I planned to do some more UI/UX enhancements.
06-03-2023	I have fixed some bugs related to platform switching. I have also added a page for showing the drill histories, users can also go back to previous search results from there.	To allow users to go back to previous results, the app need to store the last viewed platform and result position, then make use of this data go "move" the webviews. There are still some bugs, sometimes the webviews are still showing the previous results preloaded.	I planned to fix the abovementioned bugs in the coming week.
13-03-2023	I have implemented a new way to navigate through results, which is essentially swiping, and should be more convenient and intuitive than the button approach. I have also fixed some bugs and allow users to select platform before their initial search. They can now also go back to a particular search.	Going back to a particular search is way trickier than I have thought, some necessary functions provided by the InAppWebView package doesn't work as expected, and I need to make a workaround so as to realise this functionality.	I planned to implement a new way for switching platform that is swipe-based, which should be more convenient and intuitive.

20-03-2023	I have integrated the language API into the app for extracting keywords from long paragraphs. I have also integrated the YouTube API into the app, it now returns video ID, which can then be converted to URL of the video.	It is difficult to come up with a UI that is efficient and quick for users to select keywords they want.	Since I mainly implemented new features this week, I planned to continue the plan from last week. That is to implement a new way for switching platform that is swipe-based, which should be more convenient and intuitive.
27-03-2023	I have implemented a new way for switching platform, users can drag-up the joystick controller and drop at whichever platform they want. The app will then switch to that platform, the whole action can be done within a swipe. I have also added a feature so that the search records can be stored permanently and search again more conveniently. I have also revised the image search functionalities. Users can now choose between Google or Bing for image search. I have also added some animations to indicate the search process.	It is difficult to implement the swipe-to-switch-platform feature because there is no existing package that is suitable. Thus, I modified the joystick controller to achieve the goal.	I planned to further enhanced the image search feature in the coming week.
03-04-2023	I have merged search results from multiple platforms (Google and Bing for now) and display them as if they are all from a single platform.	It is difficult to incorporate this feature into the app, mainly because of the UI.	I planned to do some results post-processing so as the better merge the results, as well as expanding this idea to image search.

10-04-2023	I have simplified the UI, users now only need to choose their intended type of results rather than platform. I have also done some results post-processing to the merged results, currently by the frequency (i.e., appears in both platform = higher ranking = display first). I have also merged search results from multiple platforms (Google and Bing for now) and display them as if they are all from a single platform for image search.	It is difficult to identify if the results are the same or not. For example, www.cuhk.edu.hk and www.cuhk.edu.hk/english/index.html . They are returned by Google and Bing respectively, with the former one eventually redirected to the latter one. It is not possible to determine just by the URL.	I planned to enhance the results post-processing, such as adding more factor for the ranking process.