# DS Final Project Proposal

Group 5

1. Introduction - Motivation:
    a. Topic: Movie & TV series
    b. Motivation:
    When we google the movie or TV series we want to see, we often don't get the information we want. While searching for movies or series that are set in real events or history, you can probably only find the event itself and not the movie we want to see. Therefore, we want to develop a search engine that can search the contents related to movies, series and movie reviews more accurately.
    c. Example
        i. A movie may have many names after different translations, so we can't get all the information about this movie at once.

        ii. We want to query history-related movies, such as World War II. It is common to get the result of historical events, rather than movies with related themes.

        iii. The series I want to watch was removed by Netflix, and when I searched, I only found movie clips.

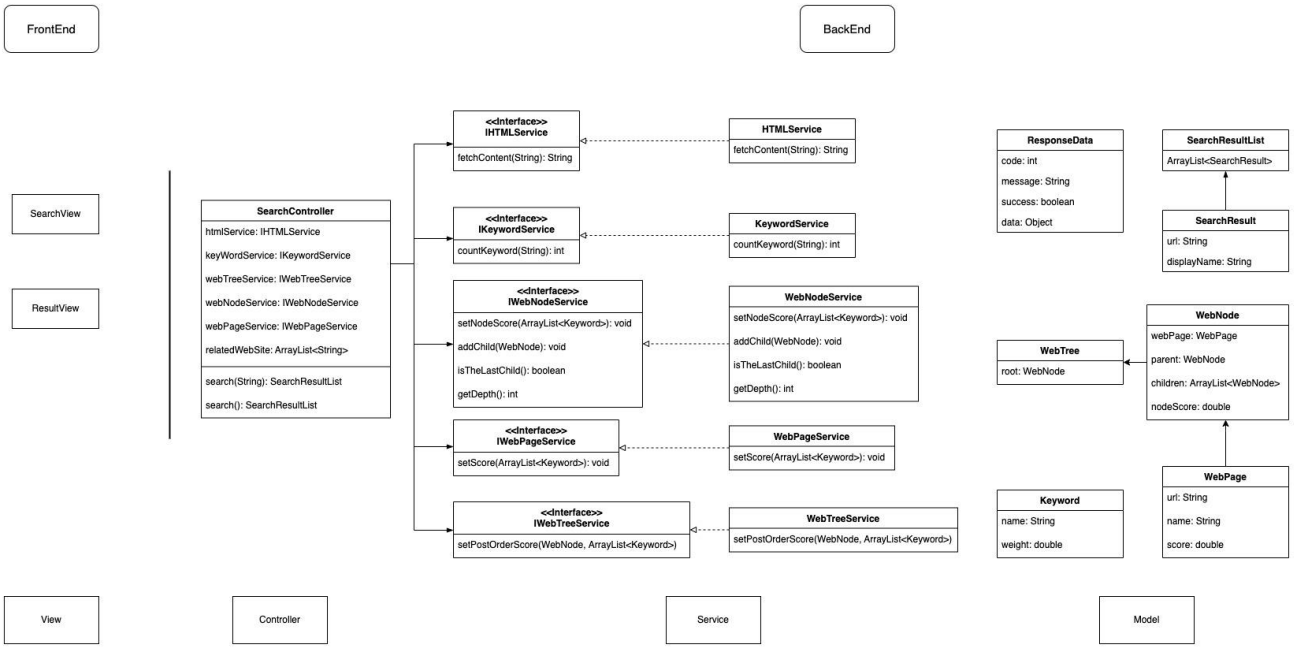
2. Search tricks - Score formulation:
    a. Keyword:
        i. 電影 (weight: 10)
        ii. 影集 (weight: 10)
        iii. 影評 (weight: 8)
        iv. Netflix (weight: 7)
        v. Disney+ / Disney Plus (weight: 7)
        vi. 電影院 (weight: 5.5)
        vii. 場次 (weight: 5.5)
        viii. 票價 (weight: 5)
        ix. 电影 (weight: 10)
        x. movie (weight:10)
        xi. 改編 (weight: 4.5)
        xii. 觀賞 (weight: 4.5)
        xiii. 评分 (weight: 4)
        xiv. 評分 (weight: 4)
        xv. 小說 (weight: 4)

    b. Formulation:
    $$Score = keywordCount * weight$$

c. The website can be ranked:
   i. 【影評人監修】2022最新十大人氣漫威電影推薦
   ii. 2022年【Netflix最高分20部】迷你影集推薦
   iii. 環球影城 - 熱售電影
   iv. 8 月不能錯過的 10 部 Disney+ 影集與電影！

3. System Design - Class diagram:

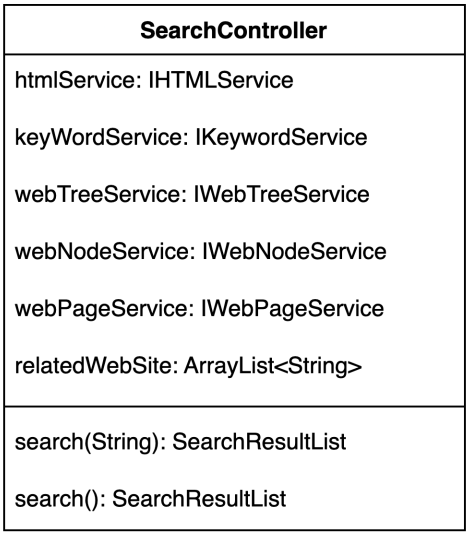
   a. Class Diagram




   b. Skill Detail

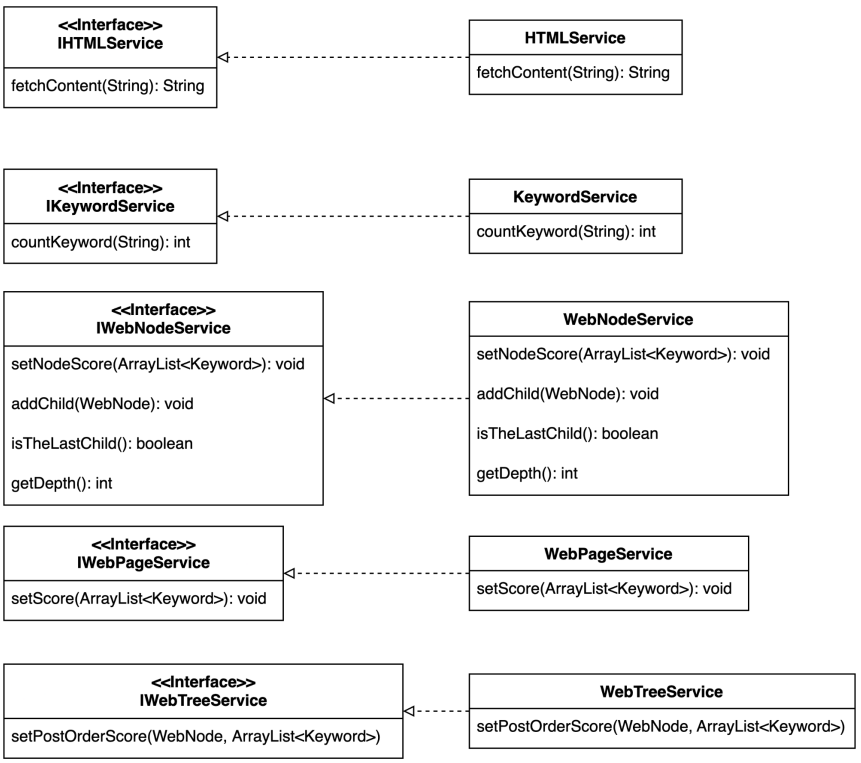
      i. View:
      The view of the APP. There are two pages in our application. The one is SearchView which is the page that gets the user's input. The second is ResultView. Like Google search results, we will provide a page for the result of our search engine after searching.

      ii. Controller:




Our SearchController uses multiple Services to do business logic. The entry point of our search system is the search route API. It gets the user input as a keyword and searches Google and then rerank by our search engine. The rerank mechanism includes all stages required in the final project.

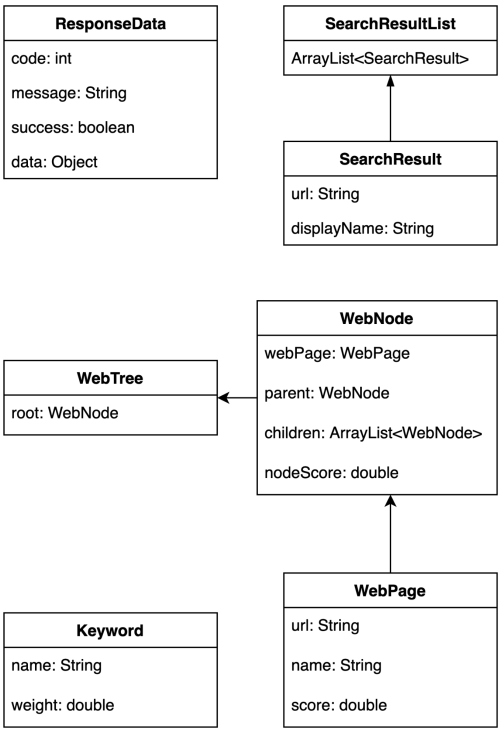
iii. Service (Business Logic)




We encapsulate our business with Service and in other modules, they can use this Services to achieve business

purposes. Note that we use Interface as the dependency of other modules with our Services. By doing so, we can implement Liskov's substitution principle. Furthermore, we can use dependency injection to decoupling the modules and our Services.

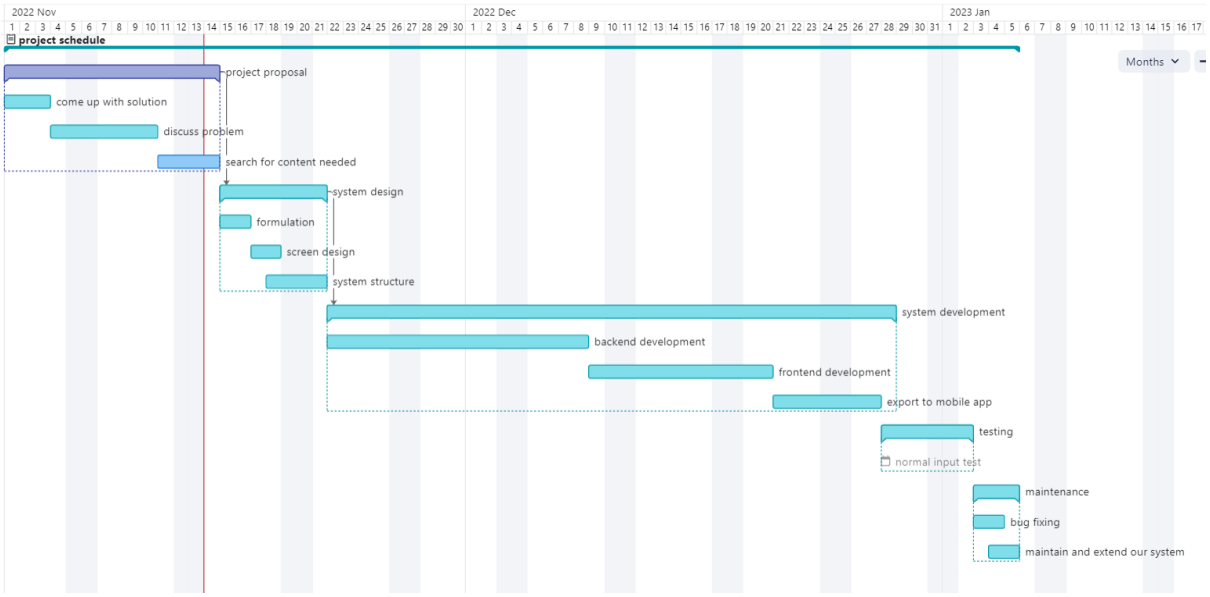
iv. Model




This layer is nothing but a data model. Note that we use ResponseData model to unify the response of our Controller.

4. Schedule:

| | | |
|---|---|---|
| project schedule | 2022/11/01 | 2023/01/05 |
| project proposal | 2022/11/01 | 2022/11/14 |
| come up with sol... | 2022/11/01 | 2022/11/03 |
| discuss problem | 2022/11/04 | 2022/11/10 |
| search for conten... | 2022/11/11 | 2022/11/14 |
| system design | 2022/11/15 | 2022/11/21 |
| formulation | 2022/11/15 | 2022/11/16 |
| screen design | 2022/11/17 | 2022/11/18 |
| system structure | 2022/11/18 | 2022/11/21 |
| system development | 2022/11/22 | 2022/12/28 |
| backend develop... | 2022/11/22 | 2022/12/08 |
| frontend develop... | 2022/12/09 | 2022/12/20 |
| export to mobile ... | 2022/12/21 | 2022/12/27 |
| testing | 2022/12/28 | 2023/01/02 |
| normal input test | | |
| maintenance | 2023/01/03 | 2023/01/05 |
| bug fixing | 2023/01/03 | 2023/01/04 |
| maintain and ext... | 2023/01/04 | 2023/01/05 |




5. Challenges: (All solved)
    a. How to use a backend framework like spring boot ?
    b. How to communicate between frontend and backend ?

6. Implementation
    a. Tool & Why use it:
        i. Frontend: React Native
            1. Cross-platform. That makes it easy to build a web and mobile app simultaneously.
            2. Easy to use. RA can use basic javascript and some fundamental frontend skill to implement a frontend APP.

3. Time-saving. Based on the above mentioned , we can achieve our goals(build web and mobile APP together), so that we can save our time.

ii. Backend: Java SpringBoot

1. Course Requirement. This course requires using Java to implement the backend of the project.
2. Well structured. SpringBoot provides a clear project structure and some pre-built useful API to use.
3. Support advanced OOP skills like DI, IoC, MVCS, Design Pattern and so on.

b. Stage Finished & Partial code

i. In our project, we complete all the stages(0~6). Below we will show the crucial code that implemented the stages.

ii. Stage0 - Keyword Counting

We simply use this function to find the keyword index and keep counting the count of the keyword in the content.

```java
@Override
public int countKeywords(String content, String keyword) {
    if (content == null){
        return 0;
    }

    //To do a case-insensitive search, we turn the whole content and keyword into upper-case:
    content = content.toUpperCase();
    keyword = keyword.toUpperCase();

    int res = 0;
    int fromIdx = 0;
    int found = -1;

    while ((found = content.indexOf(keyword, fromIdx)) != -1){
        res++;
        fromIdx = found + keyword.length();
    }

    return res;
}
```

iii. Stage1 - Page Ranking

We simply make the SearchResult model implement the Comparable interface and override the below function, and use sort() to rank the search result based on their score.

```java
@Override
public int compareTo(SearchResult o) {
    if(this.score > o.score) {
        return -1;
    }else if(this.score < o.score) {
        return 1;
    }else {
        return 0;
    }
}
```

```java
public void sort()
{
    Collections.sort(this.results);
}
```

iv. Stage2 - Site Ranking

In this stage, we have two steps. The first is building the WebTree. We capture the three subsites of the websites and build the tree relationship. Secondly, we use post order traversal to score each WebNode and finally the WebTree.

```java
//TODO: only in default search, if based on google search will be too difficult
//TODO: So we just catch some url(3 of them) in "a" tag
@Override
public WebTree buildTree(WebNode startNode) throws IOException {
    // 1. get 3 sub url that contains in "a" tag, and satisfied the http format
    String rootUrl = startNode.getPage().url;
    ArrayList<String> validUrls = new ArrayList<String>();

    Connection conn = Jsoup.connect(rootUrl);
    conn.timeout(10000);
    Document doc = conn.get();
    Elements aTags = doc.select("a");
    for(Element aTag : aTags) {
        String tryUrl = aTag.attr("href");
        if(tryUrl.startsWith("http")) {
            validUrls.add(tryUrl);
        }
        if(validUrls.size()==3) {
            break;
        }
    }

    // 2. add WebNode under the tree
    for(String url : validUrls)
    {
        WebPage page = new WebPage(url);
        WebNode node = new WebNode(page);
        node.setParent(startNode);
        startNode.addChild(node);
    }

    WebTree tree = new WebTree(startNode);

    return tree;
}
```

```java
@Override
public void setPostOrderScore(WebTree tree, ArrayList<Keyword> keywords) throws IOException, URISyntaxException {
    // TODO Auto-generated method stub
    WebNode root = tree.getRoot();
    this.setPostOrderScore(root, keywords);
}

public void setPostOrderScore(WebNode startNode, ArrayList<Keyword> keywords) throws IOException, URISyntaxException
{
    for(WebNode node : startNode.getChildren())
    {
        this.setPostOrderScore(node, keywords);
    }
    this.webNodeService.setNodeScore(startNode, keywords);;
}
```

v. Stage3 - Refine the rank of Google

We use Jsoup to fetch the google result and get the data in the page. In the google result, we preprocess the URL first because we find out that the URL data is dirty with some symbols mixed in the URL. Finally, we keep the process of Stage 0~2 to rank our results.

```java
@Override
public HashMap<String, String> searchGoogle(String keyword, int count, int skip) throws IOException {
    String content = this.fetchGoogleResultContent(keyword, count, skip);

    HashMap<String, String> res = new HashMap<String, String>();

    //using Jsoup analyze html string
    Document doc = Jsoup.parse(content);

    //select particular element(tag) which you want
    Elements lis = doc.select("div");
    lis = lis.select(".kCrYT"); // google 搜尋後結果中的class

    for(Element li : lis)
    {
        try
        {
            String citeUrl = li.select("a").get(0).attr("href");

            //修正奇怪的回傳URL, 因為爬回來的結果不完全和搜尋結果一樣
            // parse guide: https://system.camp/tutorial/searching-google-results-and-parsing-in-java/
            // 把抓回來的content print出來看看 分析
            citeUrl = "http"+citeUrl.split("http")[1];
            citeUrl = citeUrl.split("&sa").length > 0 ? citeUrl.split("&sa")[0] : citeUrl ;
            citeUrl = citeUrl.replace("%25", "%");

            String title = li.select("a").get(0).select(".vvjwJb").text();

            if(title.equals(""))
            {
                continue;
            }

            // description在緊接著下一個的kCrYT
            Element desOuterDiv = li.parent().select(".kCrYT").get(1);
            String description = desOuterDiv.select(".BNeawe").get(1).text();

            //put title and pair into HashMap
            res.put(title, citeUrl+":::"+description);

        } catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    return re
}
```

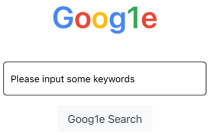
vi. Stage4 - Semantics Analysis

In this stage, we simply use Dynamic Programming(DP) to find the Longest Common Substring(LCS) of the description of the website and our keywords. We also return the relative keywords and the LCS to let users take into consideration.

```java
@Override
public ArrayList<String> deriveRelativeKeywordsFromDescription(String description, ArrayList<Keyword> keywords) {
    // use DP to find LCS of keywords and website's description
    ArrayList<String> res = new ArrayList<String>();

    for(Keyword keyword : keywords)
    {
        String[][] dpTable = new String[description.length()+1][keyword.name.length()+1];
        for(int i=0; i<description.length()+1; i++)
        {
            dpTable[i][0]="";
        }
        for(int i=0; i<keyword.name.length()+1; i++)
        {
            dpTable[0][i]="";
        }

        for(int i=0; i<description.length(); i++)
        {
            for(int j=0; j<keyword.name.length(); j++)
            {
                if(description.charAt(i) == keyword.name.charAt(j)) {
                    dpTable[i+1][j+1] = dpTable[i][j]+keyword.name.charAt(j);
                }else {
                    dpTable[i+1][j+1] = dpTable[i][j+1].length()>dpTable[i+1][j].length() ? dpTable[i][j+1] : dpTable[i+1][j];
                }
            }
        }
        if(
            !dpTable[description.length()][keyword.name.length()].strip().isEmpty()
            && !res.contains(dpTable[description.length()][keyword.name.length()])
        ) {
            res.add(dpTable[description.length()][keyword.name.length()]+"=>"+keyword.name);
        }
    }

    return res;
}
```

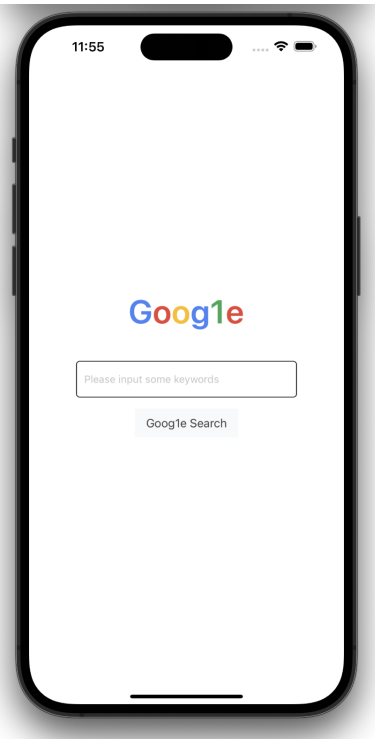
vii. Stage5 - Publish Your Work Online

We use React Native and Expo to make our work online. Below is the web view.

viii. Stage6 - Export Your Work to App

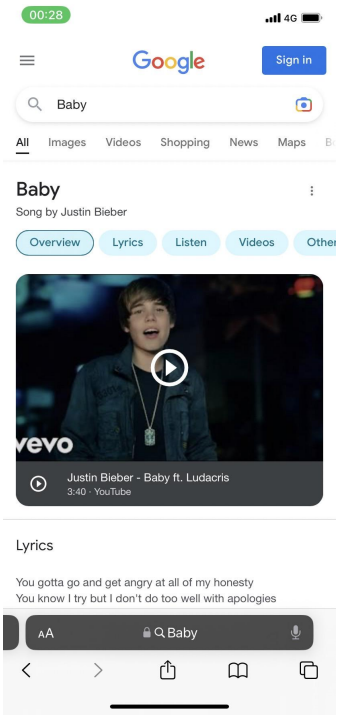
Similarly, we use React Native and Expo to export our work to APP.

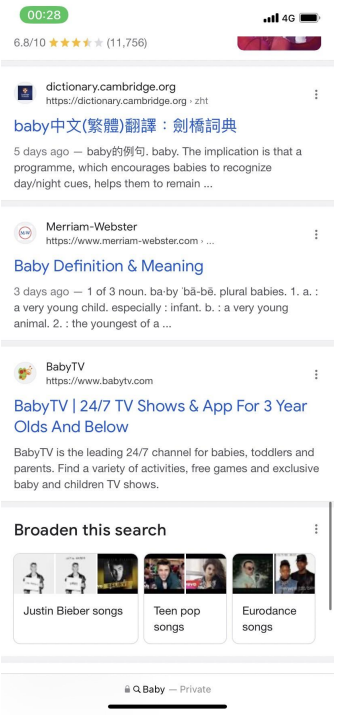



c. Example result:

Below we will show the example result of our APP.

The first is to search for the keyword “Baby”. As you can see, the original google results always show the song or baby related products.

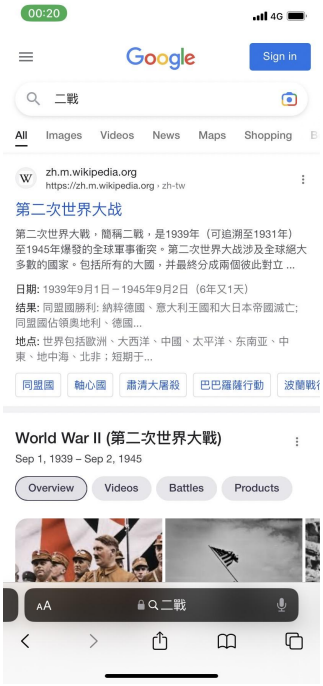


With our search engine, we can find the Netflix series or the movie related results.



The second example is WWII. If we search WWII in google, we will get the history information of WWII. In our search engine, we will get the movie or series about WWII.



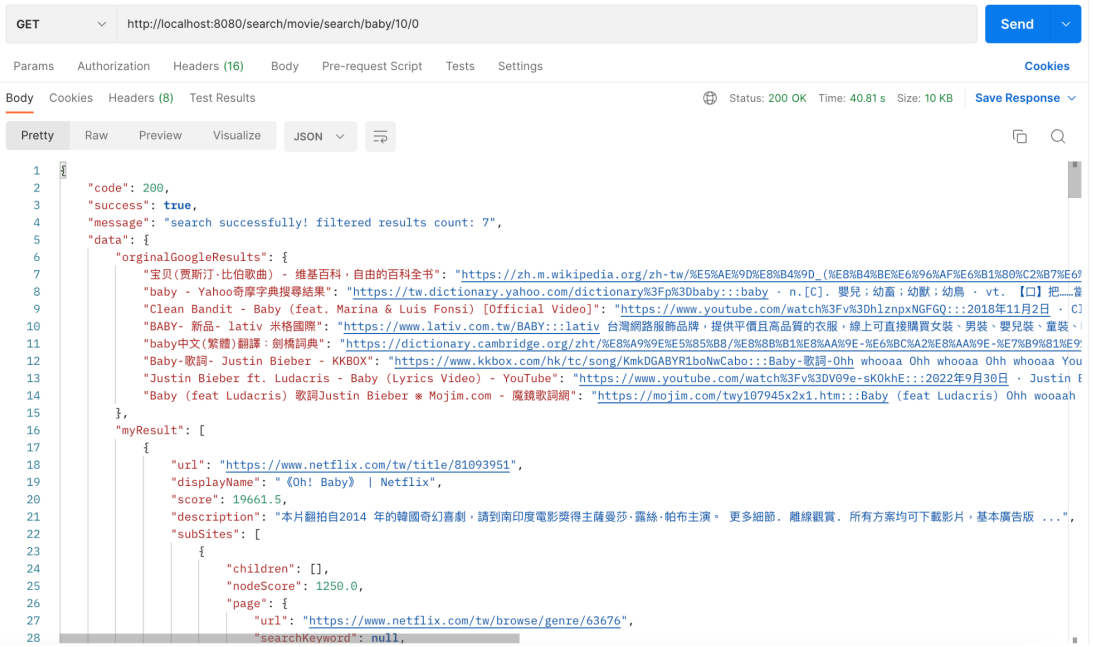

As you can see, you can also span the relative keywords and subsites results.



d. Data result:

Below we will show the backend API to get our search results.

As you can see, we will provide some basic http result data such as code, success, message and data. The main data is in the “data” field. We give original google results and our results to users. In our results, we will provide the URL, name, description, subsites and total score. In the subsites section, we will also give each subsite information such as node score and link etc.




7. GitHub Repository:
   [專案GitHub連結]

8. PPT Link:
   [PPT 連結]

9. Division of Labor:

| | |
|---|---|
| 黃茂勛 | proposal, class-diagram, programming |
| 黃政詠 | proposal, UI design |
| 劉祐辰 | proposal, Schedule |
| 李承峻 | proposal, Schedule |
| 李旻叡 | proposal |