

# Final Project Proposal — Music Festival

## 1. Members :

109306061 呂學柏 & 109306043 黃尹彤 & 109306060 劉家妤

## 2.Introduction:

(1) Our Topic : Domestic Music Festivals

(2) Our User :

For people who love rock bands, metal bands or indie bands and so on, the Google Search Engine may not meet their needs, since it may give results of the music concerts held by Taichung City Mayor Lu Xiu-Yan or other useless information display.

Obviously, it did not meet expectations, so we want to re-weight and do sorting to provide this group of people a better and more realistic result.

### (3) Our Keywords & Weight :

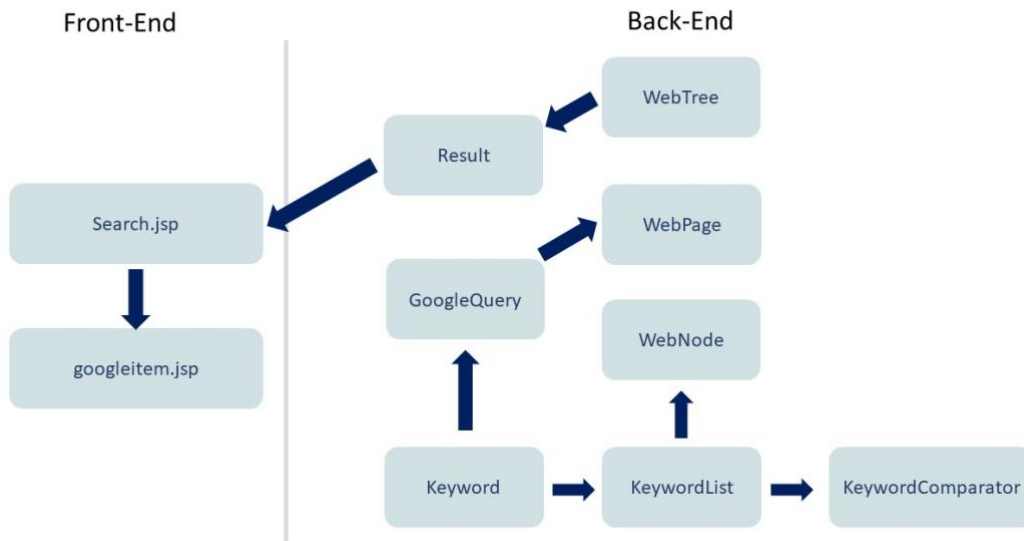
Keywords	Weight
音樂祭	20.0
獨立音樂	10.0
獨立樂團	10.0
樂團	10.0
台灣	4.5
臺灣	4.5
陣容	3.0
票價	3.0
搖滾	3.0
龐克	3.0
金屬	3.0

另類	3.0
滅火器	2.0
怕胖團	2.0
血肉	2.0
閃靈	2.0
美秀	2.0
拍謝少年	2.0
老破麻	2.0
荷爾蒙少年	2.0
海豚刑警	2.0
無妄	2.0
大港	1.5
浪人	1.5

漂遊	1.5
火球	1.5
赤聲	1.5
爛泥	1.5
山海屯	1.5
五月天	-10.0
周興哲	-10.0
盧秀燕	-10.0
流行音樂	-20.0

(4) The Score Counting Formula :  $\text{Keyword.this.name} * \text{Keyword.this.weight}$

### 3. Class Diagram :



### 4. Development Schedule :

Date	Schedule
11/1-11/7	Proposal
11/8-11/15	Midterm Exam Week
11/16-11/23	Decide Keywords, Formula and Weight
11/24-12/1	Back End Development : HTML Matcher & Handler

12/2-12/8	Back End Development :  Keyword Class & Counter
12/9-12/15	Back End Development  Front End Development
12/16-12/22	Front End Development
12/23-12/30	Debug &Testing
12/31-1/6	PPT Design and Demo
1/7-1/14	Upload Project and Codes

## 5. UML Diagram :



Figure 1. The graphical user interface (Search.jsp)

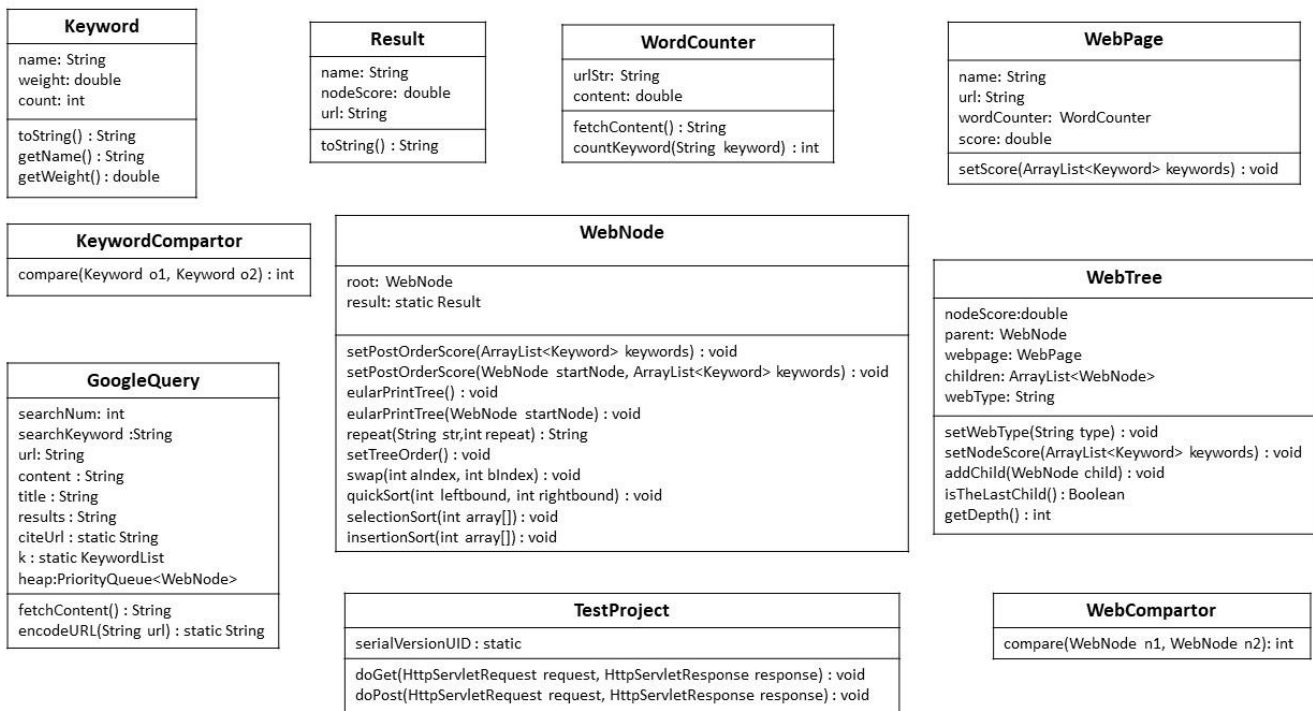


Figure 2. The UML Diagram

1. *Keyword* class

<i>Keyword</i>	
Modifier and type	Method (or Variable) and description
<b>Instance variable</b>	
<b>String</b>	name
<b>double</b>	weight
<b>int</b>	count
<b>Constructor</b>	
<b>Keyword(String name, double weight)</b> Enable to construct a <i>Student</i> object with given <i>name</i> , <i>weight</i> .	
<b>Instance methods</b>	
<i>getters</i>	getName(), getWeight()
<b>String</b>	toString() Return a String description of the keyword. <u><b>Sample output:</b></u> [海豚刑警 , 2]

2. *KeywordComparator* class

<b>KeywordComparator implements Comparator&lt;Keyword&gt;</b>	
Modifier and type	Method (or Variable) and description
<b>Instance methods</b>	
<b>int</b>	compare (Keyword o1, Keyword o2) Compare the count of the keywords and return (1, 0, -1) based on the result.



3. **Keyword** class

<b>Keyword</b>	
<b>Modifier and type</b>	<b>Method (or Variable) and description</b>
<b>Instance variable</b>	
<b>String</b>	name
<b>double</b>	weight
<b>int</b>	count
<b>Constructor</b>	
<b>Keyword(String name, double weight)</b> Enable to construct a <i>Student</i> object with given <i>name</i> , <i>weight</i> .	
<b>Instance methods</b>	
<b>getters</b>	getName(), getWeight()
<b>String</b>	toString() Return a String description of the keyword. <u><b>Sample output:</b></u> [海豚刑警 , 2]

4. **KeywordComparator** class

<b>KeywordComparator implements Comparator&lt;Keyword&gt;</b>	
<b>Modifier and type</b>	<b>Method (or Variable) and description</b>
<b>Instance methods</b>	
<b>int</b>	compare (Keyword o1, Keyword o2) Compare the count of the keywords and return (1, 0, -1) based on the result.

5. *KeywordList* class

<b>KeywordList</b>	
<b>Modifier and type</b>	<b>Method (or Variable) and description</b>
<b>Instance variable</b>	
<b>ArrayList&lt;Result&gt;</b>	lst The execution for user to connect the input with the database.
<b>Constructor</b>	
<b>KeywordList ()</b> Construct a KeywordList object and instantiate <b>ArrayList&lt;Result&gt;</b> lst.	
<b>Instance methods</b>	
<b>ArrayList&lt;Result&gt;</b>	getList () Return lst.
<b>void</b>	add(Result result) Add the result to lst.
<b>void</b>	sort() Use quickSort, bubbleSort, selectionSort, or insertionSort to sort items.
<b>private void</b>	quickSort(int leftbound, int rightbound) Implement quickSort.
<b>private void</b>	bubbleSort(int array[]) Implement bubbleSort.
<b>private void</b>	selectionSort(int array[]) Implement selectionSort.
<b>private void</b>	insertionSort(int array[]) Implement selectionSort.
<b>void</b>	swap(int a, int b) Swap the position of lst.
<b>void</b>	show() Show the result of sorting.

6. *WebComparator* class

<b>WebComparator implements from Comparator&lt;WebNode&gt;</b>	
<b>Modifier and type</b>	<b>Method (or Variable) and description</b>
<b>Instance methods</b>	
<b>int</b>	compare (WebNode n1, WebNode n2) Compare the nodeScore of the Webs and return a number based on the result.

7. *WebNode* class

<b>WebNode</b>	
<b>Modifier and type</b>	<b>Method (or Variable) and description</b>
<b>Instance variable</b>	
<b>double</b>	nodeScore
<b>WebNode</b>	parent
<b>WebPage</b>	webPage
<b>ArrayList&lt;WebNode&gt;</b>	children
<b>String</b>	webType
<b>Constructor</b>	
<b>WebNode(WebPage webPage)</b> Enable to construct a WebNode object and instantiate the webPage and children.	
<b>void</b>	setWebType (String type) Instantiate webType with given type.
<b>void</b>	setNodeScore (ArrayList<Keyword> keywords) Set the node score of keywords to arraylist.
<b>void</b>	addChild(WebNode child) Add the given child to children arraylist. Besides, set the child's parent is this.
<b>boolean</b>	isTheLastChild() Check whether it is the last child or not.
<b>int</b>	getDepth() Compute the depth of the node tree.

8. *WebPage* class

<b>WebPage</b>	
<b>Modifier and type</b>	<b>Method (or Variable) and description</b>
<b>Instance variable</b>	
<b>String</b>	url

<b>String</b>	name
<b>WordCounter</b>	wordCounter
<b>double</b>	score
<b>Constructor</b>	
<b>WebPage(String url,String name)</b> Enable to construct a WebNode object and instantiate the name, url and wordCounter. Besides, you also need to consider the UnsupportedEncodingException.	
<b>void</b>	setScore (ArrayList<Keyword> keywords) Set the score of keywords to arraylist.

### 9. *WebTree* class

<i>WebTree</i>	
Modifier and type	Method (or Variable) and description
<b>Instance variable</b>	
<b>WebNode</b>	root
<b>static Result</b>	result
<b>Constructor</b>	
<b>WebTree(WebPage rootPage)</b> Enable to construct a WebTree object and instantiate the root with given WebPage.	
<b>void</b>	setPostOrderScore(ArrayList<Keyword> keywords) Call the private void setPostOrderScore method.
<b>private void</b>	setPostOrderScore(WebNode startNode, ArrayList<Keyword> keywords) Implement setPostOrderScore.
<b>void</b>	eularPrintTree() Call the private void eularPrintTree method.
<b>private void</b>	eularPrintTree(WebNode startNode) Print the tree result include web pages and url.
<b>private String</b>	repeat(String str,int repeat) Return a string object.
<b>void</b>	setTreeOrder() Implement quickSort method.
<b>private void</b>	swap(int aIndex, int bIndex) Swap the position of root.children.
<b>private void</b>	quickSort(int leftbound, int rightbound) Implement quickSort.

<b>private void</b>	bubbleSort(int array[]) Implement bubbleSort.
<b>private void</b>	selectionSort(int array[]) Implement selectionSort.
<b>private void</b>	insertionSort(int array[]) Implement insertionSort.

10. *WordCounter* class

<i>WordCounter</i>	
Modifier and type	Method (or Variable) and description
<b>Instance variable</b>	
<b>String</b>	urlStr The website's nodeScore.
<b>String</b>	content The parent website.
<b>Constructor</b>	
<b>WordCounter (String urlStr)</b> Enable to construct a WordCounter object and instantiate the urlStr.	
<b>String</b>	fetchContent () Fetch the content of url
<b>int</b>	countKeyword(String keyword) Compute how many times does the keyword appear.

## 11. *TestProject*

```
TestProject
extends
HttpServlet
```

```
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Map.Entry;
import java.util.Properties;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.*;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class TestProject
 */
```

```
/**
 * @see HttpServlet#HttpServlet()
 */
public TestProject() {
    // Used as Main.java
    super();
}

/**
 * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
 */

protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

    // TODO Auto-generated method stub
    response.setCharacterEncoding("UTF-8");
    request.setCharacterEncoding("UTF-8");
    response.setContentType("text/html");

    int search = 20;
    if(request.getParameter("searchNum") != null) {
        search = Integer.parseInt(request.getParameter("searchNum"));
    }

    // search.jsp
    if(request.getParameter("keyword")== null) {
        String requestUri = request.getRequestURI();
        request.setAttribute("requestUri", requestUri);
        request.getRequestDispatcher("Search.jsp").forward(request,
response);
        return;
    }
}
```

```
KeywordList kLst = google.kLst;
for(int i = 0 ; i < kLst.lst.size() ; i++) {
    s[i][0] = kLst.lst.get(i).name;
    s[i][1] = kLst.lst.get(i).url;
}
request.getRequestDispatcher("googleitem.jsp").forward(request,
response);

}

/**
```

## 6.Challenges :

(1) There are too many search targets, how to capture the required information to filter, as well as making the results more accurate and efficient is a hard task.

(2) Front-End and Back-End connection issues and final conversion to a webpage or an app.

(3) The searching time is too long.



## 7. Work Division & Contributions:

Name	Contributions
呂學柏	Back-End, Front-End, BE/FE Connection, Debugging, Slides
黃尹彤	UML Diagram, Proposals, Slides
劉家妤	UML Diagram, Proposals, Slides