Final Project Proposal — Music Festival

1. Members :

109306061 呂學柏 109306043 黃尹彤 109306060 劉家妤

Instruction:

(1)Topic :

The information of domestic music festivals

(2)User :

For people who love rock bands, metal bands, and indie bands, 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)Keywords & weight :

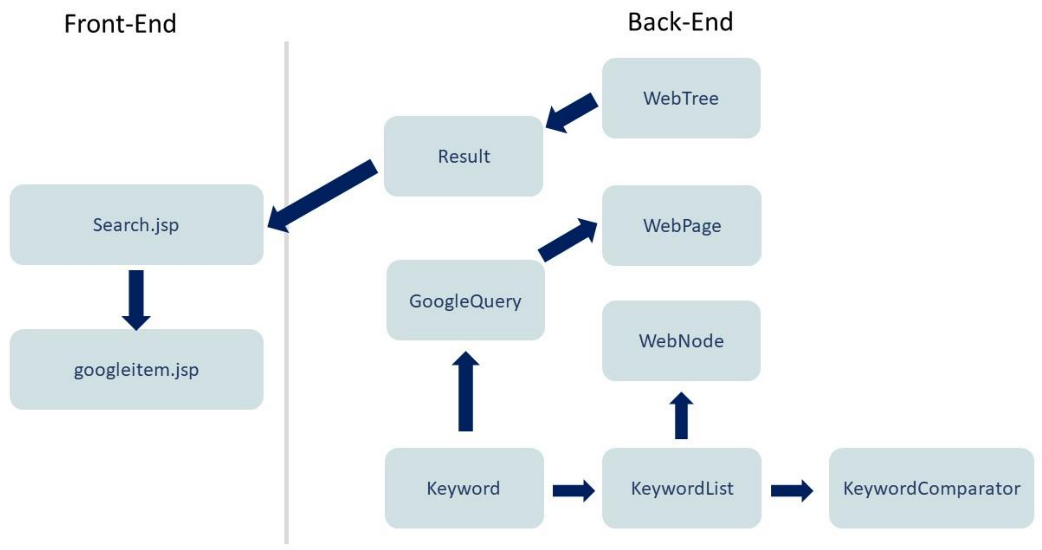
Keywords	Weight
音樂祭	20.0
獨立音樂	10.0
獨立樂團	10.0
樂團	10.0
台灣	4.5
臺灣	4.5
陣容	3.0
票價	3.0
搖滾	3.0
龐克	3.0

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
人力銀行	-20.0
臺語萌點	-20.0
教育雲	-20.0

(4) The Score Counting Formula : score = score + keyword.weight * wordCounter.countKeyword(keyword.name)

5. Class Diagram :



6. Schedule :

Date	Schedule
11/1-11/7	Proposal
11/8-11/15	Midterm Exam Week
11/16-11/23	View Webpage, Collect and 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 : Nodes & Web Trees & Ranking Front End Development : User Interface (Input & Output) and the rest
12/16-12/22	Front End Development : User Interface(I/O)
12/23-12/30	Debug &Testing
12/31-1/6	PPT Design and Demo

1/7-1/14	Upload Project and Codes

7. UML Diagram :

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

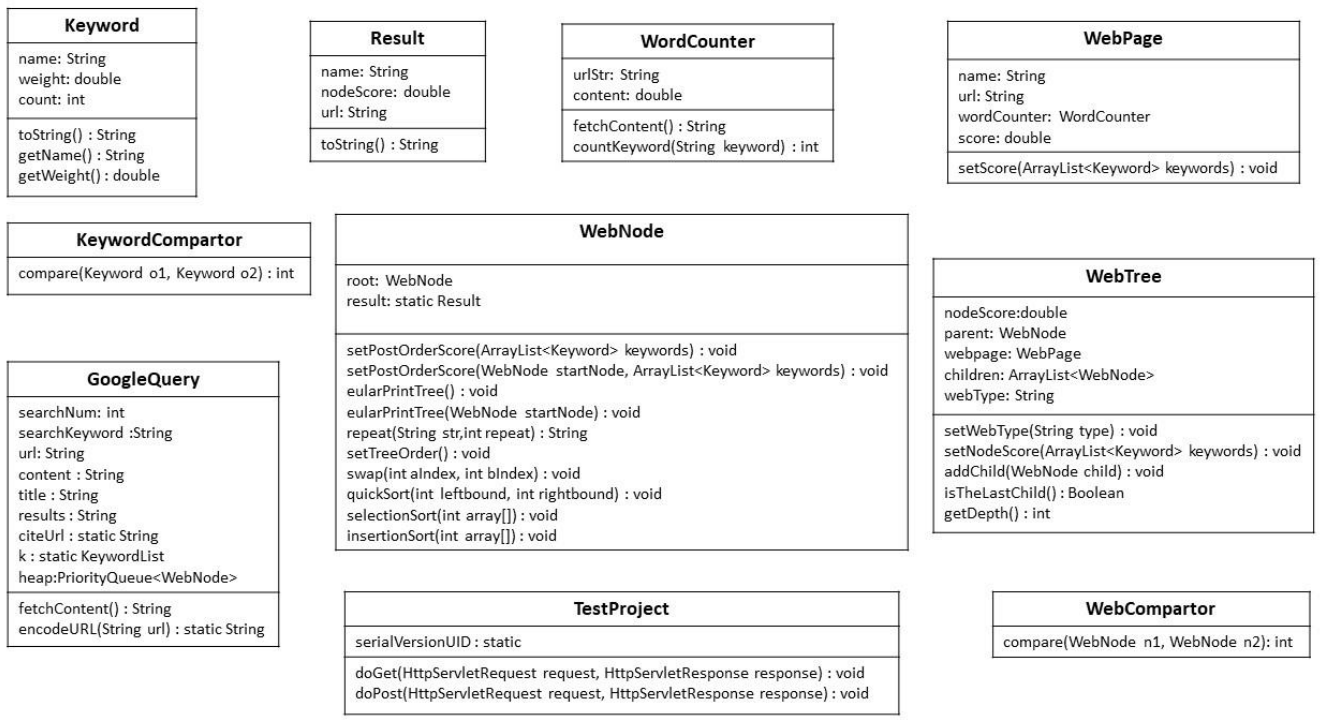

Keyword
name: String
weight: double
count: int
toString() : String
getName() : String
getWeight() : double
Result
name: String
nodeScore: double
url: String
toString() : String
WordCounter
urlStr: String
content: double
fetchContent() : String
countKeyword(String keyword) : int
WebPage
name: String
url: String
wordCounter: WordCounter
score: double
setScore(ArrayList<Keyword> keywords) : void
KeywordCompartor
compare(Keyword o1, Keyword o2) : int
WebNode
root: WebNode
result: static Result
setPostOrderScore(ArrayList<Keyword> keywords) : void
setPostOrderScore(WebNode startNode, ArrayList<Keyword> keywords) : void
eularPrintTree() : void
eularPrintTree(WebNode startNode) : void
repeat(String str,int repeat) : String
setTreeOrder() : void
swap(int aIndex, int bIndex) : void
quickSort(int leftbound, int rightbound) : void
selectionSort(int array[]) : void
insertionSort(int array[]) : void
WebTree
nodeScore:double
parent: WebNode
webpage: WebPage
children: ArrayList<WebNode>
webType: String
setWebType(String type) : void
setNodeScore(ArrayList<Keyword> keywords) : void
addChild(WebNode child) : void
isTheLastChild() : Boolean
getDepth() : int
GoogleQuery
searchNum: int
searchKeyword :String
url: String
content : String
title : String
results : String
citeUrl : static String
k : static KeywordList
heap:PriorityQueue<WebNode>
fetchContent() : String
encodeURL(String url) : static String
TestProject
serialVersionUID : static
doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void
WebCompartor
compare(WebNode n1, WebNode n2): int

Figure 2. The UML Diagram

1. ***Keyword*** class

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

2. ***KeywordComparator*** class

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

3. ***Keyword*** class

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

4. ***KeywordComparator*** class

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

5. ***KeywordList*** class

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

6. ***WebComparator*** class

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

7. ***WebNode*** class

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

8. ***WebPage*** class

WebPage	
Modifier and type	**Method (or Variable) and description**
Instance variable	
String	url

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

9. ***WebTree*** class

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

private void	bubbleSort(int array[]) Implement bubbleSort.
private void	selectionSort(int array[]) Implement selectionSort.
private void	insertionSort(int array[]) Implement insertionSort.

10. ***WordCounter*** class

WordCounter	
Modifier and type	**Method (or Variable) and description**
Instance variable	
String	urlStr The website's nodeScore.
String	content The parent website.
Constructor	
WordCounter (String urlStr) Enable to construct a WordCounter object and instantiate the urlStr.	
String	fetchContent () Fetch the content of url
int	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);

	 }

	 /**
```

8.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.