Dexter

Making search web 2.0 ready!

Submitted as a

Final Year Project

For the partial completion of

B. Tech in Computer Science and Engineering

By

Vaibhav Bajpai

Rahul Burman

Nupur Dixit

Yadavendra

Under the guidance of

Mr. Sanjeev Pippal

Associate Professor

Department of Computer Science and Engineering

Galgotias College of Engineering and Technology

Dexter

Making search web 2.0 ready!

CERTIFICATE OF ORIGINALITY

This to certify that the project entitled 'Dexter' has been completed successfully in Galgotias College of Engineering Technology by:-

Rahul Burman (0509710079)

Vaibhav Bajpai (0509713060)

Nupur Dixit (0509710063)

Yadavendra (0509710121)

Guide	Head of Department(Computer Science)
Mr. Sanjeev Pippal	Prof. Parmanand Astya

Dexter

Making search web 2.0 ready!

ACKNOWLEDGEMENT

We began a journey last year not knowing what perils lay in our path but only with a determination to be able to create a product, off which ourselves would be proud. Now when we see Dexter our heart gets filled with joy and eyes with tears as we remember all the effort, sacrifices and toils we underwent to that made this day possible.

In this moment of happiness we would like to thank Prof. Parmanand (HOD-CS) and our guide Mr., Sanjeev Pippal. If it was not for their guidance and encouragement this project would have remained a dream. I would also like to thank Mr. Vimal Gupta and all other faculty members who constantly supported us during the development of the project.

Dexter

Making search web 2.0 ready!

ABSTRACT

Objective – to blend social networking into the Google search engine and encapsulate it into a Web 2.0 browsing environment.

Inadequacies in Google search engine –

 The Google search results are still currently “only” ordered based on a page ranking algorithm completely isolating the ability of the user to provide a feedback to the search results.

 The search engine does not still provide the ability to bookmark the search results in the profile, resting the functionality to be provided by the browser (that stores bookmarks in local storage) or third party web solutions to keep them in the “cloud”.

 The Google homepage rests waiting for the user to provide the keyword before some “information content” could be made available.

 Google does not provide possibility of collaborating a search with a friend!

Solution –

• Dexter would enable users to vote the search results and comment on them, still preserving the current page rank algorithm

• Dexter would provide a full-fledged social bookmarking using tags on top of Google search results and save ‘em in the cloud.

• Dexter would also include another interface “Dexter Surf”, that would “push” the top voted content in a specific category or a web service(like YouTube) for a

desired timeframe in the homepage itself, providing “information content” as soon as one logs in.

• Dexter would maintain profile pages and provide ability to make friends whose activities could be followed, thereby increasing collaboration

Dexter

Making search web 2.0 ready!

TABLE OF CONTENTS

Dexter

Making search web 2.0 ready!

Dexter

Making search web 2.0 ready!

FEASIBILITY STUDY

Objective *TO BLEND SOCIAL NETWORKING INTO A SEARCH ENGINE AND ENCAPSULATE IT TO A WEB 2.0 BROWSING ENVIRONMENTS.*

-

Implementation Alternatives

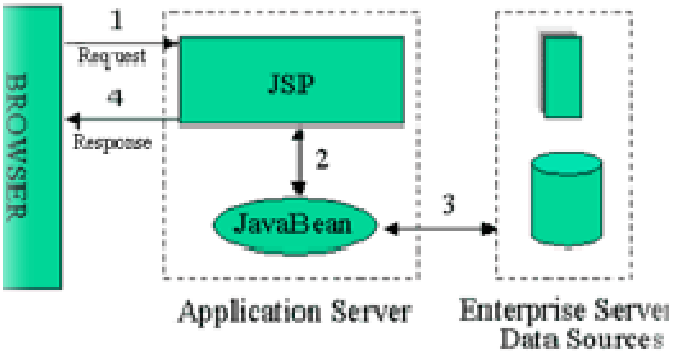


FIGURE 2: MODEL 1 ARCHITECTURE

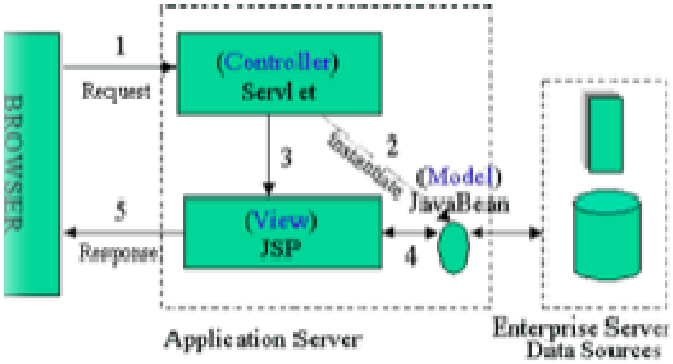


FIGURE 1: MODEL 2 ARCHITECTURE

Architectural Alternative

The Java EE Framework builds upon one of the two aforementioned web component architectures

Dexter

Making search web 2.0 ready!

Dexter is proposed to use the MVC Design Pattern (Model 2) with the view of completely isolating the presentation layer from the business logic to ensure higher degree of maintainability, reusability, and security of the product. This would eventually guarantee that the project would be able to overlay its Web 2.0 capabilities over any search engine whatsoever with minimum effort.

Moreover the inculcation of a full fledged ORM framework for back-end processing demands a design pattern as sophisticated as MVC in itself.

Economic Feasibility

The expected economic cost of the project is as follows

- Cost of Software Packages
  - There was a conscious decision to use Free and Libre' Open Source Software (Database, Package, IDE, SDK's, Application Servers), so the economic liability to their usage is ruled out.
- Cost of Hardware.
  - We would require 4 machines having at least 512MB ram and Pentium IV architecture. To develop this project
- Software Cost Estimation Through COCOMO
  - Dexter is an intermediate (in size and complexity) software projects in which team member have mixed experience levels in the technologies being used we categorize 'Dexter' as an intermediate project.

Intermediate COCOMO computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes. This extension considers a set of four "cost drivers", each with a number of subsidiary attributes. Given next is how Dexter evaluates to these attributes.

Drivers	**Ratings**	**EAF**
Product attributes		
Required software reliability	Normal	1.00
Size of application database	High	1.08
Complexity of the product	Very High	1.30
Hardware attributes		
Run-time performance constraints	High	1.11
Memory constraints	Normal	1.00

Volatility of the virtual machine environment	Low	0.87
Required turnaround time	Normal	1.00
Personnel attributes		
Analyst capability	Low	1.19
Applications experience	Normal	1.00
Software engineer capability	Low	1.17
Virtual machine experience	Normal	1.00
Programming language experience	Normal	1.00
Project attributes		
Use of software tools	Very High	0.82
Application of software engineering methods	Very High	0.83
Required development schedule	High	1.04
Total effort adjustment factor		**1.34**

The Intermediate COCOMO formula is of the form:

$E=a_i(\text{KloC})^{(b)}.\text{EAF}$

Where **E is the effort applied in person-months**, **KloC** is the estimated number of thousands of delivered lines of code for the project, and **EAF** is the factor calculated above. The coefficient $\mathbf{a_i}$ = 3.0 and exponent $\mathbf{b_i}$ = 1.12 for a semi detached project.

Dexter is expected to have a delivered line of code of around 10 Kloc

$E= 3(10)^{(1.12)}.1.34$

E = 53

Average staffing = (53 Person-Months) / (11 Months) = 4.81 people:

Technical Feasibility

Programming Language

A number of languages can be used for developing a web-app today. Most of them are well equipped with web framework that uses the MVC architecture. Some of the

most prominent languages used in web-application development and their corresponding MVC framework are given below.

- **Java**
  - Struts
  - Java Server Faces
  - Spring
- **Python**
  - Django
  - Pylons
- **Ruby**
  - Ruby on Rails
  - Nitro
- **Perl**
  - Catalyst
- **.Net Languages**
  - ASP .net MVC Framework
  - Monorail

Dexter Team has prior experience of coding in Java using Struts, thus it was chosen as the web framework.

Operational Feasibility

- DESIGN GOALS OF DEXTER INCLUDE
- CONSISTENT LOOK AND FEEL ACROSS THE WEB-APPLICATION
- SIMPLE USER INTERFACE
- FEEDBACK AFTER EVERY ACTION
- INFORMATION TO BE DISPLAYED IN A STRUCTURED MANNER BY GROUPING SIMILAR INFORMATION
- TOLERANCE TOWARDS USER ERROR (IGNORING WRONG INPUT, PROVIDING EASY RECOVERY OR GRACEFUL FALL)

Dexter is a web-application which is being designed keeping in mind the least common denominator in terms of the user's technological capability which amounts a user with only a basic understanding of computers enabling any person who uses internet to become its prospective user.

Dexter

Making search web 2.0 ready!

Political Feasibility

As we use open source or/and free technology we don't expect to face any legal or political difficulty.

Dexter

Making search web 2.0 ready!

PROJECT PLAN

Dexter has been divided into four modules.

MODULE NAME	*DESIGNATED TO*
DATABASE HANDLING USING HIBERNATES	Rahul Burman
MVC USING STRUTS	Vaibhav Bajpai
PRESENTATION LAYER	Nupur Dixit
ACCESSORIES	Yadavendra Yadav

Schedule decided for Dexter is as follows

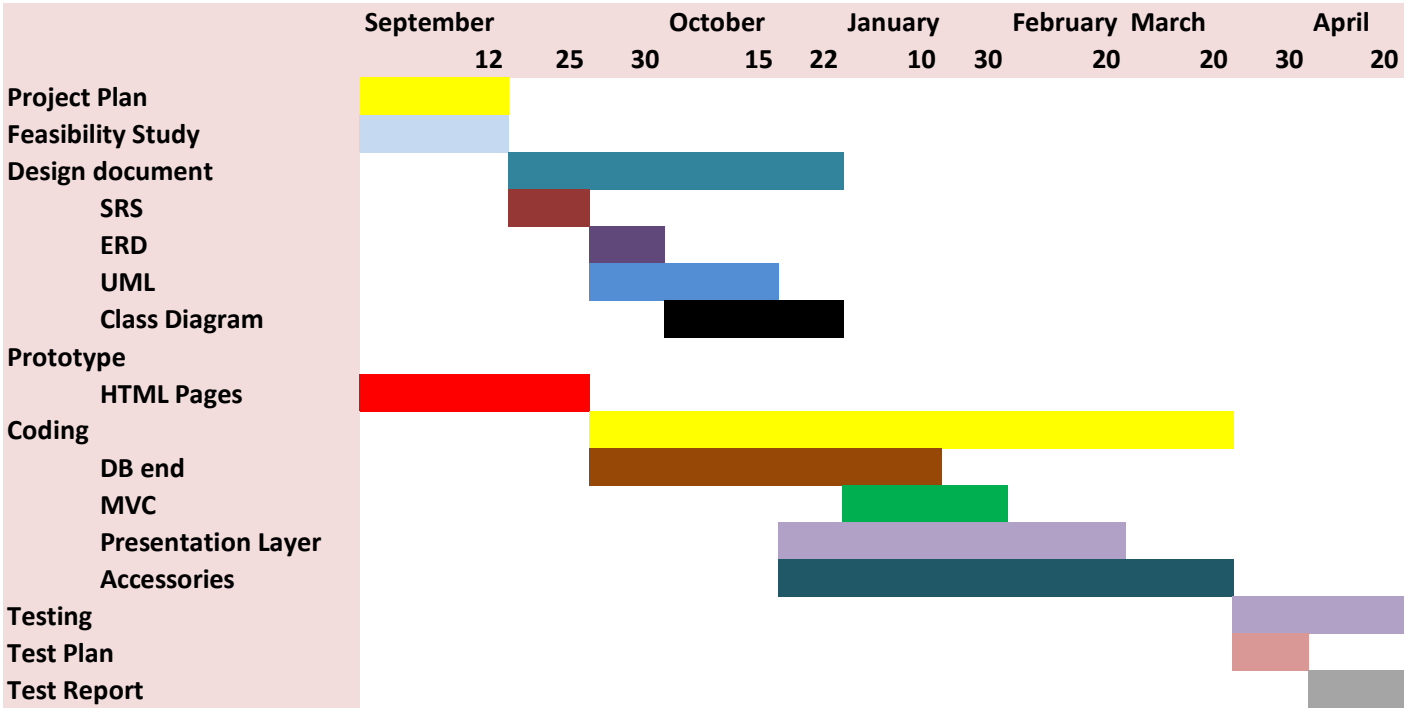
PHASE	*DATE OF COMPLETION*
PROJECT PLAN	12th September
FEASIBILITY STUDY	12th September
DESIGN DOCUMENT	
SRS	25th September
ERD	30th September
UML	15th October
Class Diagram	22nd October
PROTOTYPE	
HTML Pages	25th September
CODING	
Database and Access Layer	10th January
MVC	30th January

Dexter

Making search web 2.0 ready!

Presentation Layer	20th February
ACCESSORIES	20th March
TESTING	
Test Plan	30th March

Time Line Chart



Dexter

Making search web 2.0 ready!

SOFTWARE REQUIREMENTS SPECIFICATION

INTRODUCTION

PURPOSE

The purpose of the document is to describe the system requirements of the project 'Dexter'. The document contains sufficient detail in the functional system requirements so that a full-fledged design solution can be devised. The functional level requirements are specified using the Use Case Model approach.

The document would place borders around the problem, solidify ideas, and help break down the problem into its component parts in an orderly fashion.

The document would also serve as the parent document for testing and validation strategies that will be applied to the requirements for verification.

Dexter is a product which allows user to search the web. It differs from existing product in the market like Google, yahoo by taking an alternative methodology to search by taking in account the human opinion.

DOCUMENT CONVENTIONS

The terms included in single quotation mark signify terms having special meaning in context of 'Dexter' and their corresponding meaning have been explained in Glossary.

Dexter

Making search web 2.0 ready!

INTENDED AUDIENCE AND READING SUGGESTIONS

The intended audience of this document is people interested in understanding the basic requirement and functionality of 'Dexter'.

PRODUCT SCOPE

Objective – to blend social networking into the Google search engine and encapsulate it into a Web 2.0 browsing environment.

Goals –

- The search results should be ordered in a way that they give preference to reader's opinion over algorithmic or machine based approaches.
- The non submitted search results should show up in the same order as that by the page rank algorithm
- The search interface should be as simplistic as the Google search engine homepage.
- The user should be able to separately 'submit' a URL to the Dexter database.
- Any URL ever 'voted up' or submitted should be logged in for future reference by the user.
- The user should be able to 'favorite' the voted results, and be able to tag them.
- User should be able to add 'friends'.
- The user should be able to follow the activity of his 'friends' and be able to 'shout'/'email' them of an interesting content.
- The privacy of the user profile should be maintained.
- User should be able to choose one of the two interfaces as default home ('Surf' /'Search')
- Mechanism should exist to weed out 'spam' and 'inappropriate' url and comment

Dexter

Making search web 2.0 ready!

Benefits –

• Dexter implements the social search paradigm where relevance of search results is determined by considering the interactions or contributions of users, in contrast to established algorithmic or machine-based approaches where relevance is determined by analyzing the text of each document or the link structure of the documents

• Dexter would enable users to vote the search results and comment on them, still preserving the current page rank algorithm

• Dexter would provide a full-fledged social bookmarking using tags on top of Google search results and save 'em in the cloud.

• Dexter would also include another interface "Dexter Surf", that would "push" the top voted content in a specific category or a web service(like YouTube) for a desired timeframe in the homepage itself, providing "information content" as soon as one logs in.

• Dexter would maintain profile pages and provide ability to make friends whose activities could be followed, thereby increasing collaboration

• Reduces impact of link spam on search result by relying less on link structure of web pages

• Increased relevance because each result has been selected by users

• leverage a network of trusted individuals by providing an indication of whether they thought a particular result was good or bad

• The introduction of 'human judgment' suggests that each web page has been viewed and endorsed by one or more people, and they have concluded it is relevant and worthy of being shared with others using human techniques that go beyond the computer's current ability to analyze a web page.

• Web pages are considered to be relevant from the reader's perspective, rather than the author who desires their content to be viewed, or the web master as they create links.

• As 'Dexter' would be constantly getting feedback it is potentially able to display results that are more current or in context with changing information

REFERENCES

Software Requirements Specifications – Wikipedia

Dexter

Making search web 2.0 ready!

OVERALL DESCRIPTION

PRODUCT PERSPECTIVE

Dexter is assumed to be a replacement for current conventional Web 1.0 search engines, which are based upon an algorithm that decides the value/order of a search result URL based upon a fixed criteria *(link recommendation as used in Google or keyword based search as used in Yahoo!).*

These search engines do not involve the human element in deciding the effectiveness of a search result URL which seems inadequate in today's Web 2.0 scenario.

The philosophy behind "Dexter" is that humans are the best judge of a page's relevance and it's their opinion which is given a higher priority over algorithmic approaches.

PRODUCT FUNCTIONS

- The user should be able to search the web without having to leave 'Dexter'
- The user should be able to search without being logged in, though should not be allowed to vote or comment.
- The user should be able to separately submit a URL to the Dexter database.
- The user should be able to bookmark the voted results, and be able to tag them.
- The user should be able to see its recent activity as in comments and votes.
- The user should be able to follow the activity of its friends and be able to shout/email them of an interesting content.
- User should be able to choose one of the two interfaces as default home ('Surf' / 'Classic')
- User should be able to add Dexter friends from Google contacts.
- User can view the article that have been endorsed maximum no of times in various 'category', 'channel' or 'timeframe'
- Categories and channel can be added/removed dynamically.
- Any url/comment can be reported by the user after the user has 'voted it down' and moderators would take decision on the reported URL and comment.

Dexter

Making search web 2.0 ready!

USER CLASSES AND CHARACTERISTICS

Users

Dexter treats every “user” as equal invariant of one’s technical expertise, educational level, or experience to reflect transparency in the sanctity of the votes.

Moderators

Will have the privilege to

- delete a submitted url based on profanity of content
- delete a submitted comment based on profanity of content
- Change the category of submitted URL based on reports received.

Administrators

- Have privilege to add/remove categories and order them as would appear on the web-page.
- Have privilege to add/remove channel and order them as would appear on the web-page.
- have privilege to ban user
- have privilege to add/remove IM service name offered to user to supply their IM details
- have privilege to add/remove a visibility control level
- Assign moderator privilege to users.

Dexter

Making search web 2.0 ready!

OPERATING ENVIRONMENT

Client-Side Requirements

Hardware Platform –

Dexter, being a web-application only restricts the hardware requirements for the client to the capability of a machine being able to connect to the internet or the LAN in which the web application is deployed to

Operating System –

Dexter, being a web-application would allow the client side interface to run on all Operating Systems that have an inbuilt network stack incorporated into the kernel.

Software Platform –

Browser – IE 6+, Firefox 2+, Safari, Opera, Chrome

Server-Side Requirements

Hardware Platform –

- at-least 256 MB RAM
- 10 GB HDD space

Operating System –

- 32-bit Linux/Windows Operating System
- 2 GB Page File/Swap

Software Platforms –

- Java SE 6
- Java EE 5 compatible Application Server (Glass fish)
- Apache Web Server
- Struts and Hibernate libraries.

Dexter

Making search web 2.0 ready!

DESIGN AND IMPLEMENTATION CONSTRAINTS

Paradigm Constraints

- As users can directly add results to a social search engine there is a risk that some users could insert search spam directly into the search engine. Elimination or prevention of this spam would require the ability to detect the validity of a users' contribution, such as whether it agrees with other trusted users.

- There are so many unique searches conducted that most searches, while valid, are performed very infrequently. A search engine that relied on users filling in all the searches would be at a disadvantage to one that used machines to crawl and index the entire web.

- The application would not be much useful to users until it acquires a reasonable user base.

Hardware/Software Limitations

- Oracle 10XE used in the project is not available for Mac OSX
- Java SE 6 is not available for 32-bit hardware running Mac OSX

Software Constraints

- Explicit usage of Free and/or Open Source Software
- Explicit Avoidance of Bloated Packages (Netbeans)

Design Constraints:

- No Explicit Style code in HTML
- No Flash Content
- No Frames
- No Tables for Layout Design

Coding Constraints:

- No Business Logic in Presentation Layer
- Separation of Form Beans and Data Beans
- Security Layer in Authentication

Dexter

Making search web 2.0 ready!

- Resolving the Object-Relational Mismatch
- Explicit HTML Coding using HTML Editor than using RAD tools

USER DOCUMENTATION

- Java Docs for the source.
- About/FAQ in the Web-application website.
- Video Screen-cast explaining the features/usage.

ASSUMPTIONS AND DEPENDENCIES

- Dexter being an overlay on top of the search engine, assumes the 24x7 functioning of the search engine in itself.

- Dexter uses Google AJAX Web Search API to retrieve the search results pushed against a query, and assumes the API would remain open to public.

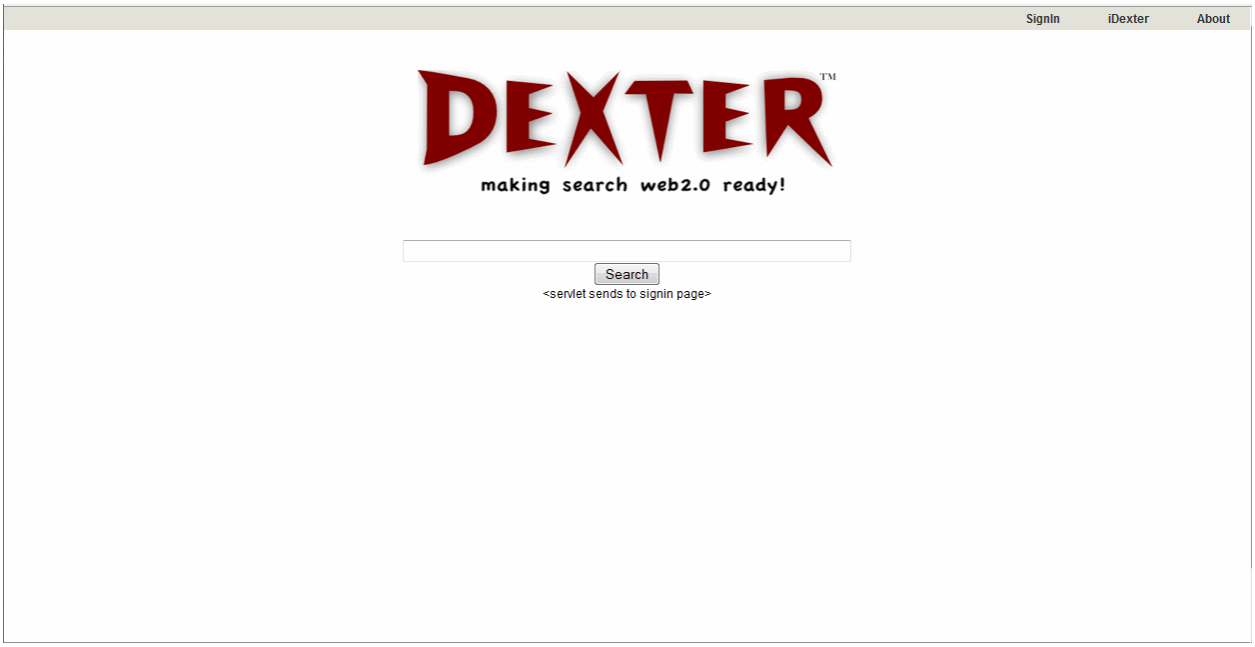
- Dexter uses Google Account Authentication API to retrieve contact details of a user, and assumes the API would remain open to public.

Dexter

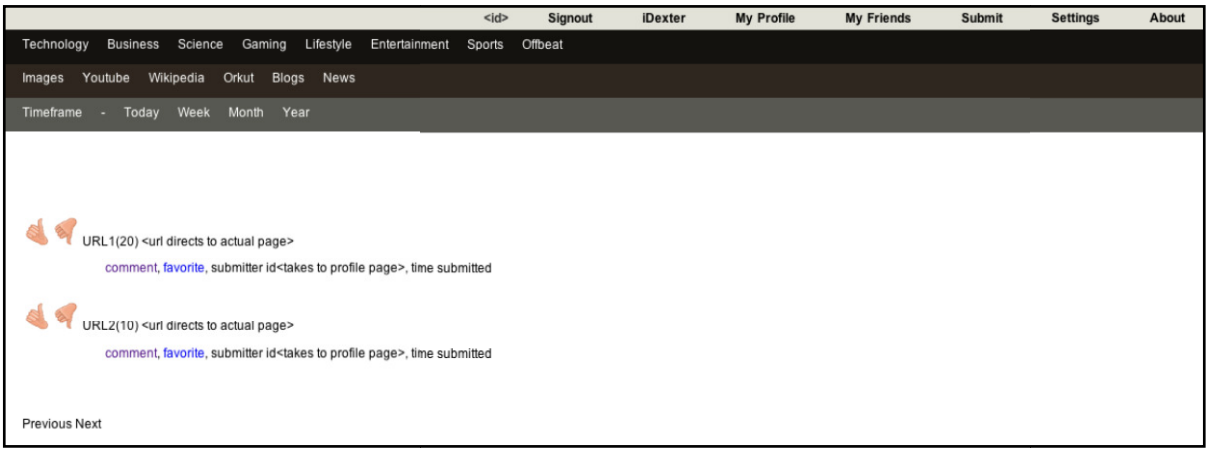
Making search web 2.0 ready!

EXTERNAL INTERFACE REQUIREMENTS

USER INTERFACES



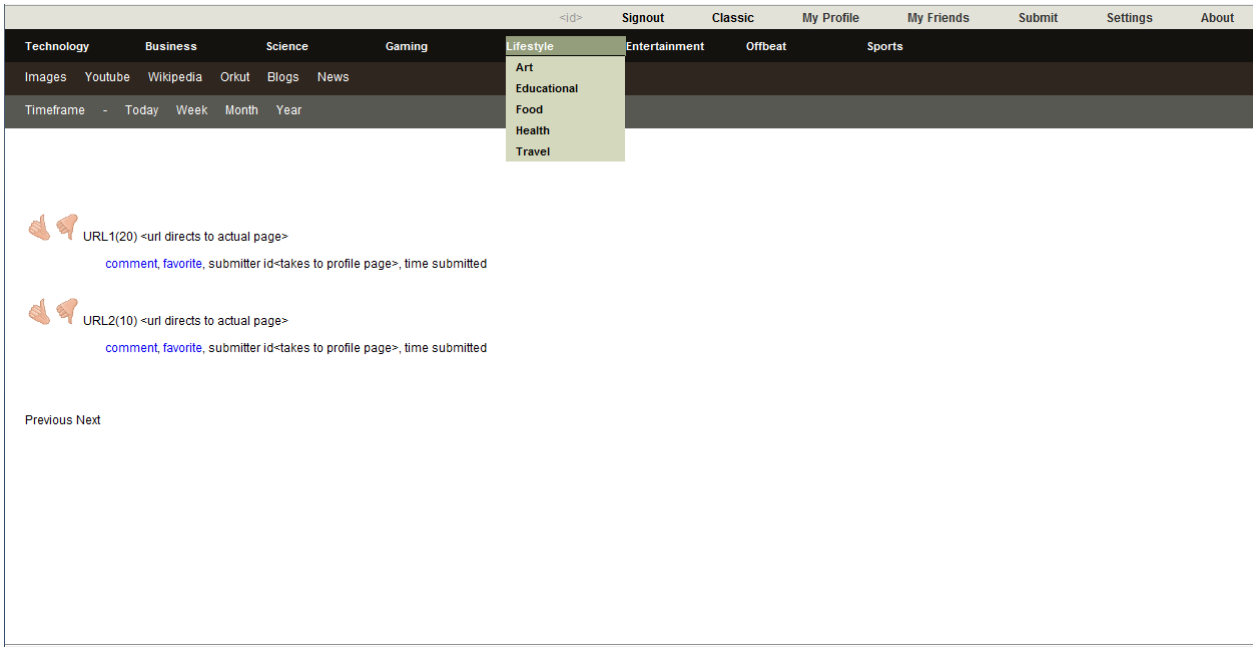
Dexter Search Page



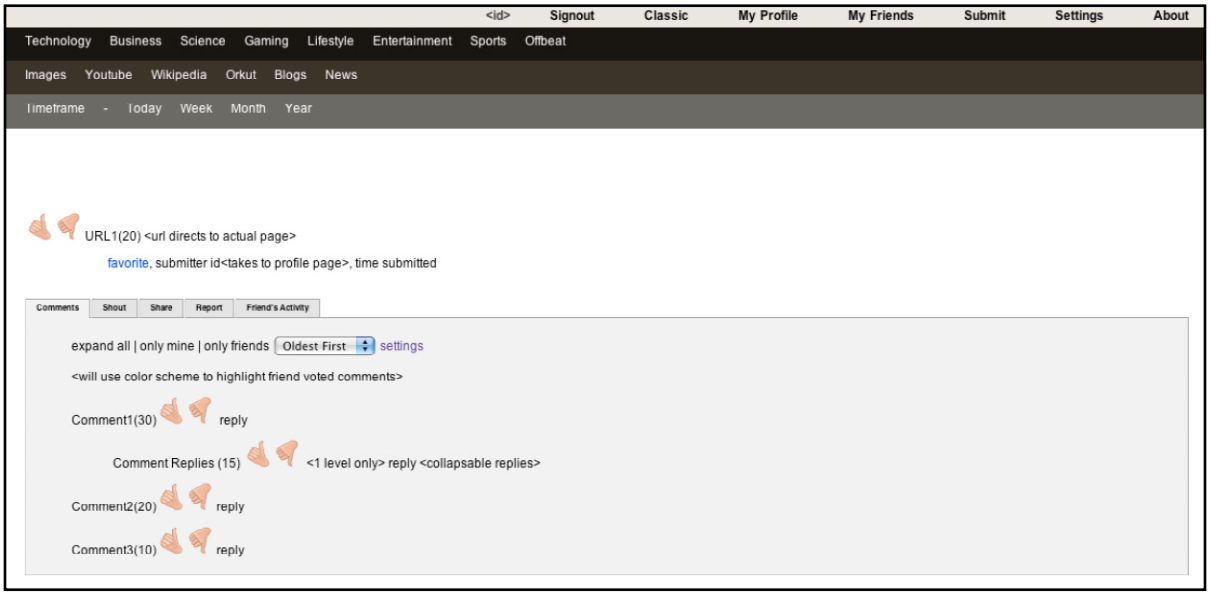
Search Results Page – Higher voted results on top.

Dexter

Making search web 2.0 ready!



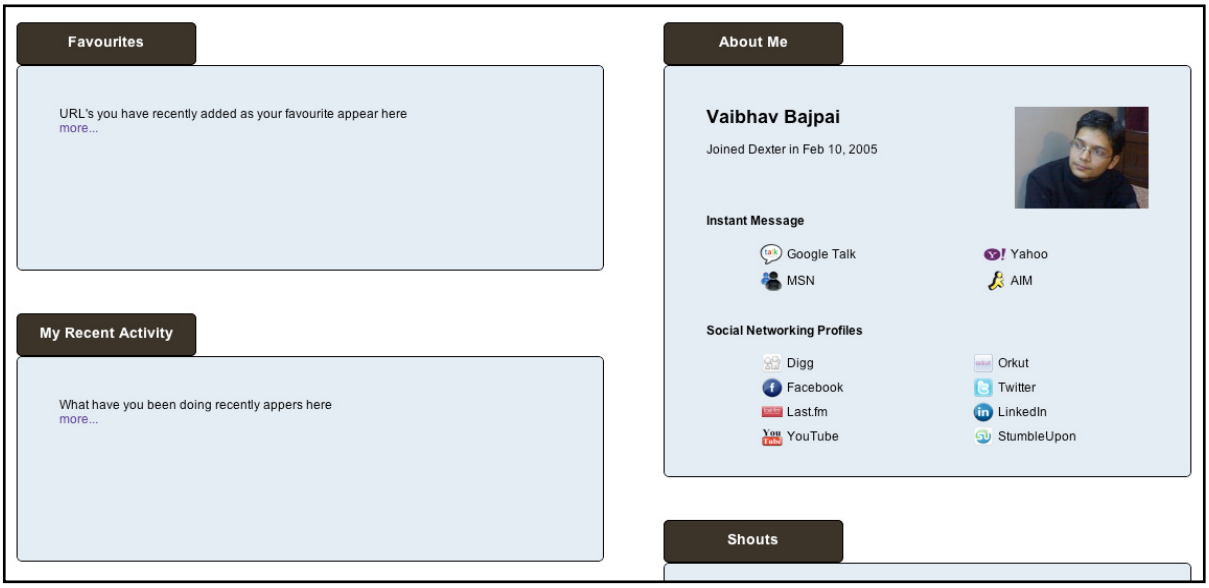
Dexter Surf (URLs belonging to a chosen category are listed)



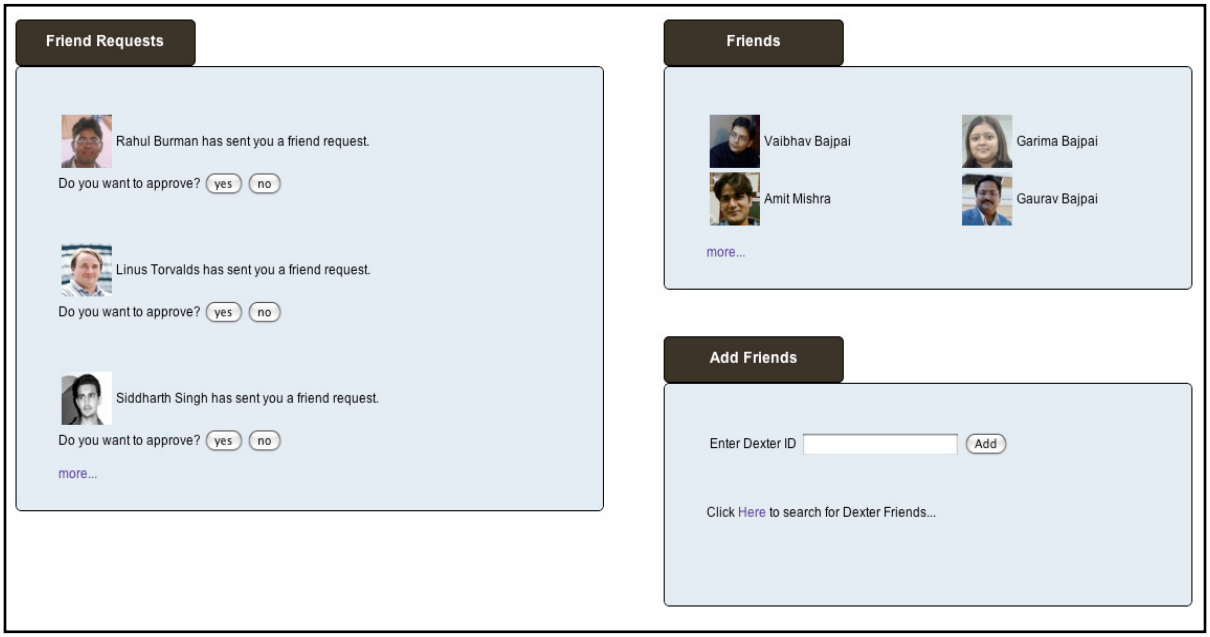
Action specific to a particular URL

Dexter

Making search web 2.0 ready!



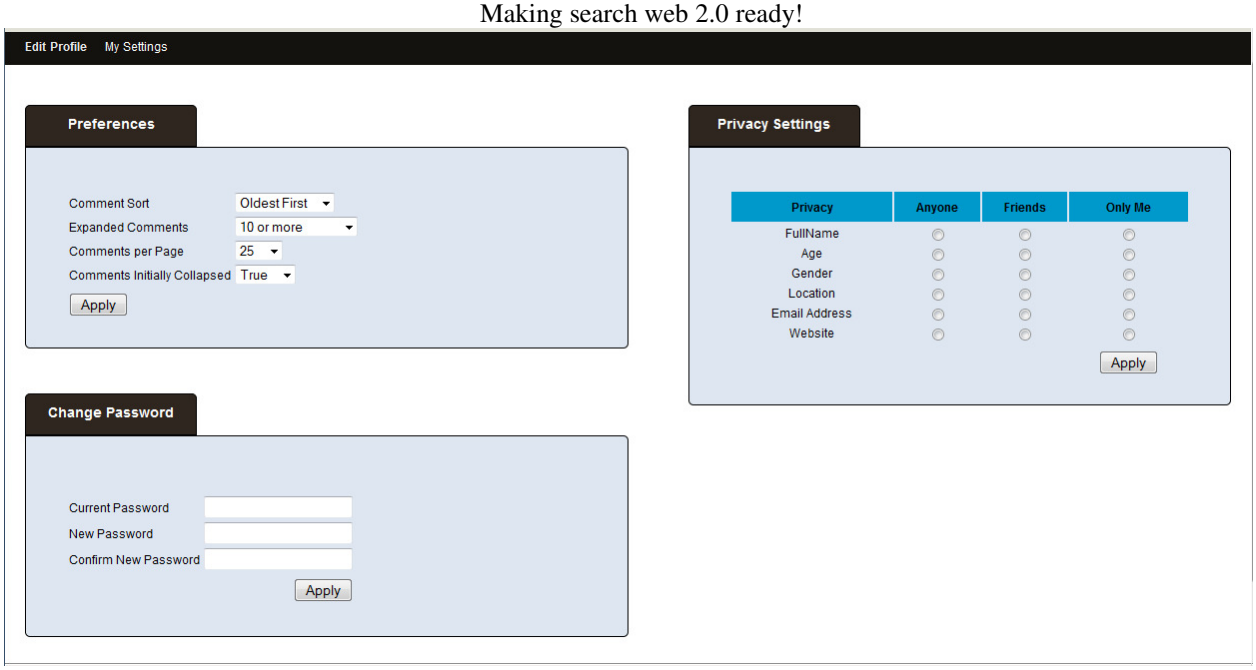
User Profile Page



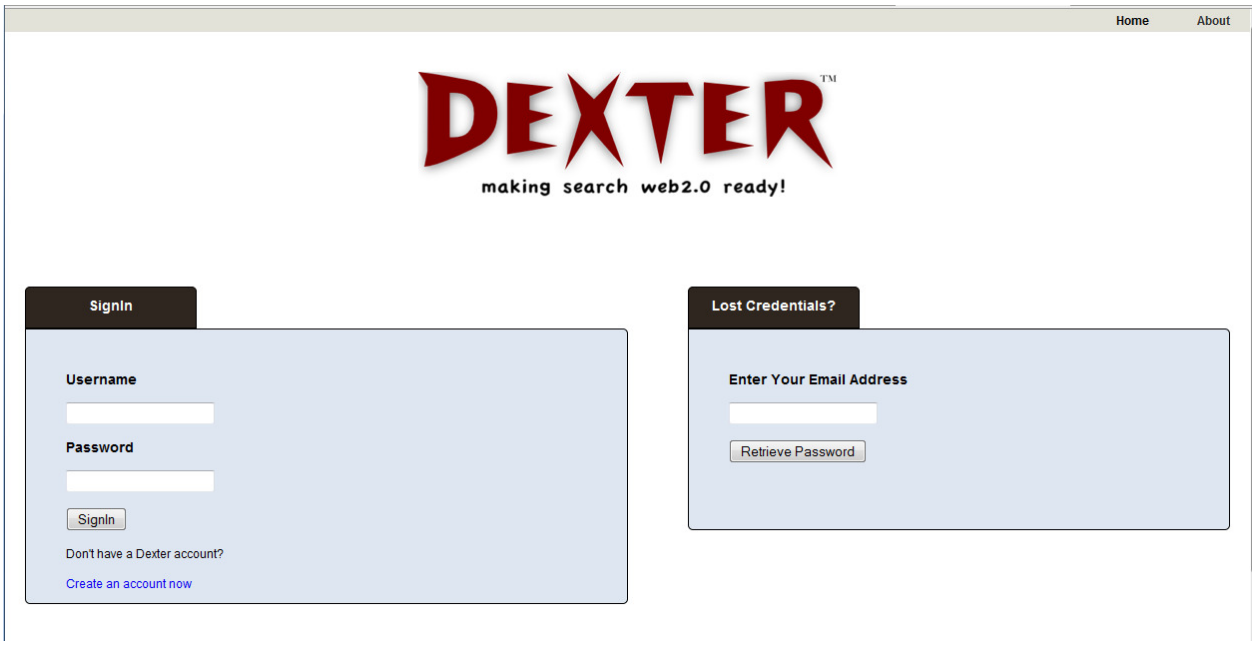
User's Friends Page

Dexter

Making search web 2.0 ready!



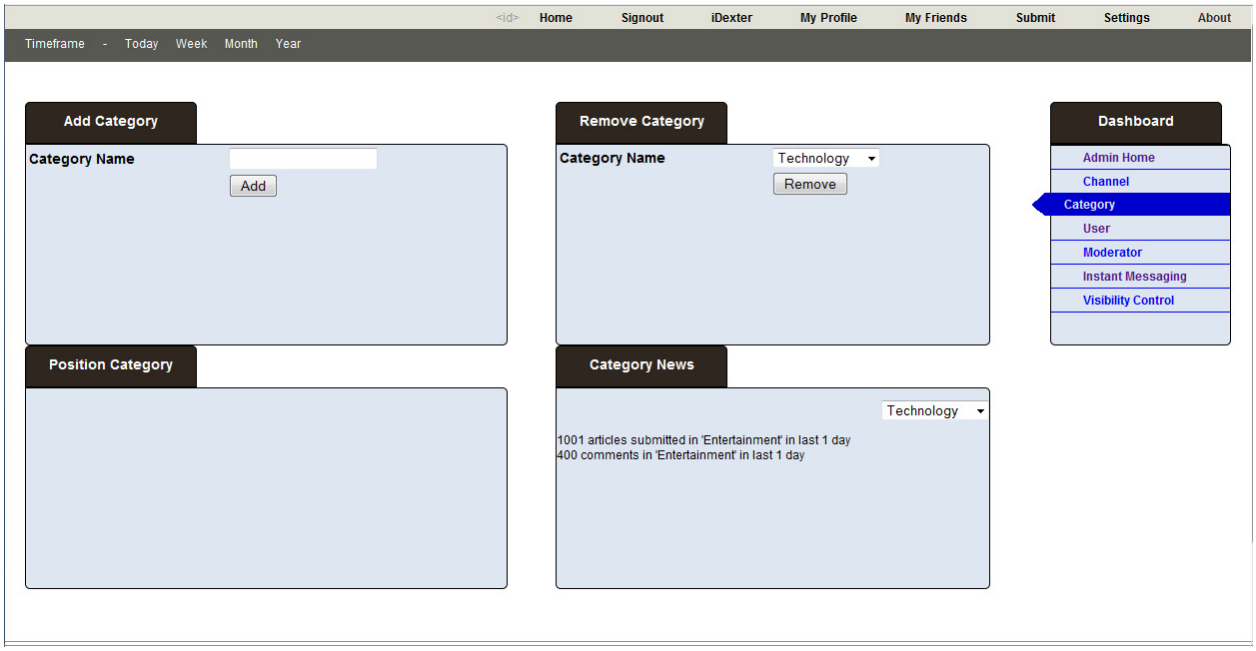
Settings Page



Sign In Page

Dexter

Making search web 2.0 ready!



Admin Page

Dexter

Making search web 2.0 ready!

HARDWARE INTERFACES

Communication Protocols

- HTTP 1.1
- SSL for Login Authentication.

Minimum Number of Machines

- one or more - Database Storage
- one - Application Server Deployment (Server)
- one or more - Client

SOFTWARE INTERFACES

Database will be distributed across machines, the recordset would be retrieved by the application server deployed at another machine, where the result sets would map to EJB modules inside the EJB container which will be pushed to the Web Container for the servlets to work upon and build presentation level java beans for the JSP to invoke and display the results as standard HTML page, which would be sent to the client upon a GET/POST request.

COMMUNICATIONS INTERFACES

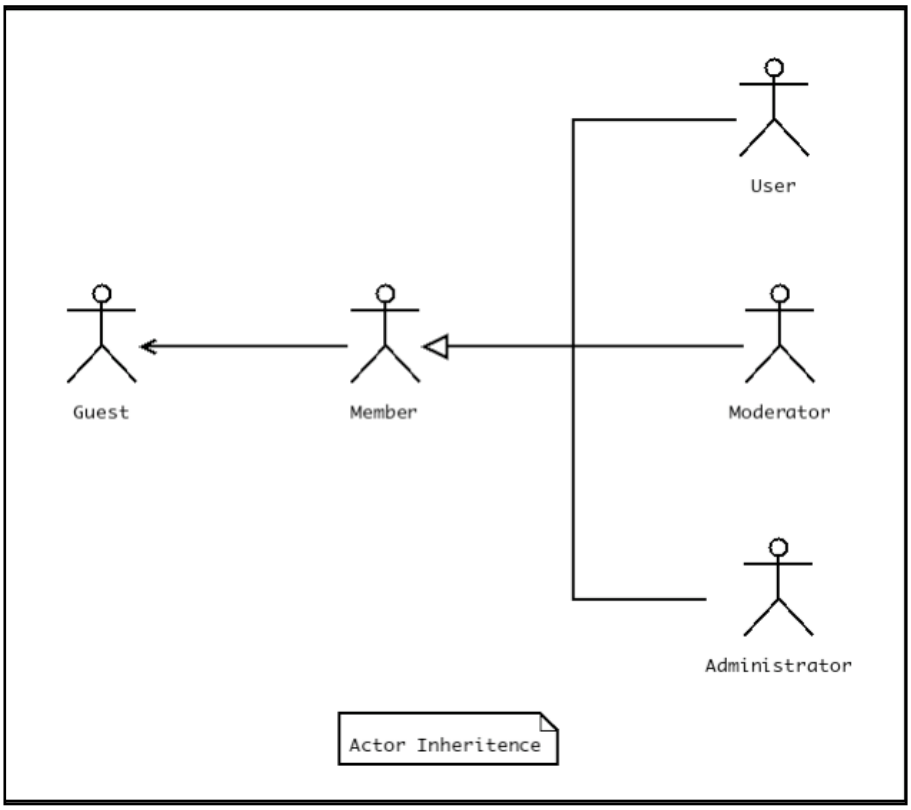
- Web-browser – the web-app would be accessed by the client using a web browser
- Protocols – HTTP 1.1, SMTP for sending email to friends
- Security – SSL for login authentication
- Data Transfer – the server response would be gzipped for efficiency.

Dexter

Making search web 2.0 ready!

SYSTEM FEATURES

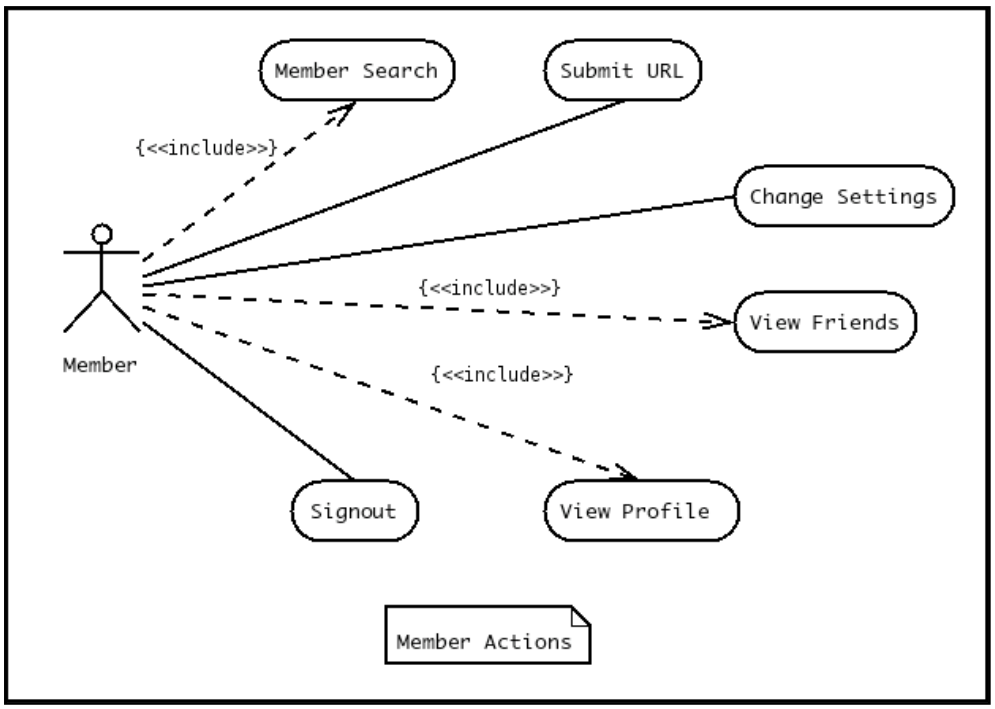
ACTOR INHERITANCE



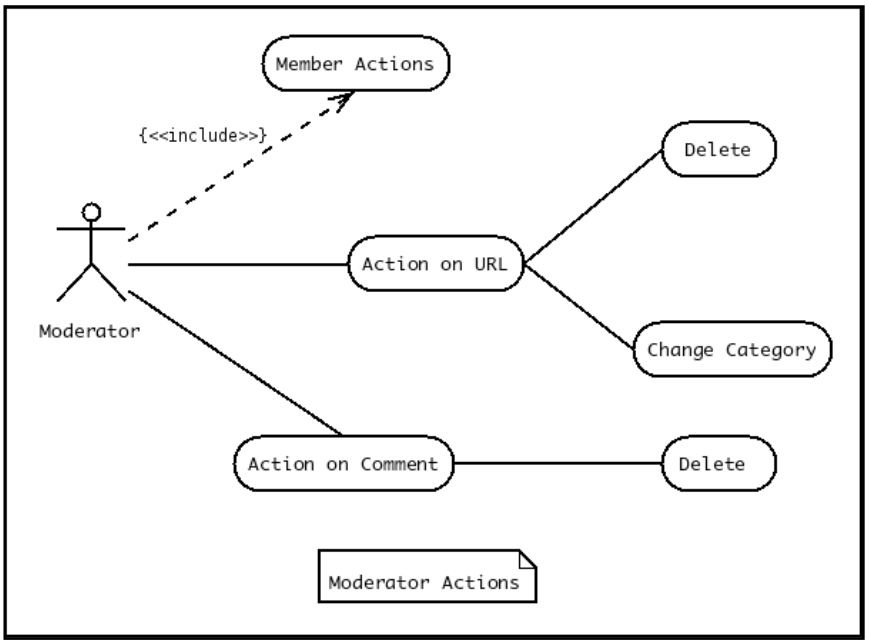
Dexter

Making search web 2.0 ready!

MEMBER ACTIONS (FEATURE 1)



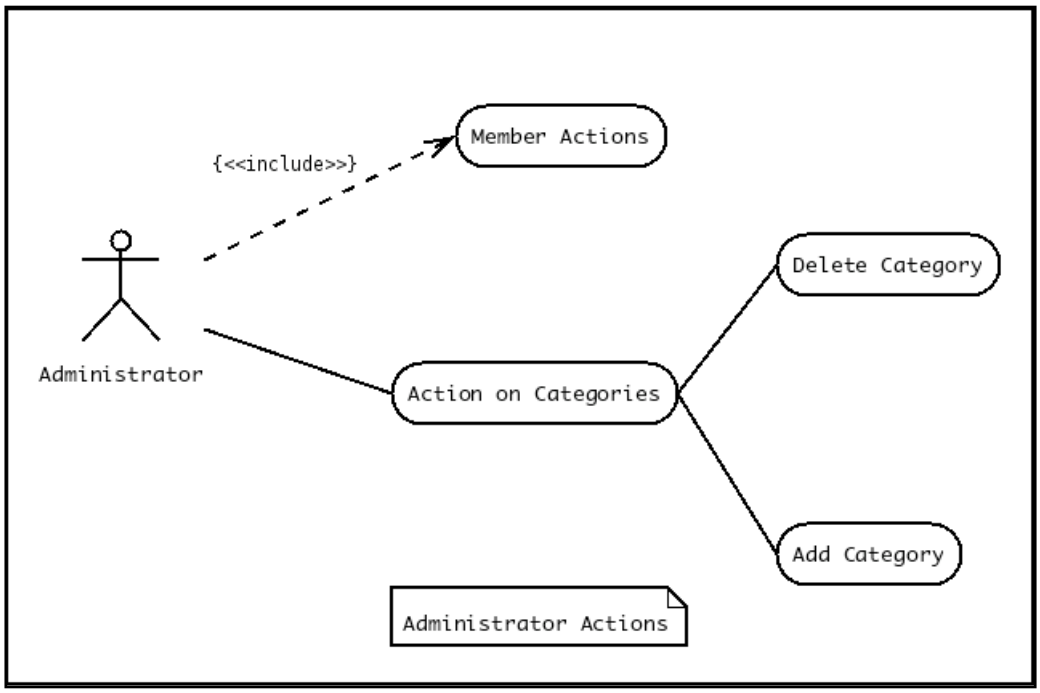
MODERATOR ACTIONS (FEATURE 2)



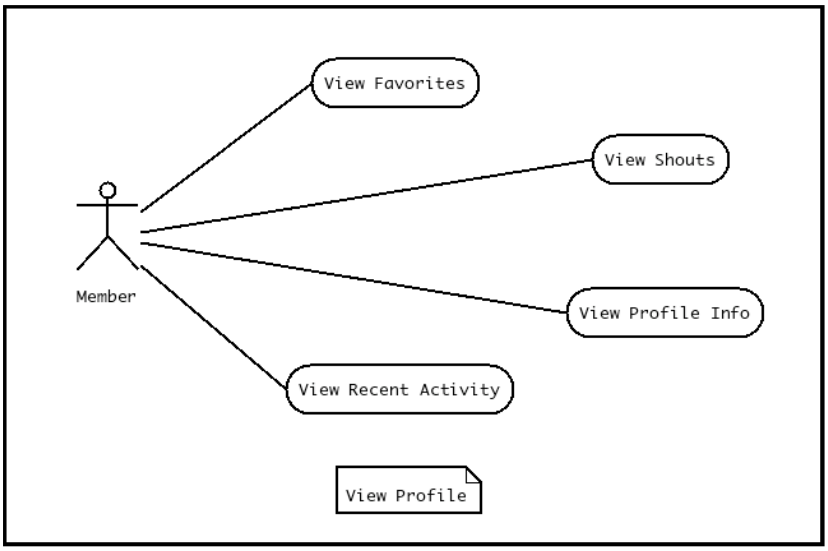
Dexter

Making search web 2.0 ready!

ADMINISTRATOR ACTIONS (FEATURE 3)



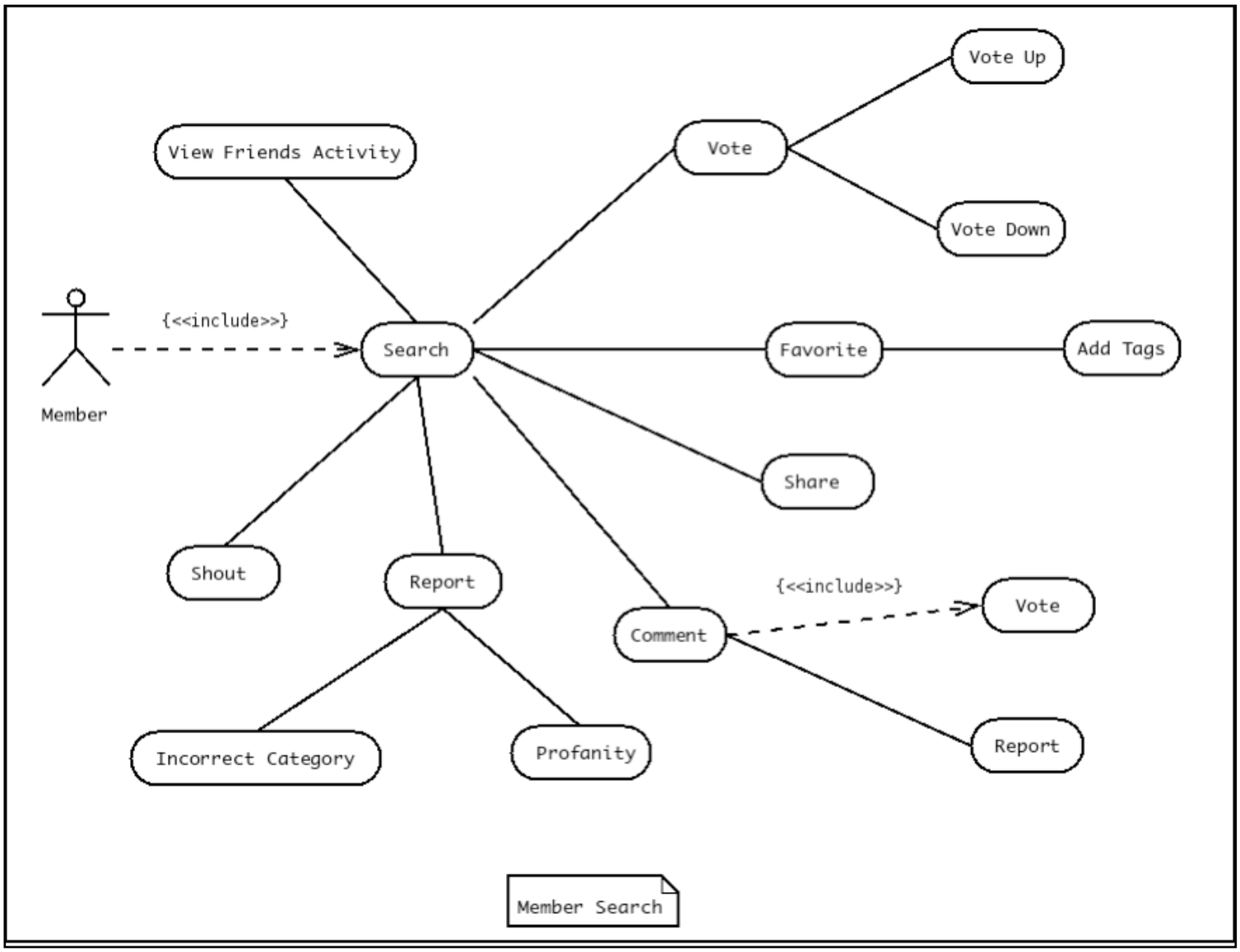
VIEW PROFILE (FEATURE 4)



Dexter

Making search web 2.0 ready!

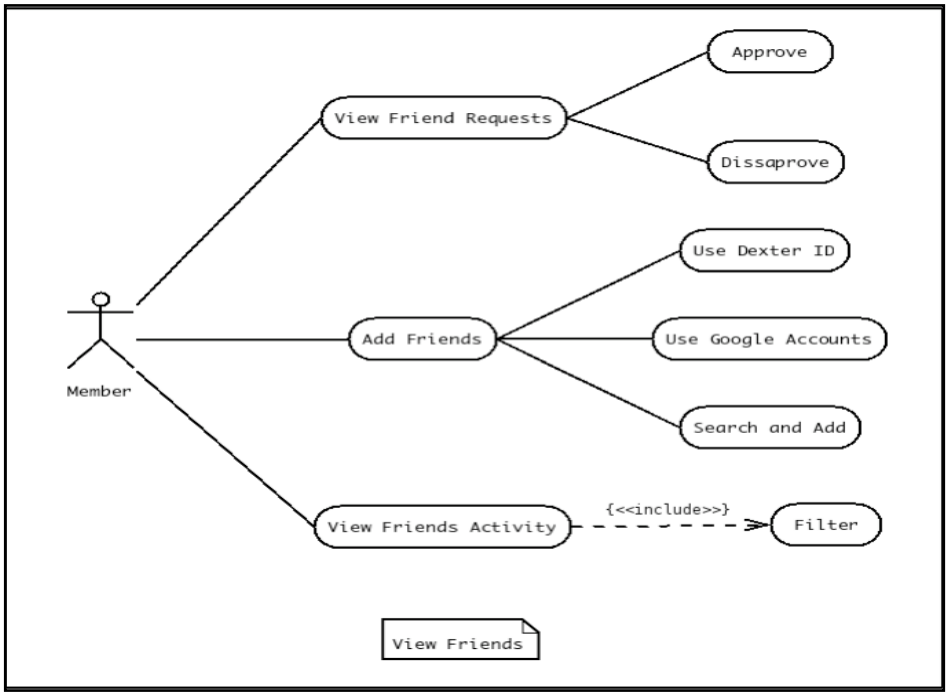
MEMBER SEARCH (FEATURE 5)



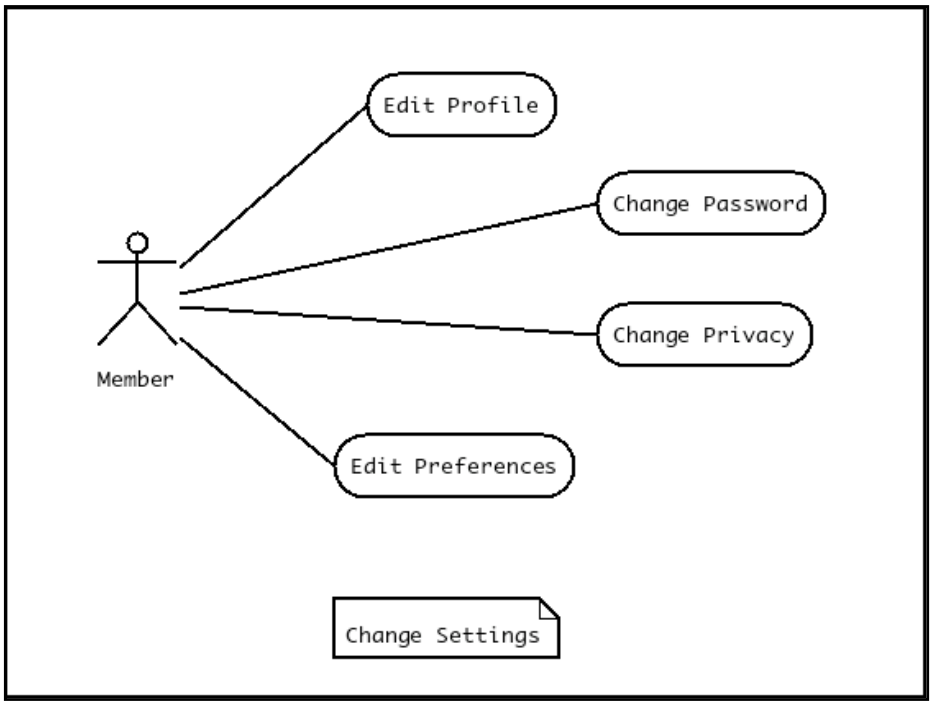
Dexter

Making search web 2.0 ready!

□ VIEW FRIENDS (FEATURE 6)



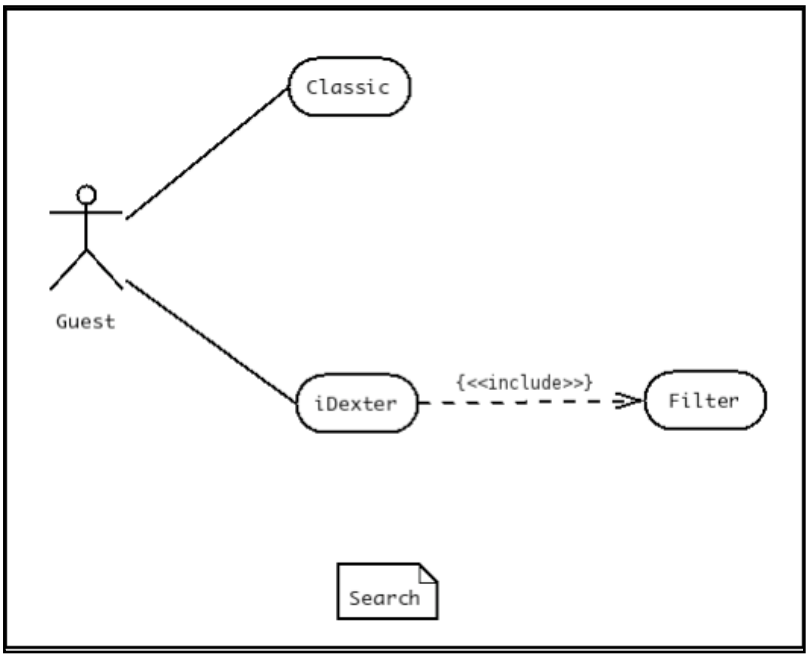
□ SETTINGS (FEATURE 7)



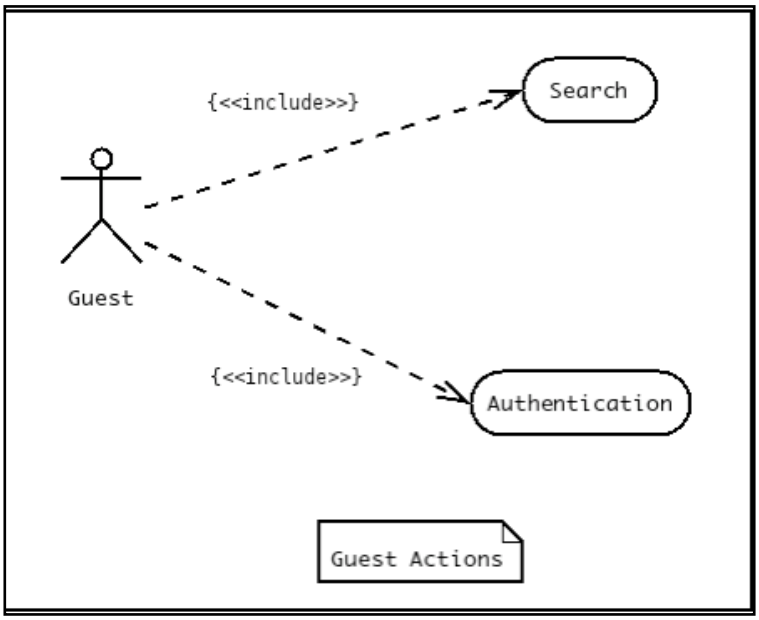
Dexter

Making search web 2.0 ready!

□ SEARCH (FEATURE 8)



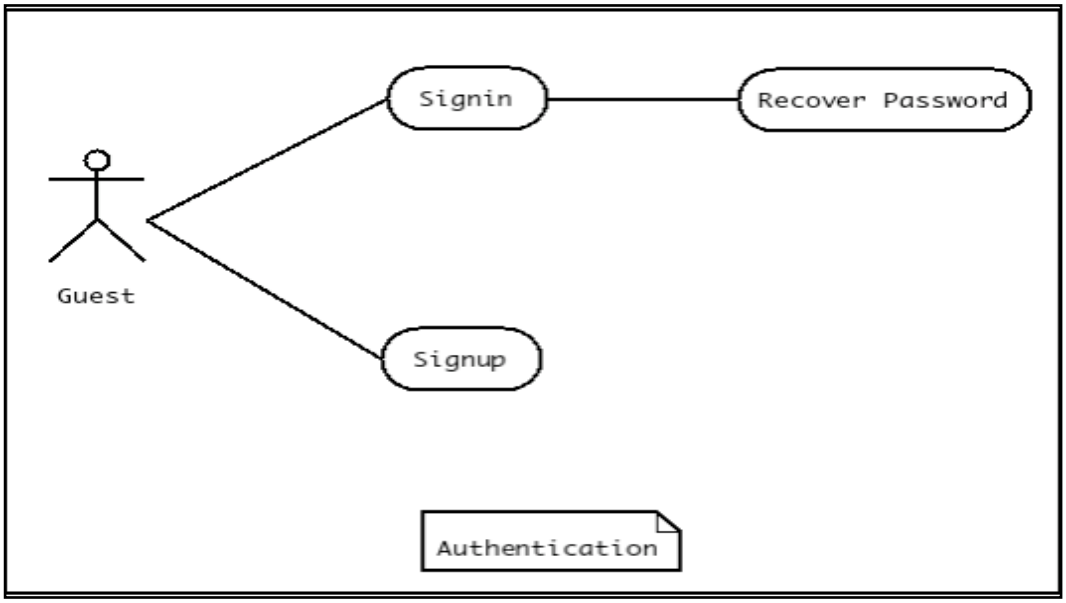
□ GUEST ACTIONS (FEATURE 9)



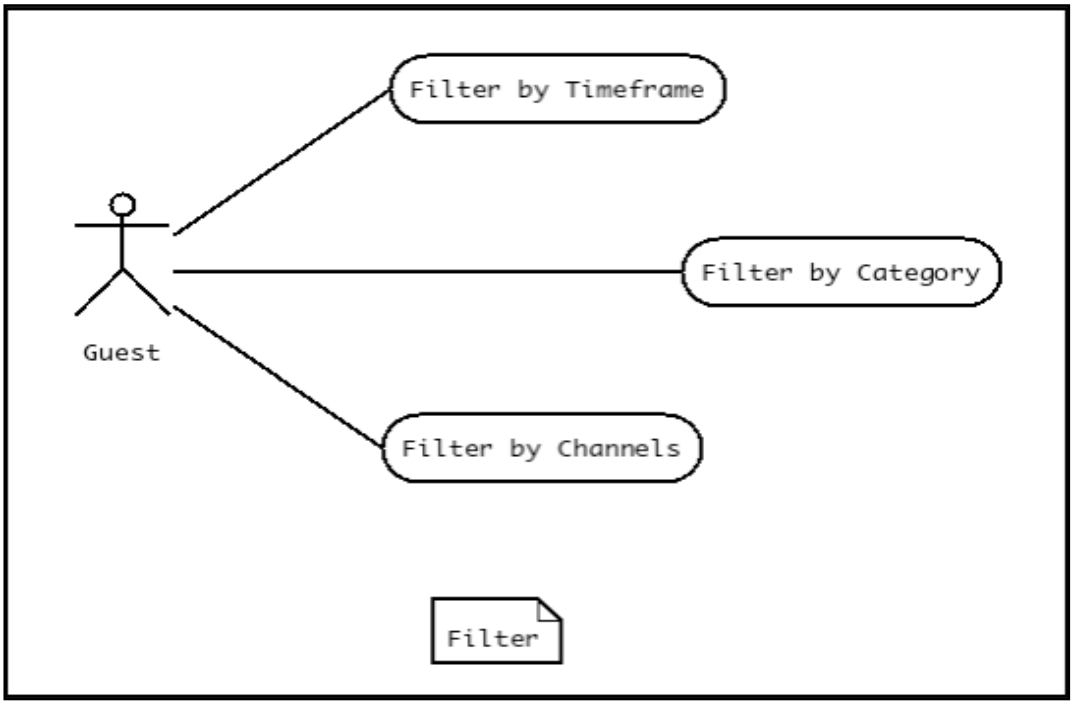
Dexter

Making search web 2.0 ready!

AUTHENTICATION (FEATURE 10)



FILTER (FEATURE 11)



Dexter

Making search web 2.0 ready!

OTHER NONFUNCTIONAL REQUIREMENTS

PERFORMANCE REQUIREMENTS

- The Google search result retrieval and display should be as fast as possible.

SAFETY REQUIREMENTS

- The search pattern as being logged by the web-application would reveal the user-characteristics and one's activity over the web.

SECURITY REQUIREMENTS

- The registration information should traverse the cloud in an encrypted form to prevent eavesdropping (need to use SSL)

- User needs to have options to set the privacy level for registration information and its activities in the form of comments or votes.

SOFTWARE QUALITY ATTRIBUTES

Adaptability – as Dexter directly uses the Google search result API, it would automatically adapt to the changing order of search results which have not yet been submitted.

Flexibility – the search results are flexible as they are user ordered.

Maintainability – Moderators/Administrators have the work to maintain the content in the Dexter database

Reusability – Dexter being open-source could be used by other projects who would wish to add to its functionality.

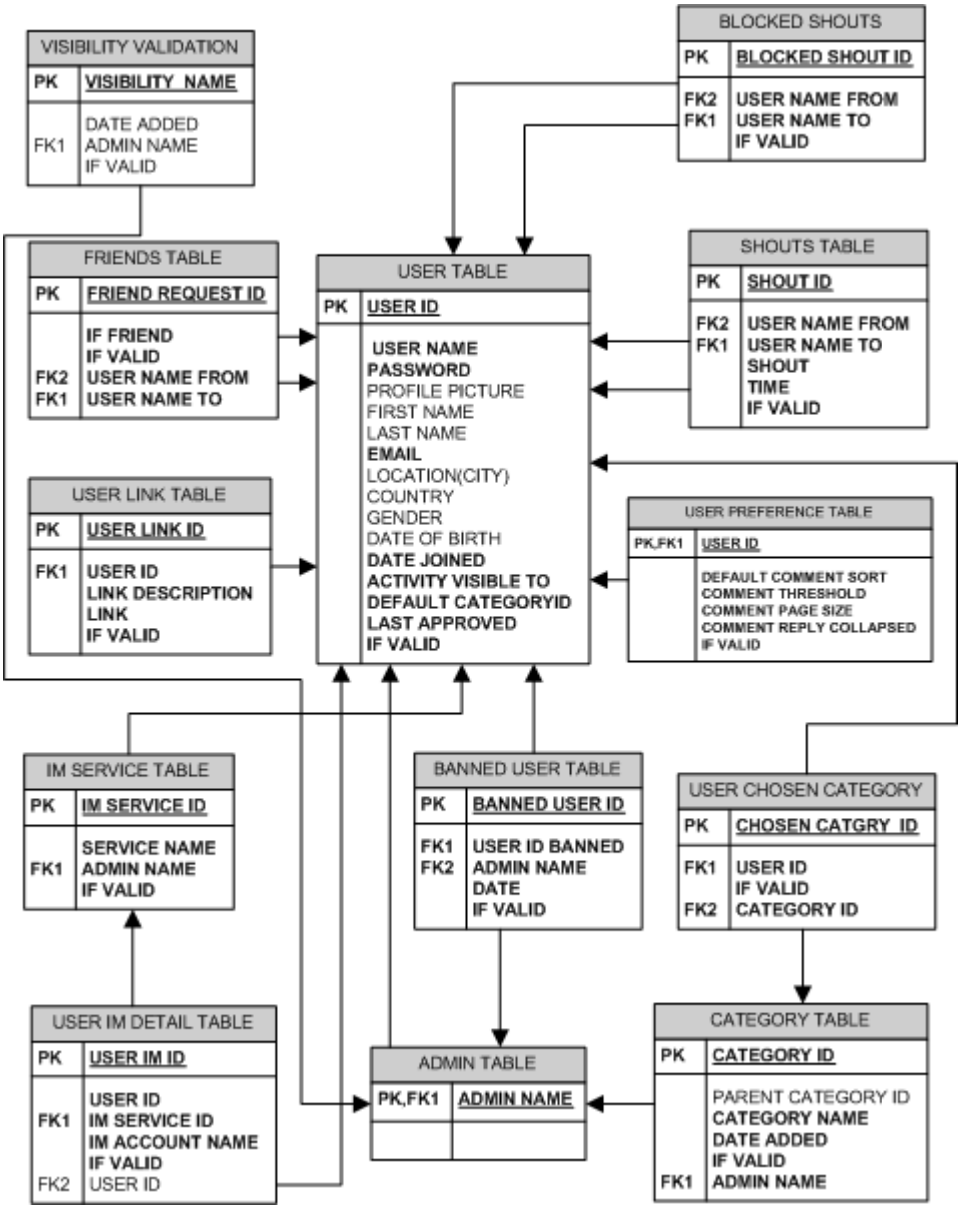
Usability – It's even usable in cases when user does not know what to search for, and can use the iDexter interface to get started.

Dexter

Making search web 2.0 ready!

ENTITY RELATIONSHIP DIAGRAM

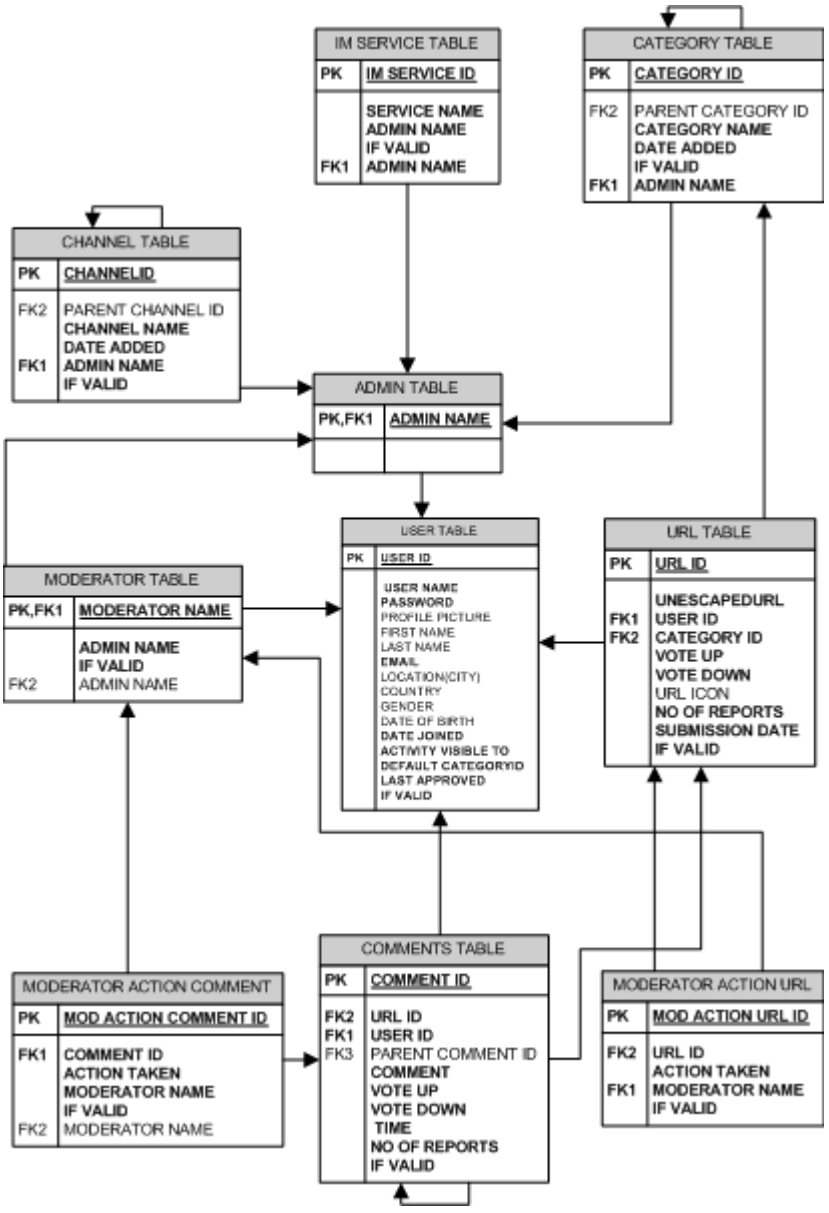
USER MODULE



Dexter

Making search web 2.0 ready!

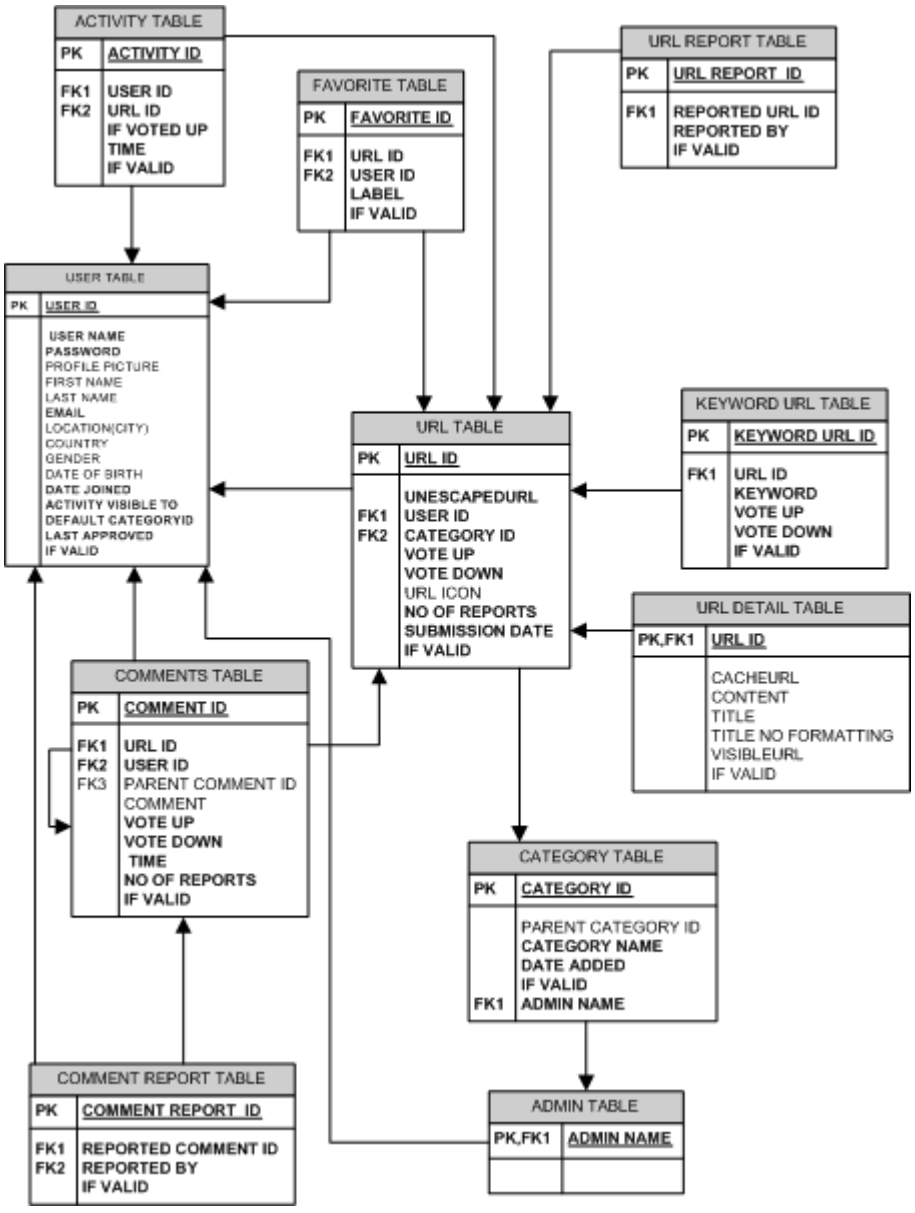
ADMIN MODULE



Dexter

Making search web 2.0 ready!

URL MODULE

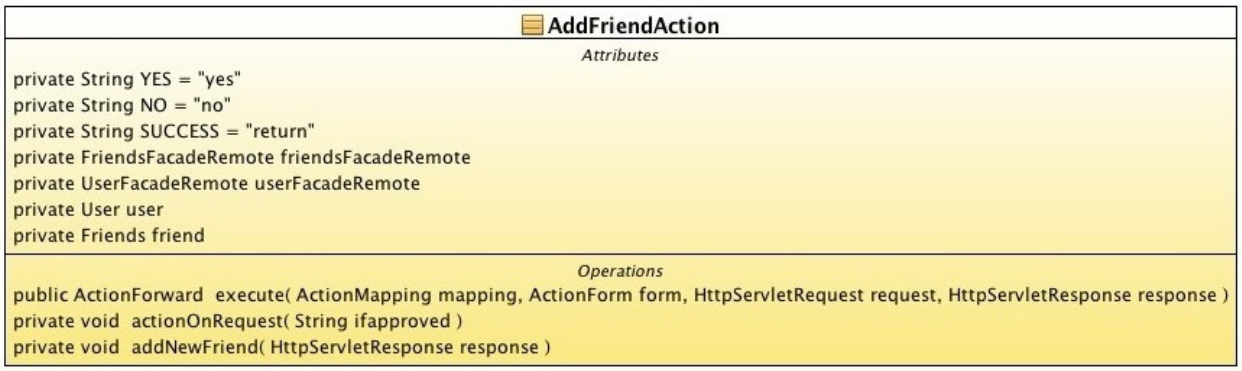


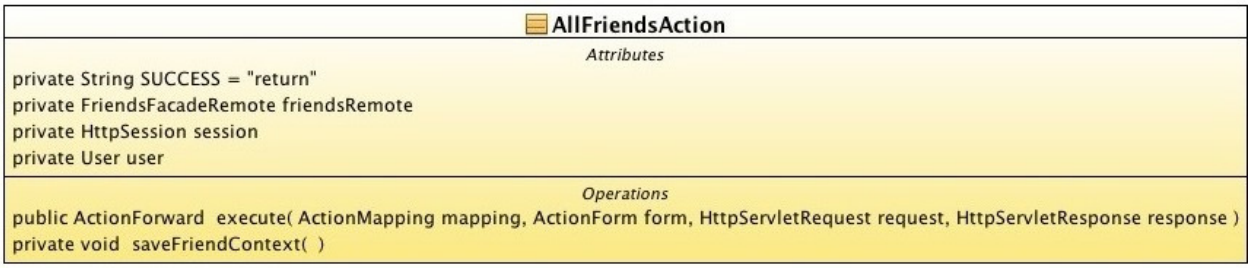
Dexter

Making search web 2.0 ready!

CLASS DIAGRAM

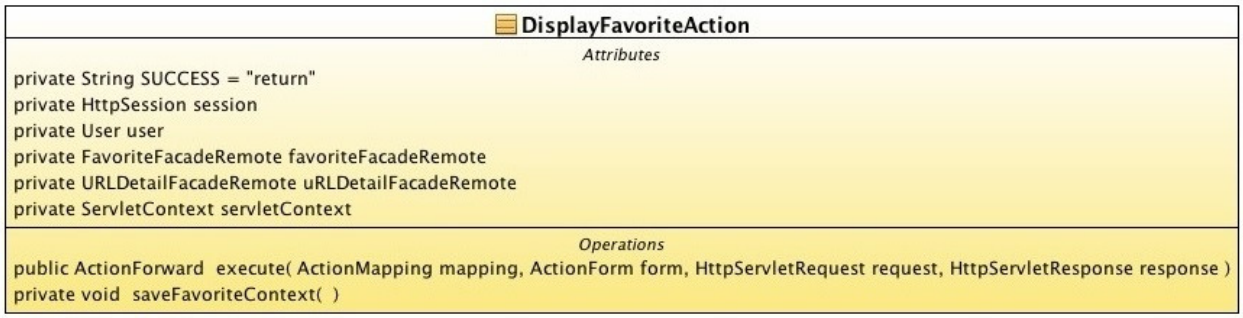
ACTION PACKAGE

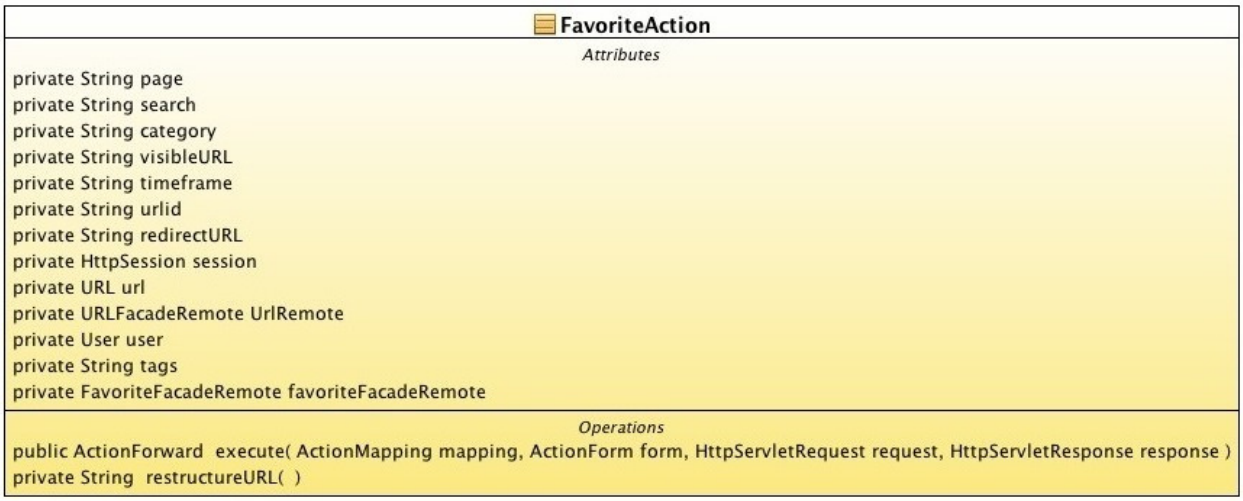




Dexter

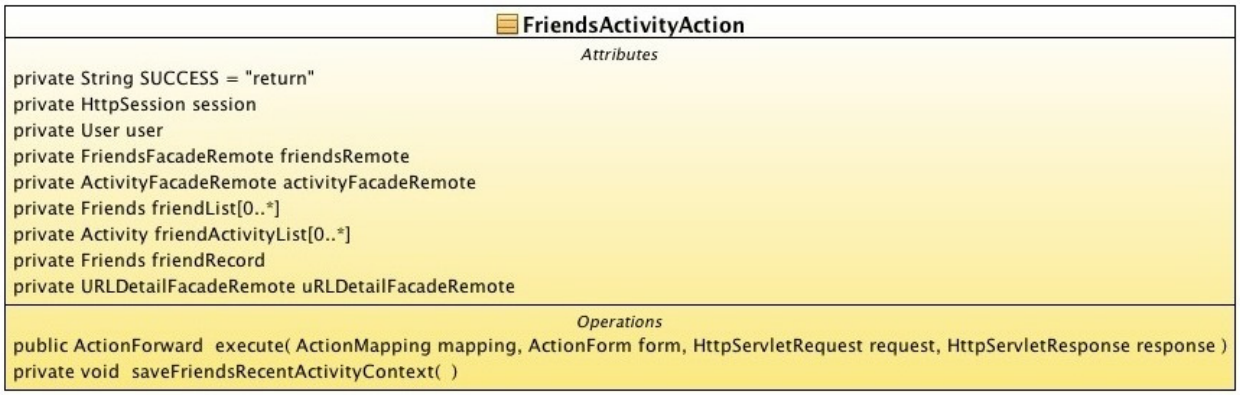
Making search web 2.0 ready!

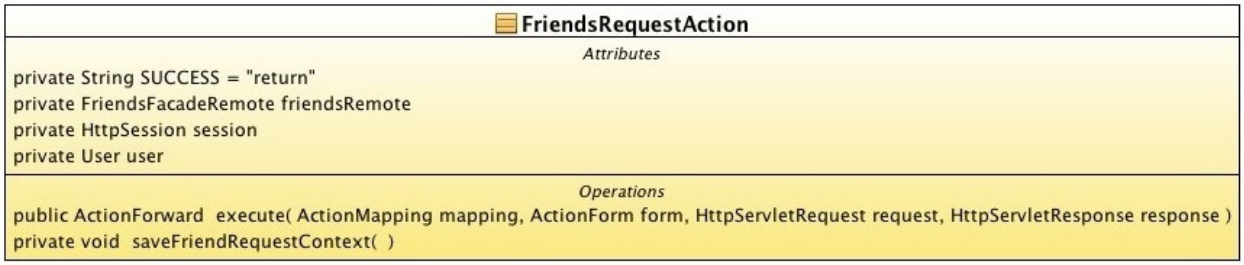


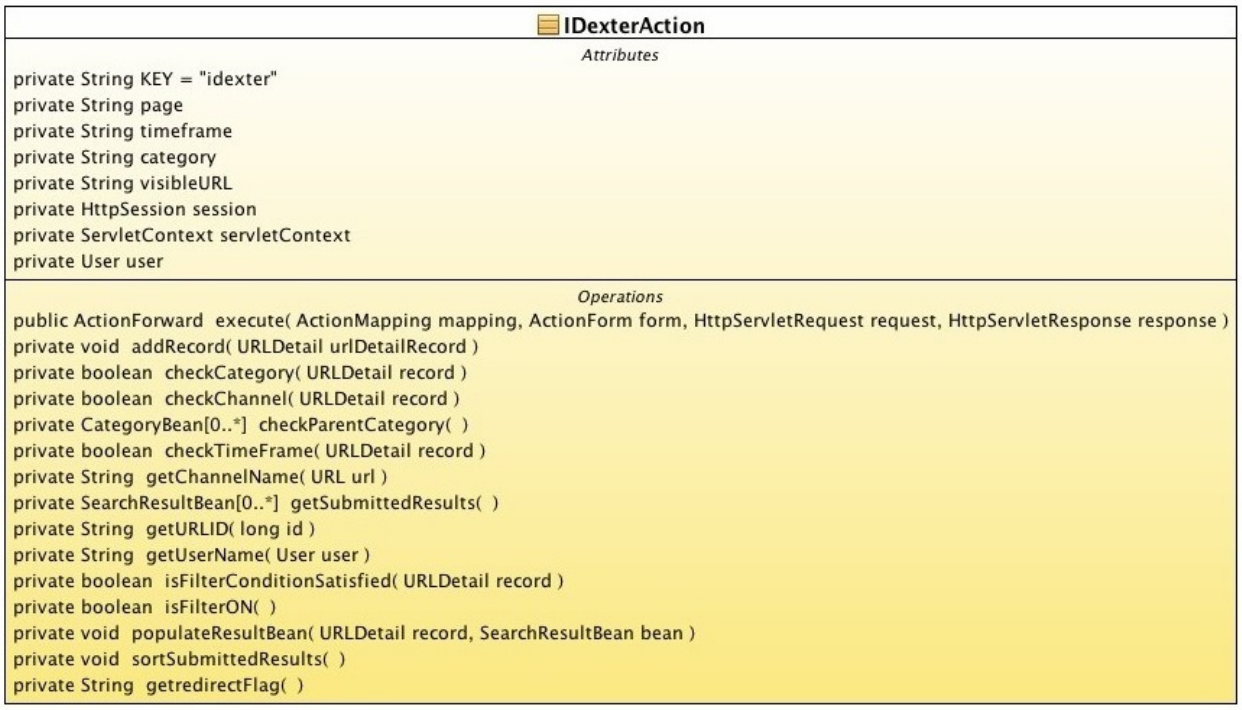


Dexter

Making search web 2.0 ready!

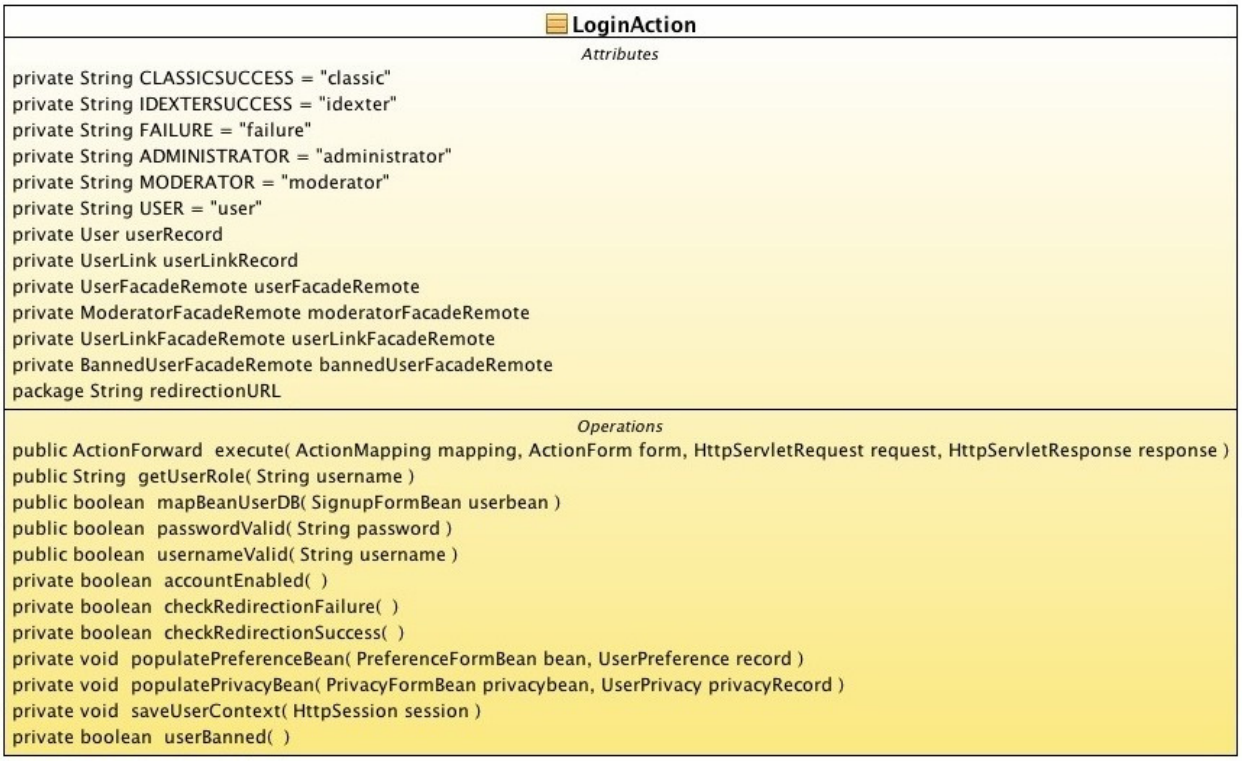


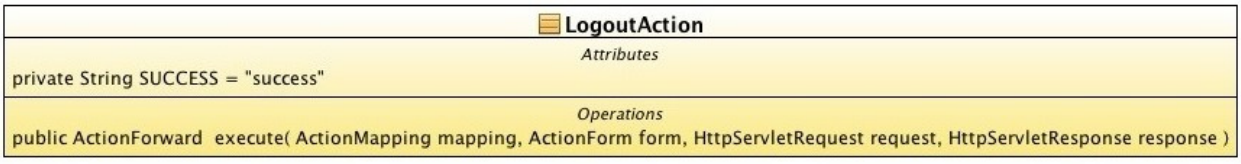




Dexter

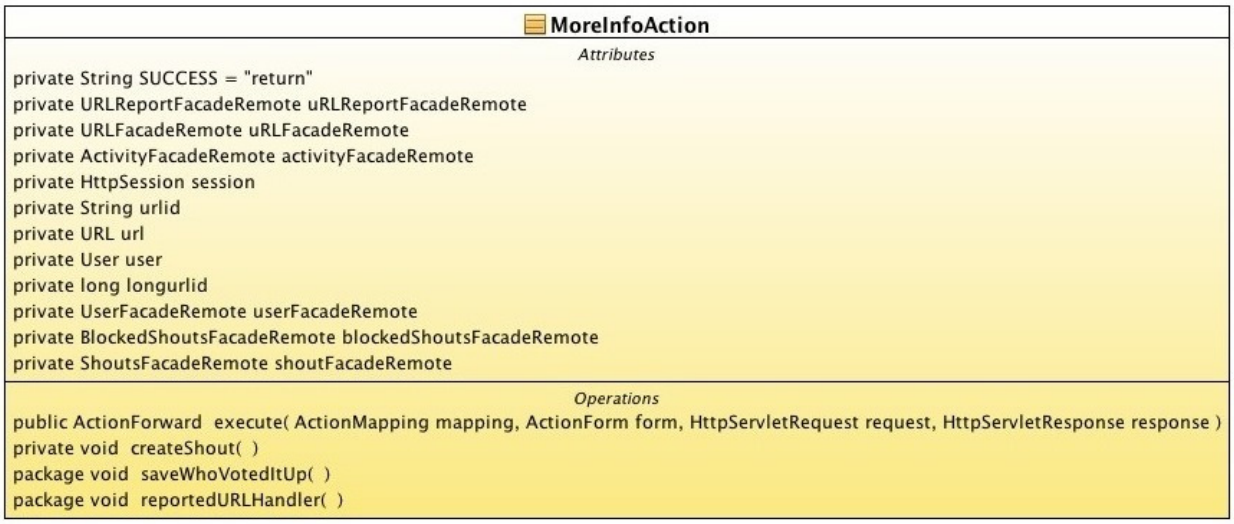
Making search web 2.0 ready!

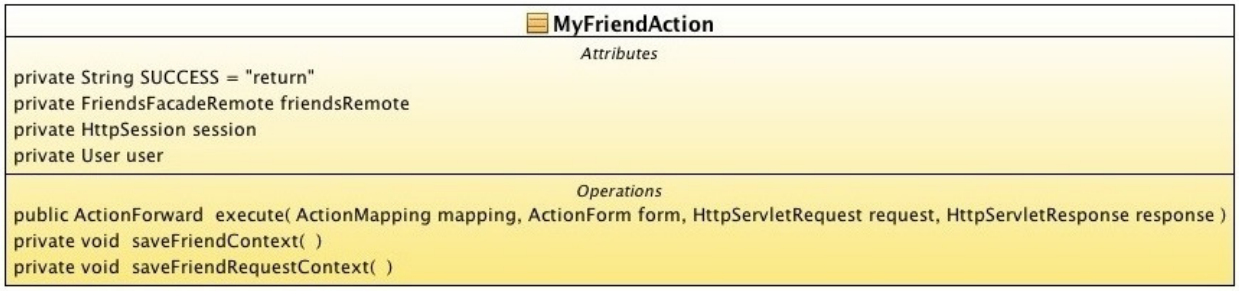


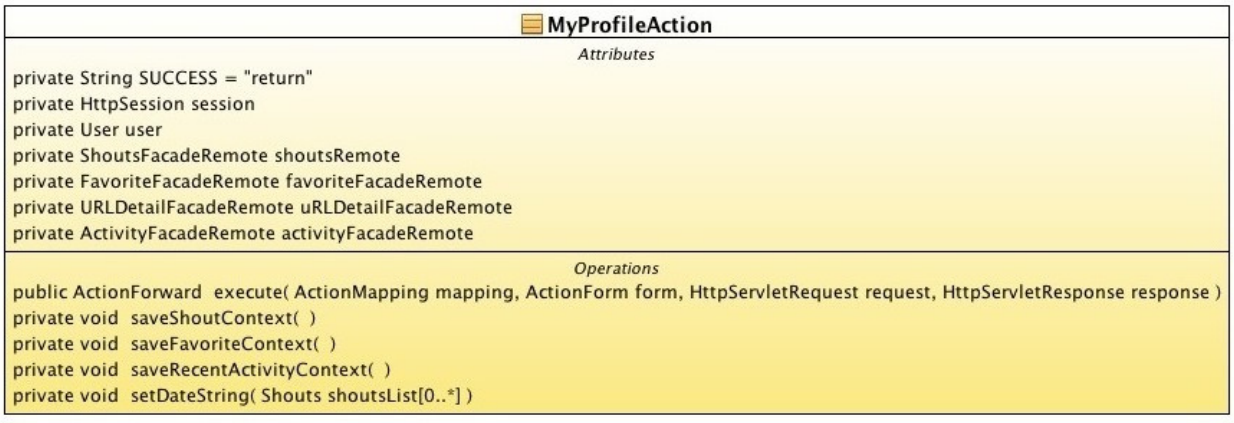


Dexter

Making search web 2.0 ready!

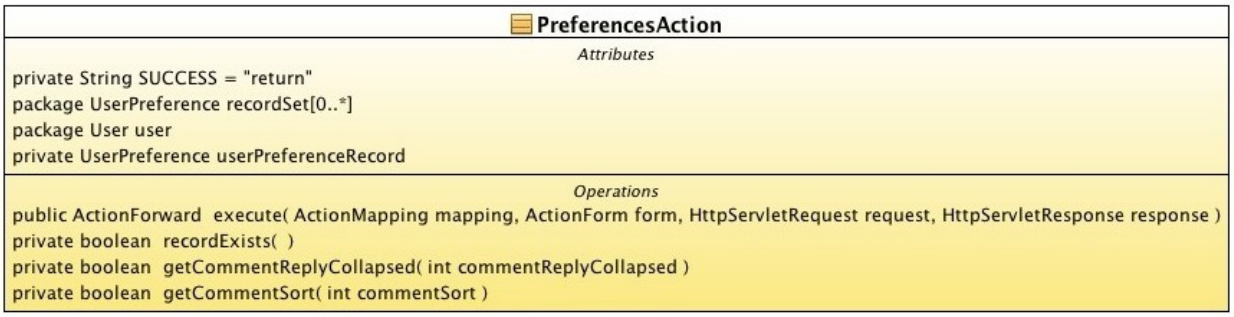


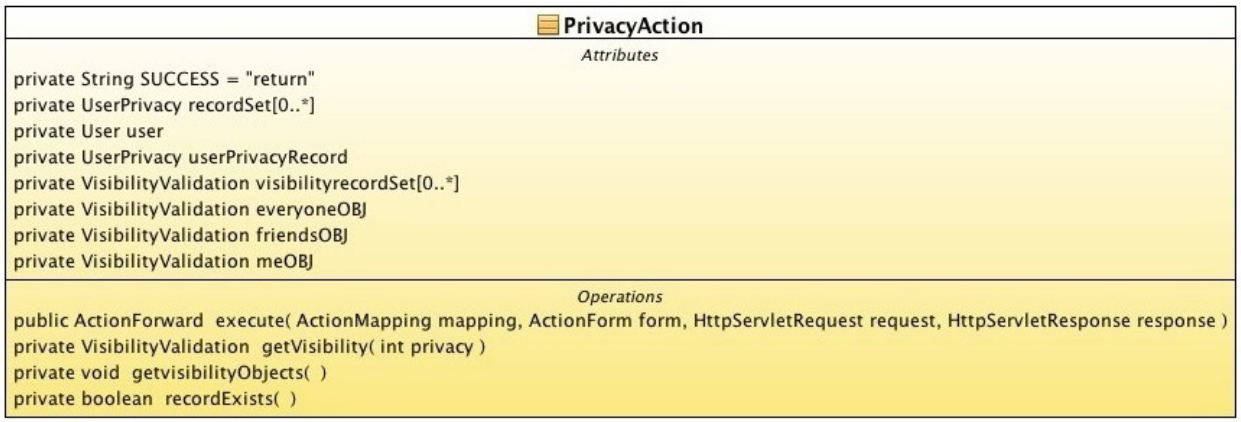


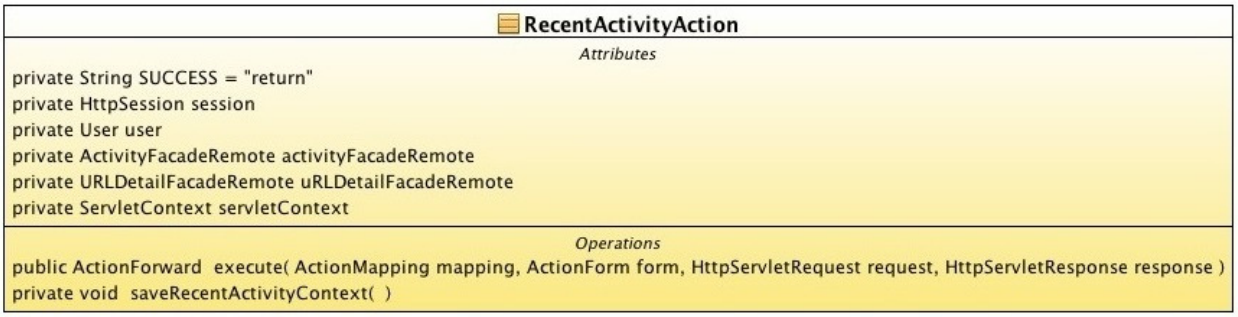


Dexter

Making search web 2.0 ready!

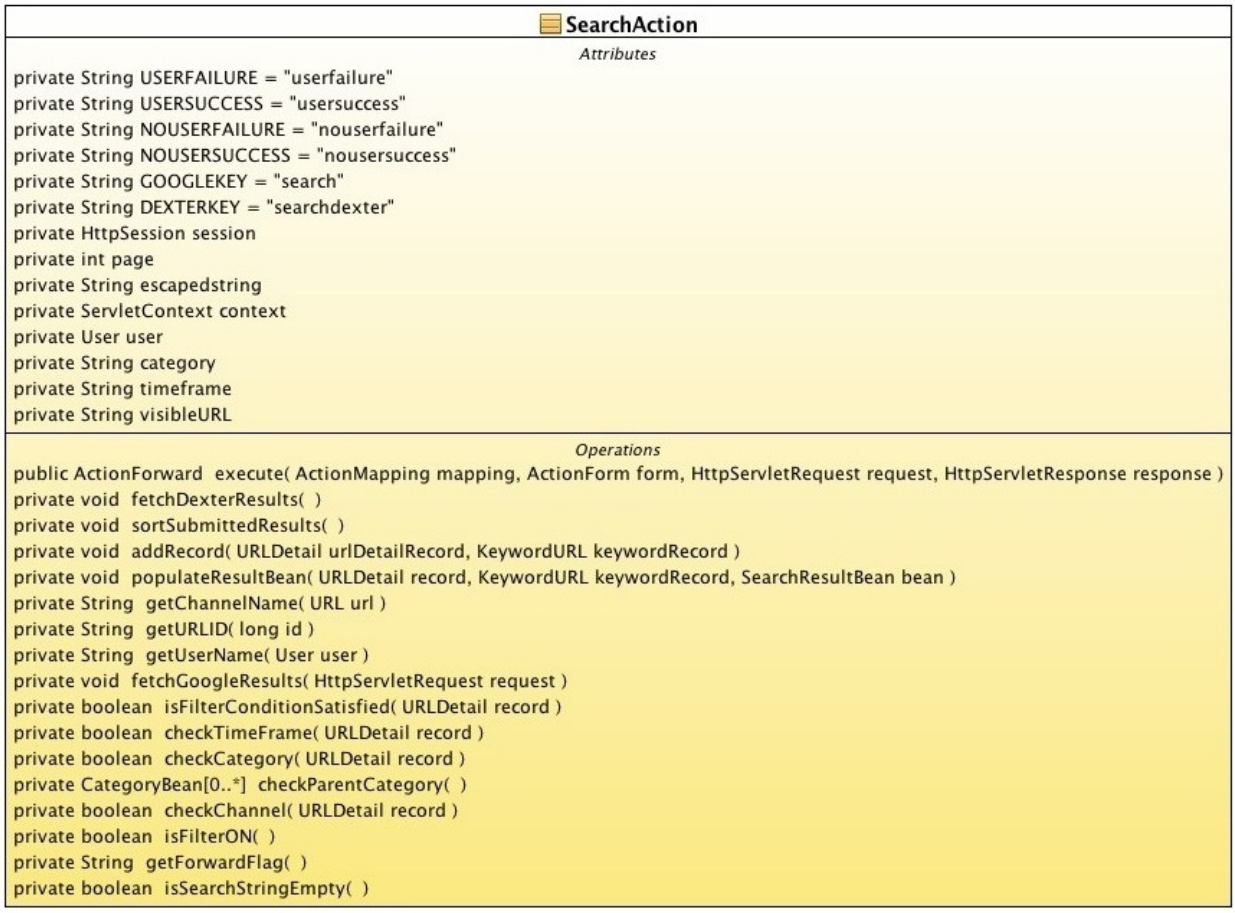






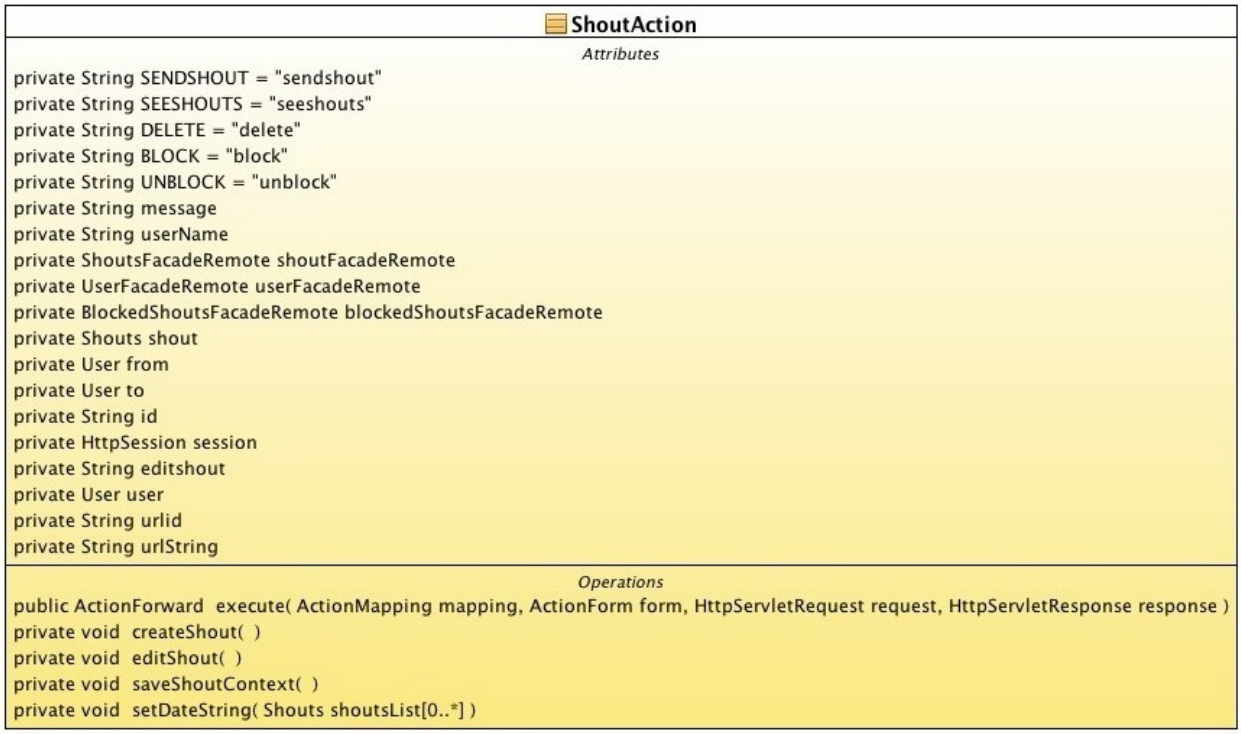
Dexter

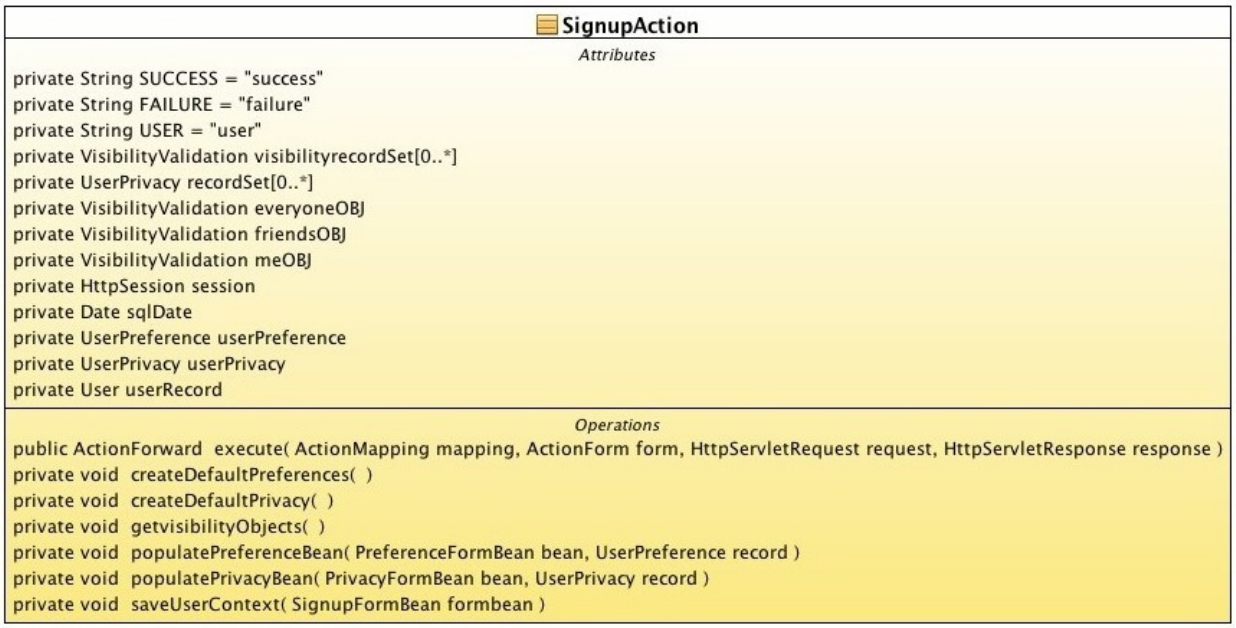
Making search web 2.0 ready!



Dexter

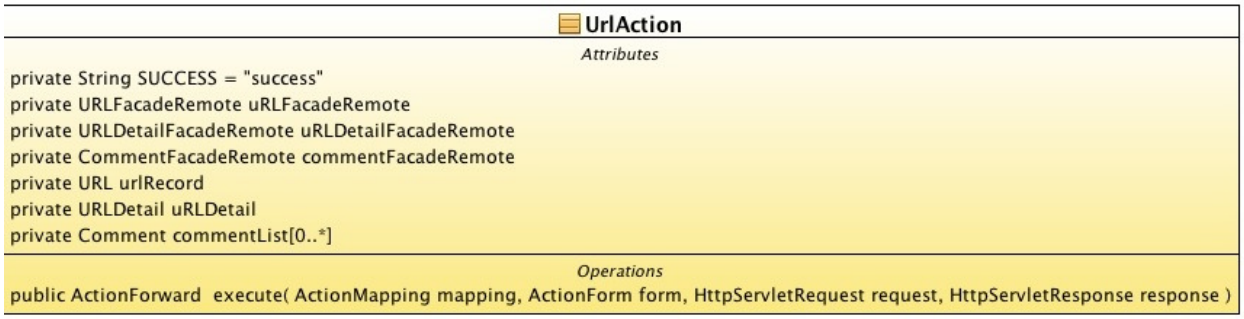
Making search web 2.0 ready!





Dexter

Making search web 2.0 ready!



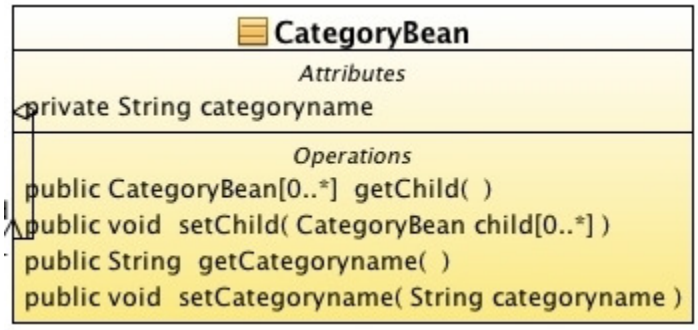
Dexter

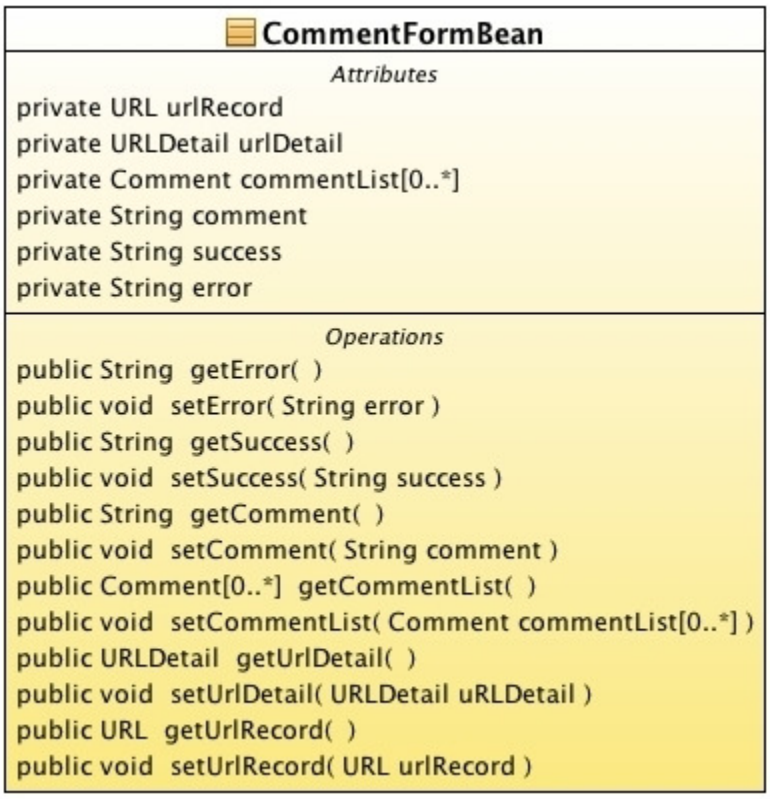
Making search web 2.0 ready!

BEANS PACKAGE

Dexter

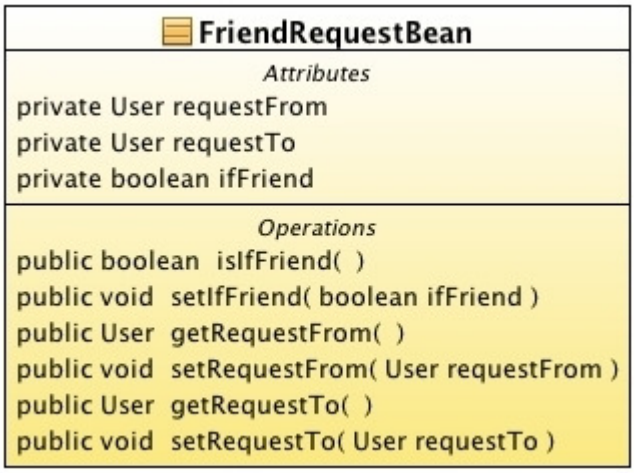
Making search web 2.0 ready!

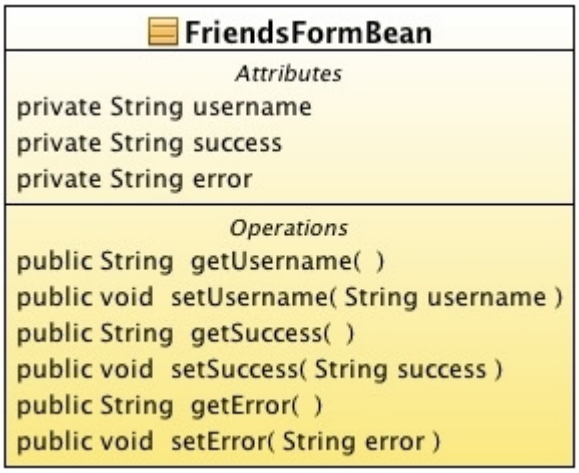




Dexter

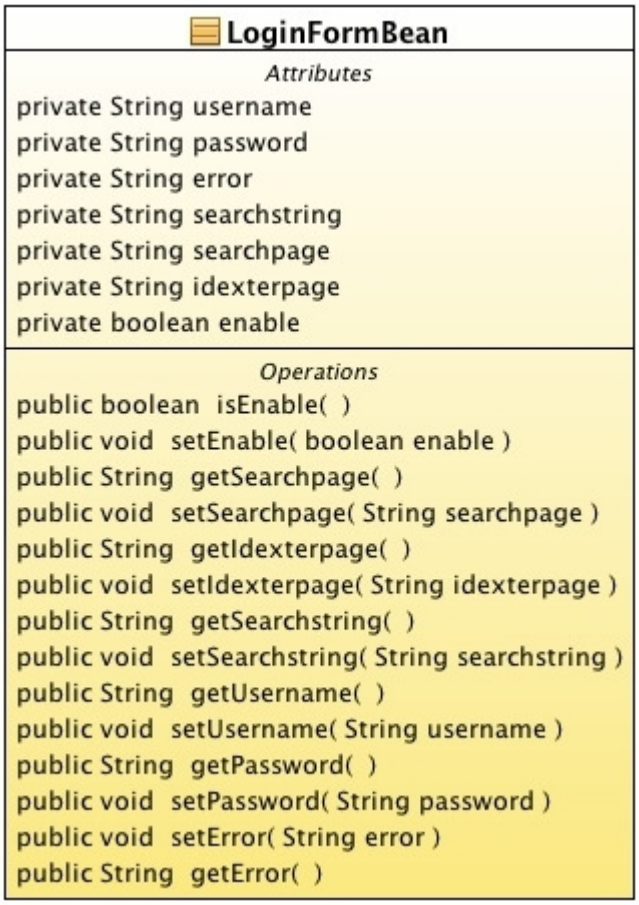
Making search web 2.0 ready!





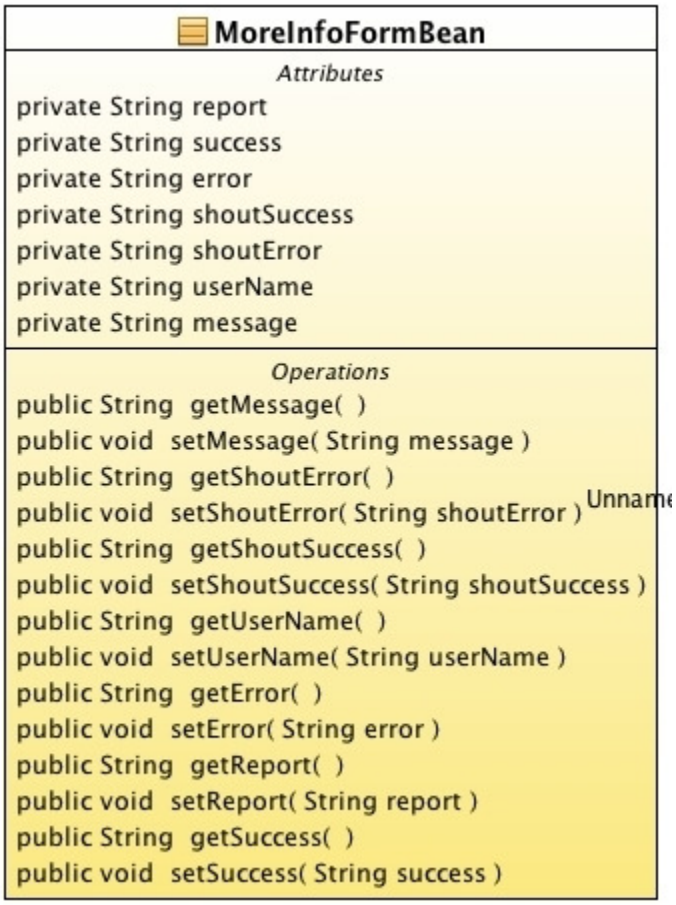
Dexter

Making search web 2.0 ready!



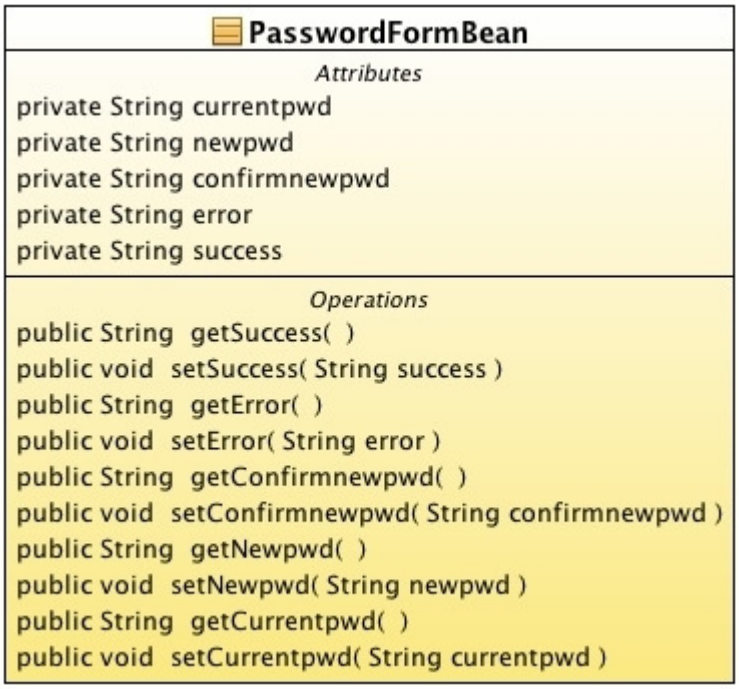
Dexter

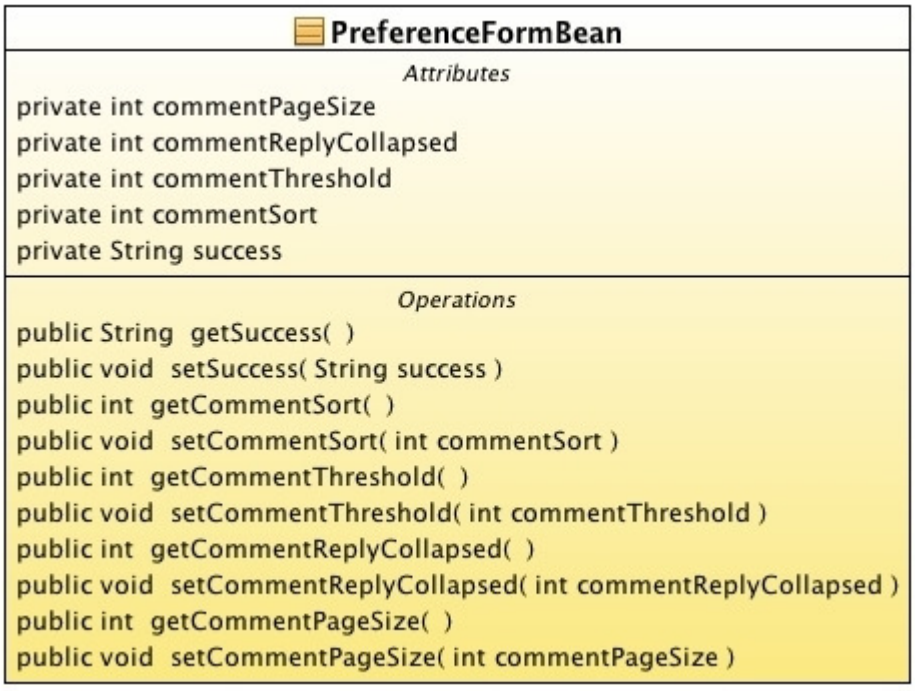
Making search web 2.0 ready!



Dexter

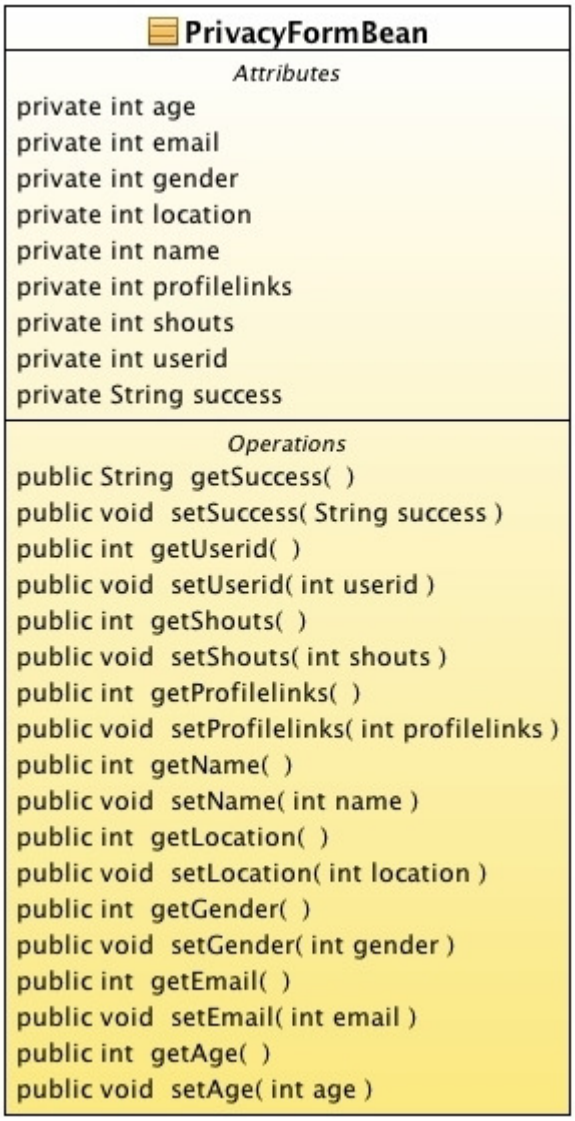
Making search web 2.0 ready!





Dexter

Making search web 2.0 ready!

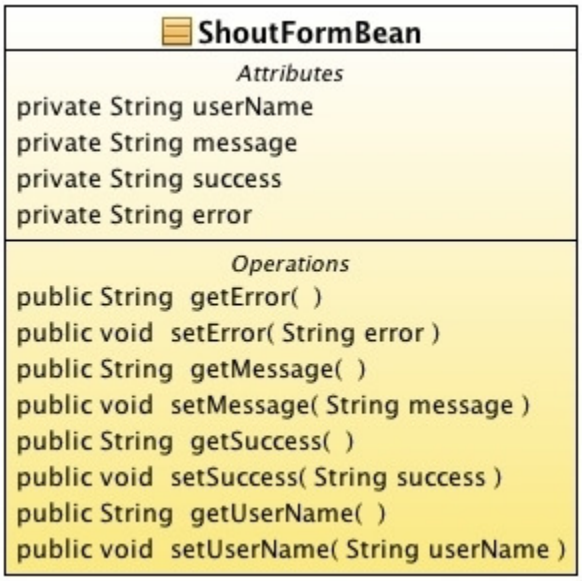


Dexter

Making search web 2.0 ready!

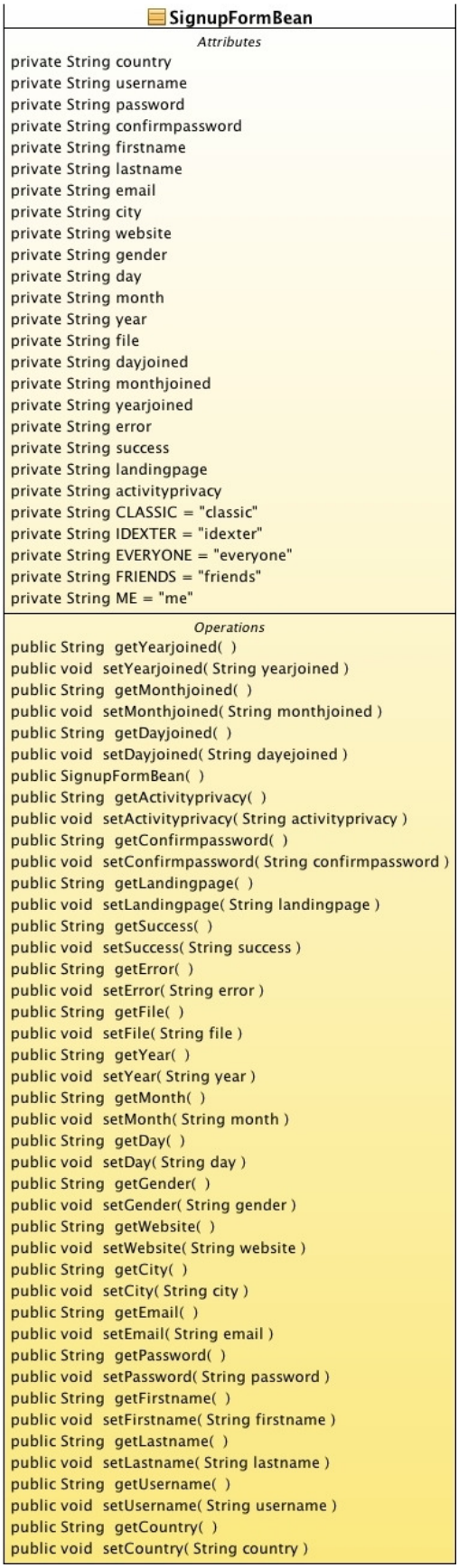
Dexter

Making search web 2.0 ready!



Dexter

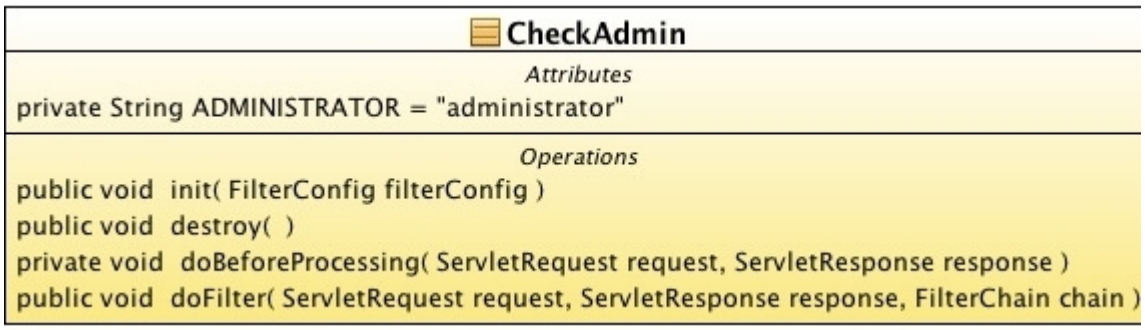
Making search web 2.0 ready!

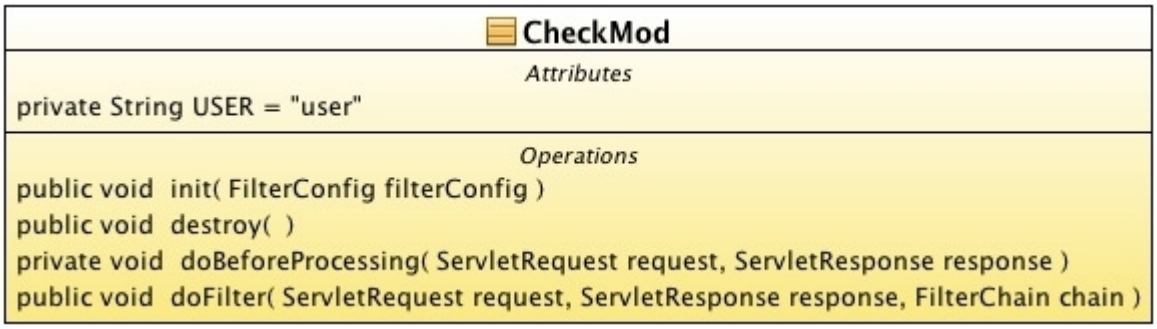


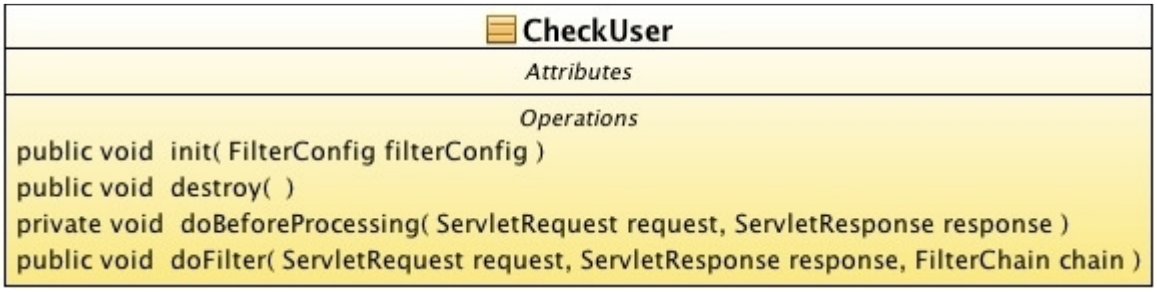
Dexter

Making search web 2.0 ready!

FILTERS PACKAGE

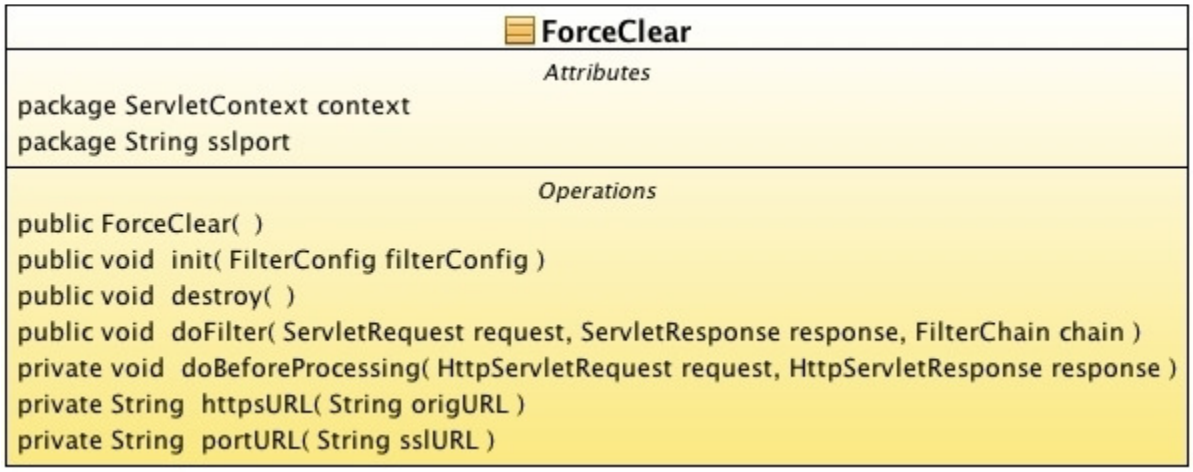


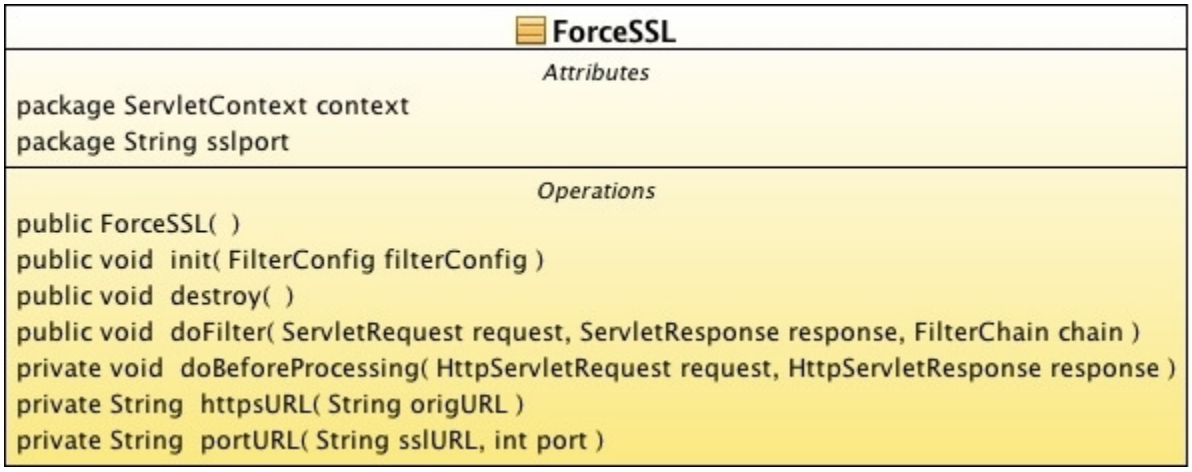




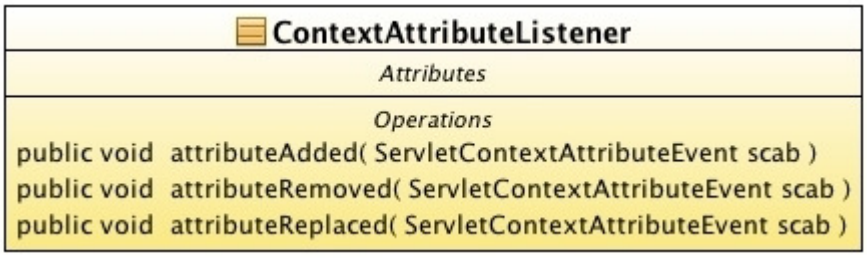
Dexter

Making search web 2.0 ready!



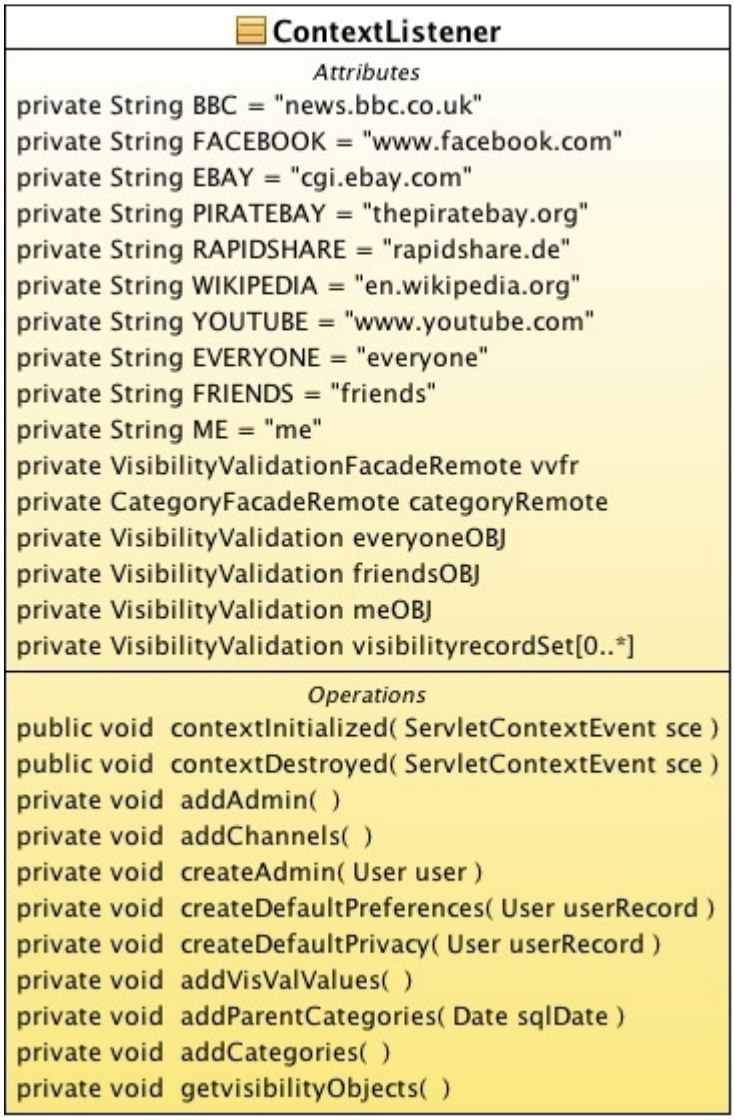


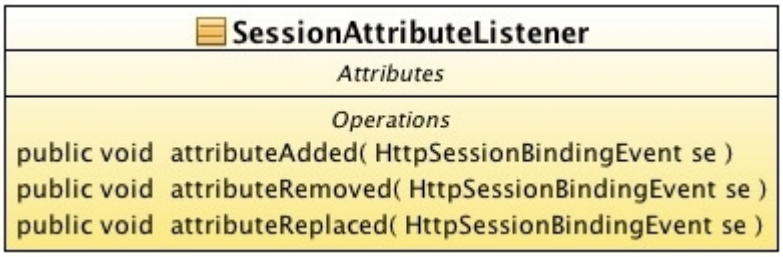
LISTENER PACKAGE



Dexter

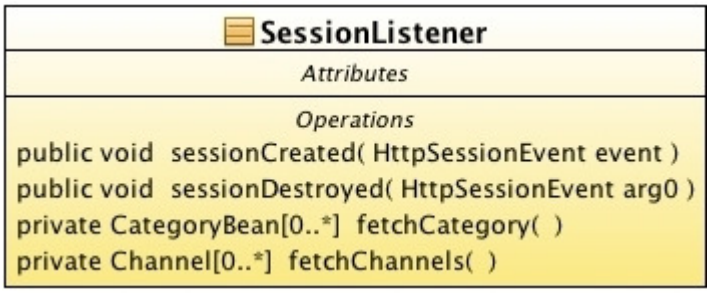
Making search web 2.0 ready!



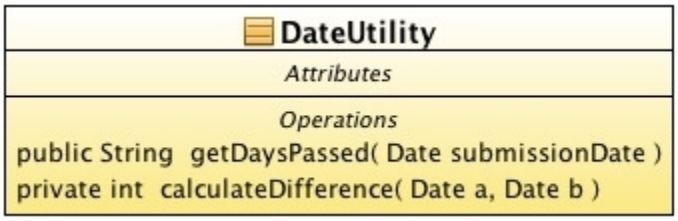


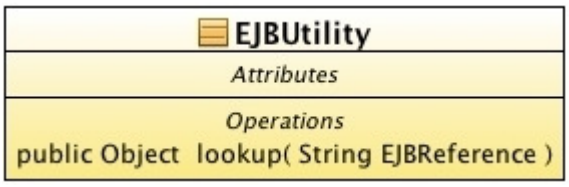
Dexter

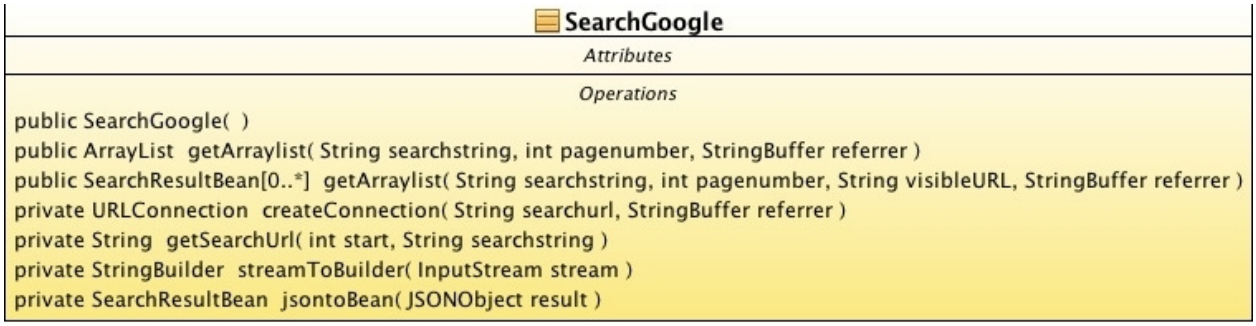
Making search web 2.0 ready!



UTILITIES PACKAGE

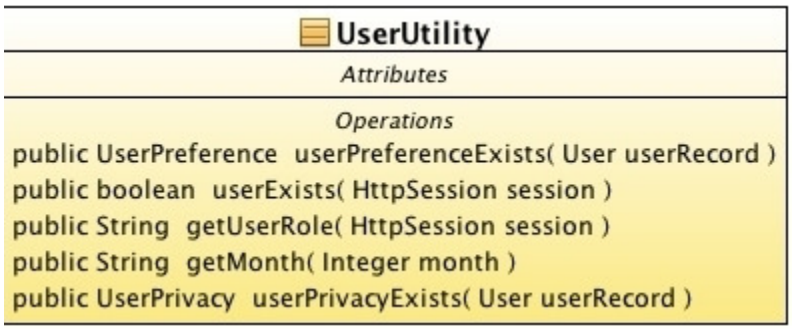






Dexter

Making search web 2.0 ready!



Dexter

Making search web 2.0 ready!

IMPLEMENTATION

CODE SNIPPET

SearchGoogle.java

```
package utilities;


import beans.SearchResultBean;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.UUID;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;


public class SearchGoogle {


    ArrayList<SearchResultBean> resultBeanArray;
```

Dexter

Making search web 2.0 ready!

```
    public SearchGoogle() {
        resultBeanArray = new ArrayList();
    }

    public ArrayList getArraylist(String searchstring, int pagenumber, StringBuffer referrer)
throws IOException, JSONException {

        /*initialize start*/
        //int pagenumber = Integer.parseInt(page);
        int start = (pagenumber – 1) * 8;

        /* Loop twice to get 2 JSON Objects
         * each JSON Object contains 4 search results
         * each page would return 8 search results
         */
        for (int I = 1; I <= 2; i++, start += 4) {

            /*initialize search url*/
            String searchurl = getSearchUrl(start, searchstring);

            /* create a socket */
            URLConnection connection = createConnection(searchurl, referrer);

            /* get the response */
            InputStream stream = connection.getInputStream();
```

```
StringBuilder builder = streamToBuilder(stream);

/* Wrap the response in JSONObject */
JSONObject json = new JSONObject(builder.toString());

/* Get StatusCode sent by Google */
int statusCode = json.getInt("responseStatus");

if (statusCode == 200) {   // status(200) is OK

   /* Fetch responsedata */
   JSONObject responseData = json.getJSONObject("responseData");

   /* Ferch results */
   JSONArray results = responseData.getJSONArray("results");

   /* Iterate the results JSONArray */
   for (int j = 0; j < results.length(); j++) {

      /* Fetch a search result */
      JSONObject result = results.getJSONObject(j);

      /* Push search result details to searchresultbean */
      SearchResultBean resultbean = jsontoBean(result);

      /* Add the bean to ArrayList */
```

```
                resultBeanArray.add(resultbean);
            }
        }
    }
    return (resultBeanArray);
}

    public ArrayList<SearchResultBean> getArraylist(String searchstring, int pagenumber,
String visibleURL, StringBuffer referrer) throws IOException, JSONException {

    do {

        /*initialize start*/
        //int pagenumber = Integer.parseInt(page);
        int start = (pagenumber – 1) * 8;

        /* Loop twice to get 2 JSON Objects
         * each JSON Object contains 4 search results
         * each page would return 8 search results
         */
        for (int I = 1; I <= 2; i++, start += 4) {

            /*initialize search url*/
            String searchurl = getSearchUrl(start, searchstring);

            /* create a socket */
```

Dexter

Making search web 2.0 ready!

```
URLConnection connection = createConnection(searchurl, referrer);

/* get the response */
InputStream stream = connection.getInputStream();
StringBuilder builder = streamToBuilder(stream);

/* Wrap the response in JSONObject */
JSONObject json = new JSONObject(builder.toString());

/* Get StatusCode sent by Google */
int statusCode = json.getInt("responseStatus");

if (statusCode == 200) {   // status(200) is OK

   /* Fetch responsedata */
   JSONObject responseData = json.getJSONObject("responseData");

   /* Ferch results */
   JSONArray results = responseData.getJSONArray("results");

   /* Iterate the results JSONArray */
   for (int j = 0; j < results.length(); j++) {

      /* Fetch a search result */
      JSONObject result = results.getJSONObject(j);
```

```
                /* Push search result details to searchresultbean */
                SearchResultBean resultbean = jsontoBean(result);

                /* Check if Channel Matches */
                if (resultbean.getVisibleUrl().equalsIgnoreCase(visibleURL)) {

                   /* Add the bean to ArrayList */
                   resultBeanArray.add(resultbean);
                }
             }
          }
       }
       pagenumber += 1;
    } while (pagenumber < 8 && resultBeanArray.size() < 8);

    return (resultBeanArray);
  }

  /* create a socket */
  private URLConnection createConnection(String searchurl, StringBuffer referrer) throws
IOException, MalformedURLException {

    URL sendurl = new URL(searchurl);
    URLConnection connection = sendurl.openConnection();
    connection.addRequestProperty("Referer", referrer.toString());
    return connection;
```

```
}

/*initialize search url*/
private String getSearchUrl(int start, String searchstring) {

    String startparam = "&start=" + start;
    String searchparam = "&q=" + searchstring;
    String googleapipath = "http://ajax.googleapis.com/ajax/services/search/web?v=1.0";

    String url = googleapipath + startparam + searchparam;
    return (url);
}

/* wrap stream to stringbuilder */
private StringBuilder streamToBuilder(InputStream stream) throws IOException {

    BufferedReader reader = new BufferedReader(new InputStreamReader(stream));
    StringBuilder builder = new StringBuilder();
    String line = null;

    do {    /* iterate until EOF */

        line = reader.readLine();
        if (line != null) {
            builder.append(line);
        }
```

```
    } while (line != null);

    return (builder);
}

/* wrap jsonobject result to searchresult bean */
private SearchResultBean jsontoBean(JSONObject result) throws JSONException {

    /* Fetch details of search result */
    String cacheUrl = result.getString("cacheUrl");
    String content = result.getString("content");
    String title = result.getString("title");
    String titleNoFormatting = result.getString("titleNoFormatting");
    String unescapedUrl = result.getString("url");
    String url = result.getString("url");
    String visibleUrl = result.getString("visibleUrl");

    /* Generate random    oolean    */
    UUID randomUUID = UUID.randomUUID();
    String    oolean    = randomUUID.toString();

    /* Initialize search result bean */
    SearchResultBean resultbean = new SearchResultBean();

    /* Populate the bean */
    resultbean.setCacheUrl(cacheUrl);
```

```
        resultbean.setContent(content);
        resultbean.setTitle(title);
        resultbean.setTitleNoFormatting(titleNoFormatting);
        resultbean.setUnescapedUrl(unescapedUrl);
        resultbean.setUrl(url);
        resultbean.setVisibleUrl(visibleUrl);
        resultbean.setID(   oolean   );

        return (resultbean);
    }
}
```

idexterAction.java

```
package actions;

import beans.CategoryBean;
import beans.SearchResultBean;
import ejb.ActivityFacadeRemote;
import ejb.URLDetailFacadeRemote;
import entity.Activity;
import entity.Category;
import entity.Channel;
import entity.URL;
import entity.URLDetail;
import entity.User;
```

Dexter

Making search web 2.0 ready!

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import javax.naming.NamingException;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForward;
import utilities.DateUtility;
import utilities.EJBUtility;
import utilities.UserUtility;

public class IdexterAction extends Action {

    private final static String KEY = “idexter”;
    private String page;
    private String timeframe;
    private String category;
    private String visibleURL;
    private HttpSession session;
```

```
private ServletContext servletContext;
ArrayList<SearchResultBean> resultList;
private User user;

@Override
public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {

    /* Get Servlet Context */
    servletContext =    oolean   .getServletContext();

    /* Get Request Parameters */
    page = request.getParameter("page");
    timeframe = request.getParameter("timeframe");
    visibleURL = request.getParameter("channel");
    category = request.getParameter("category");

    /* Get Current Session */
    session = request.getSession();

    /* Get Current User */
    user = (User) session.getAttribute("userRecord");

    /* Get ArrayList of Submitted ResultBeans */
    resultList = getSubmittedResults();
```

```
    /* Sort ArrayList wrt. Votes */
    sortSubmittedResults();

    /* Save ArrayList in Servlet Context */
    servletContext.setAttribute(KEY, resultList);

    /* Get Redirect Flag */
    String redirectFlag = getredirectFlag();

    /* Return */
    return mapping.findForward(redirectFlag);
}

private void addRecord(URLDetail urlDetailRecord) throws NamingException {

    /* Create a Result Bean */
    SearchResultBean resultBean = new SearchResultBean();

    /* Populate Result Bean with Record */
    populateResultBean(urlDetailRecord, resultBean);

    /* Add Result Bean to ArrayList */
    resultList.add(resultBean);

}
```

Dexter

Making search web 2.0 ready!

```
    private    oolean checkCategory(URLDetail record) {

        /* Get Record Category Name */
        Category recordCategory = record.getUrl().getCategory();

        /* Check for Parent Category Filter */
        ArrayList<CategoryBean> childCategoryList = checkParentCategory();

        if (childCategoryList != null) {

            if
(recordCategory.getParentCategory().getCategoryName().equalsIgnoreCase(category)) {
                return true;
            } else {
                return false;
            }

        } else {

            if (recordCategory.getCategoryName().equalsIgnoreCase(category)) {
                return true;
            } else {
                return false;
            }
        }
```

```
}

private    oolean checkChannel(URLDetail record) {
    if (record.getVisibleURL().equalsIgnoreCase(visibleURL)) {
        return true;
    } else {
        return false;
    }
}

private ArrayList<CategoryBean> checkParentCategory() {

    /* Get Category Bean */
    ArrayList<CategoryBean> categoryList = (ArrayList<CategoryBean>)
servletContext.getAttribute("category");

    /* Get Iterator */
    Iterator<CategoryBean> iterator = categoryList.iterator();

    /* Iterate ArrayList */
    while (iterator.hasNext()) {
        CategoryBean parentCategoryBean = iterator.next();
        if (parentCategoryBean.getCategoryname().equalsIgnoreCase(category)) {
            return (parentCategoryBean.getChild());
        }
    }
```

Dexter

Making search web 2.0 ready!

```
    /* Category Filter Selected is Not Parent Category */
    return (null);
}

private    oolean checkTimeFrame(URLDetail record) {

    /* Get Days Passed Since Submission */
    String daysPassed = DateUtility.getDaysPassed(record.getUrl().getSubmissionDate());
    int days = Integer.parseInt(daysPassed);

    /* Check Timeframe */
    if (timeframe.equalsIgnoreCase("today")) {
        if (daysPassed.equalsIgnoreCase("0")) {
            return (true);
        } else {
            return (false);
        }
    }

    if (timeframe.equalsIgnoreCase("week")) {
        if (days <= 7) {
            return (true);
        } else {
            return (false);
        }
```

```
    }

    if (timeframe.equalsIgnoreCase("month")) {
        if (days <= 30) {
            return (true);
        } else {
            return (false);
        }
    }

    if (timeframe.equalsIgnoreCase("year")) {
        if (days <= 365) {
            return (true);
        } else {
            return (false);
        }
    }
    return false;
}

private String getChannelName(URL url) {

    String channelName;

    Channel channel = url.getChannel();
    if (channel != null) {
```

```
            channelName = channel.getChannelName();
        } else {
            channelName = null;
        }

        return channelName;
    }

    private ArrayList<SearchResultBean> getSubmittedResults() throws NamingException {

        /* Create ArrayList */
        resultList = new ArrayList<SearchResultBean>();

        /* Get Remote Object */
        URLDetailFacadeRemote urlDetailRemote = (URLDetailFacadeRemote)
EJBUtility.lookup("URLDetailFacade");

        /* Get Recordset */
        List<URLDetail> urlDetailRecordset = urlDetailRemote.findAll();

        /* Iterate Recordset */
        Iterator<URLDetail> iterator = urlDetailRecordset.iterator();
        while (iterator.hasNext()) {

            URLDetail urlDetailRecord = iterator.next();
```

```
        /* Check Filters */
        if (isFilterON()) {
            if (isFilterConditionSatisfied(urlDetailRecord)) {
                addRecord(urlDetailRecord);
            }
        } else {
            addRecord(urlDetailRecord);
        }
    }

    /* Return ArrayList */
    return resultList;

}

private String getURLID(long id) {
    Long iD = new Long(id);
    return (iD.toString());
}

private String getUserName(User user) {

    String firstName = user.getFirstName();
    String lastName = user.getLastName();
    String fullName = firstName + " " + lastName;
    return (fullName);
```

```
}

private    oolean isFilterConditionSatisfied(URLDetail record) {

      oolean flag = false;

   if (!timeframe.equals(“”)) {
      flag = checkTimeFrame(record);
      if (!flag) {
         return flag;
      }
   }
   if (!visibleURL.equals(“”)) {
      flag = checkChannel(record);
      if (!flag) {
         return flag;
      }
   }
   if (!category.equals(“”)) {
      flag = checkCategory(record);
   }

   return (flag);
}

private    oolean isFilterON() {
```

```
        if (timeframe == null && visibleURL == null && category == null) {
            return false;
        } else {
            if (timeframe.equals("") && visibleURL.equals("") && category.equals("")) {
                return false;
            }
            return true;
        }
    }


    private void populateResultBean(URLDetail record, SearchResultBean bean) throws
NamingException {


        /* Get Remote Objects */
        ActivityFacadeRemote activityRemote = (ActivityFacadeRemote)
EJBUtility.lookup("ActivityFacade");


        /* Get URL Record */
        URL url = record.getUrl();


        /* Check if Current User has Modded the URL */
        if (user != null) {
            Activity activityRecord = activityRemote.findByUserUrl(user, url);


            /* Populate Activity Parameters */
            if (activityRecord != null) {
```

```
        bean.setIfmodded(true);

        /* Check Vote */
        if (activityRecord.isIfVotedUp()) {
            bean.setIfvotedup(true);
        }
    }
}

/* Populate Bean */
bean.setCacheUrl(record.getCacheURL());
bean.setContent(record.getContent());
bean.setID(getURLID(url.getId()));
bean.setTitle(record.getTitle());
bean.setTitleNoFormatting(record.getTitleNoFormating());
bean.setVisibleUrl(record.getVisibleURL());
bean.setUnescapedUrl(url.getUsescapedURL());

/* Populate iDexter Specific Parameters */
bean.setCategory(url.getCategory().getCategoryName());
bean.setChannel(getChannelName(url));
bean.setNumberofreports(url.getNoOfReports());
bean.setSubmissiondate(DateUtility.getDaysPassed(url.getSubmissionDate()));
bean.setUserFullName(getUserName(url.getUser()));
bean.setVoteUp(url.getVoteUp());
```

```
        bean.setVoteDown(url.getVoteDown());
    }

    private void sortSubmittedResults() {
        Collections.sort(resultList);
    }

    private String getredirectFlag() {

        if (UserUtility.userExists(session)) {
            return (“usersuccess”);
        } else {
            return (“nousersuccess”);
        }
    }
}
```

Dexter

Making search web 2.0 ready!

FUTURE POSSIBILITIES

Dexter can be extended to include the following changes to make it a better product.

- Google/OpenID Authentication
- Syncing with Facebook Friends
- Inline Youtube Preview within Google Results
- Adding Yahoo/Live Search
- 'DexterBar' – A toolbar that would solve submission issue.
- Widget for Windows/OSX
- RSS/Atom Support
- iPhone and Android App for Mobile Support
- Email Notification for Shouts/FriendRequests using JavaMail
- 'Did you mean?' support in Google search
- 'Digg style Suggestions' for related keywords

Dexter

Making search web 2.0 ready!

CONCLUSION

Dexter is a social search engine, bookmarking site, content recommendation engine and a social networking site all rolled into one. As it amalgamates so many features Dexter has the potential of becoming your web home.

Search Result are sorted according to the votes every URL receives. Result once voted up stay in your profile for future reference. You can add friends and collaborate in search as URL they are voting up are visible to you.

The iDexter Module allows user to surf the URL which has been submitted into the database thus you get quality links to surf. You can filter results to suit your preferences. It can be a great tool when you just want to explore the internet as the URL which have high vote up are essentially recommended by so many people.

In addition to the other features Dexter allows you to do social networking. You can send messages to your friends see what they are searching and in effect stay connected.

In the end I would like to add a word of caution- Dexter like all social networking sites is bounded by a need to have a minimum critical mass of users. Without having a substantial user base the search results and the recommendation can become lopsided and easily manipulated but as it would achieve the critical mass effects to manipulate the results would yield no result.

Dexter

Making search web 2.0 ready!

APPENDIX A: GLOSSARY

Dexter – A Java EE Web Application based upon Struts and Hibernate Framework building an overlay on top of conventional Web 1.0 search engines to provide human collaboration and participation possibility

Dexter Surf - One of the two possible environments of interaction with the web-app. Surf provides content suggestion based upon popularity of voted search results in a particular timeframe, category and channel.

Dexter Search – One of the two possible environments of interaction with web-app. Search restricts to the classic search scenario waiting for the user to push through a keyboard for relevant response. Search being simplistic to Surf is considerably faster and recommended for normal search scenarios.

Channel – Filter the search results based on a particular website (Wikipedia, Orkut) or a media type (images/videos)

Category – Filter the search results based on field of interest (technology, Fun, Offbeat)

Timeframe – Filter the search results falling in a particular week, month or year.

Votes – A method for the user to participate and help in providing the apparent usefulness of a search URL by voting up or down. Votes eventually affect the positioning of the search URL the next time a search for the same keyword is done.

URL Submission – A possibility to submit an arbitrary URL a user stumbles upon to when browsing and would want the Dexter database to have, being unsure of which keyword the search engine would show the URL against.

Shouts – Messages sent from one Dexter user to another. (Not Instant Message)

Favorite – Bookmark a search result for later.

Tag – keywords stored against a favorite for effective retrieval.

Report Comment – comments could be reported on profanity to the moderator for possible deletion.

Dexter

Making search web 2.0 ready!

Report URL – Search result URL's could be reported on profanity or incorrect category submission to the moderator for possible actions.

Dexter

Making search web 2.0 ready!

APPENDIX B: BIBILIOGRAPHY

We referred the following books during the implementation of the project 'Dexter'

- Thinking in Java by Bruce Eckel
- Apress Pro JSP 2/2.1
- Hibernate in Action by Christian Bauer and Gavin King
- Struts in Action by Cedric Dumoulin and Ted Husted
- Java: The Complete Reference by Herbert Schildt