A
PROJECT REPORT
ON

Web Crawler Supported Smart Search

B.Tech (CE) Sem-VI
In fulfilment of all requirements for

Bachelor of Technology
In
Computer Engineering
SEM VI

In the Subject of
System Design Practice

Dhruv Saraiya (CE-107)(15CEUOS046)
Pratik Patel (CE-087)(15CEUON125)
Pranay Shah (CE-113)(15CEUOS034)

Under the Guidance of
Prof. Jigar M. Pandya
Assistant Professor

DEPARTMENT OF COMPUTER ENGINEERING
FACULTY OF TECHNOLOGY,
DHARMSINH DESAI UNIVERSITY
COLLEGE ROAD, NADIAD- 387001

DHARMSINH DESAI UNIVERSITY

NADIAD-387001, GUJARAT

<u>CERTIFICATE</u>

This is to certify that the project carried out in the subject of System Design Practice titled "WEB CRAWLER SUPPORTED SMART SEARCH" and recorded in this report is a work of

Dhruv Saraiya	CE-107	15CEUOS046
Pratik Patel	CE-087	15CEUON125
Pranay Shah	CE-113	15CEUOS034

Department of Computer Engineering, semester VI. They were involved in Project developing during the period Dec-17 to Apr-2018.

Guide:
Prof. Jigar M. Pandya
Assistant Professor,
CE Dept.,
Faculty of Technology
Dharmsinh Desai University, Nadiad

Dr. C.K.Bhensdadia,
Head, CE Dept.,
Faculty of Technology,
Dharmsinh Desai University, Nadiad

CONTENTS

i. Abstract v

1. Introduction 6

1.1 Project Details: Broad Specifications 7

1.2 Technology Used 7

1.3 Project Planning 9

2. Software Requirement and Specification 10

2.1 Purpose 11

2.2 Scope 11

2.3 Overall Description 11

2.4 External Interface Requirements 14

2.5 Non Functional Requirements 15

3. Design 16

3.1 Use Case Diagram 17

3.2 Data Flow Diagrams 17

3.3 Network Activity Diagram 19

3.4 Gantt Chart 19

4. Implementation 20

4.1 Implementation Environment 21

4.2 Description of Modules 21

4.3 Algorithm 30

5. Testing 32

5.1 Testing Plan 33

5.2 Testing Strategy 33

5.3 Testing Methods 34

5.4 Test Cases 34

6. Screenshots 37

6.1 Home Page 38

6.2 Crawl by Admin 40

6.3 View Result by Admin 41

6.4 Admin Manage 42

6.5 Request Crawl by Employee 43

7. Limitations and Future Enhancements 44

7.1 Limitations 45

7.2 Future Enhancements 45

8. Conclusion 46

9. Bibliography 48

10. Software Version Deployment Document 50

11. Miscellaneous 53

Abstract

A Web crawler is a computer program that browses the World Wide Web in a methodical, automated manner or in an orderly fashion. Web crawling is an important method for collecting data on, and keeping up with, the rapidly expanding Internet. A vast number of web pages are continually being added every day, and information is constantly changing. This project is an overview of Web Crawler Supported Smart Search and the policies like searching, ranking, indexing involved in it.

CHAPTER : 1

INTRODUCTION

1. Introduction

1.1 Project Details: Broad Specifications

Web Crawler Supported Smart Search is a web application which allows normal users to search through local intranet and allows employees and admin to add seed link to be crawled. So, basically it is *“From Employees For Employees”* to share knowledge.

1.2 Technology Used

- **Front End** : HTML , CSS , JS, Bootstrap
  - **HTML:** Hypertext Markup Language is the standard markup language for creating web pages and web applications. With Cascading Style Sheets and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web.

  - **CSS:** Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language.

  - **JS:** JavaScript, often abbreviated as JS, is a high-level, interpreted programming language. It is a language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm.

  - **Bootstrap:** Bootstrap is a free and open-source front-end library for designing websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.

- **Binding** : Flask Framework(Python)
  Flask is a micro web framework written in Python and based on the Jinja2 template engine. It is BSD licensed. The latest stable version of Flask is 0.12.2 as of May 2017

- **Back End** : Python
  Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

- **Database** : MongoDB
  MongoDB is an open-source cross-platform document-oriented database program. Classified as NoSQL database program, MongoDB uses JSON-like documents with schemas .(JSON - Javascript Object Notation)

- **Indexing** : Apache Solr
  Solr is an open source enterprise search platform, written in Java, from the Apache Lucene project. Its major features include full-text search, hit highlighting, faceted search, real-time indexing.

1.3 Project Planning

- **Requirement Gathering:** An analysis of search engine methodologies and how web crawler works in internet web structure.

- **System Design:** Data collection for system design. Database structure design. Preparation of function specification.

- **Systems Development:** Database server, web server, indexing server and four modules of systems will be developed in sequence. The four modules are spider module, indexer module, ranker module and query module.

- **Systems Testing:** Final systems testing, unit testing and integration testing will be performed.

- **Deployment:** System is currently deployed to local server only which is localhost.

CHAPTER : 2

SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS)

2. Software Requirement Specifications

2.1 Purpose

This is a SRS document which refers to Web Crawler Supported Smart Search Release 2018 version 1. It describes the functionality and specification of how the web crawler will help in serving the employees of a firm in an efficient way.

2.2 Scope

This system is designed to crawl any website where employees and administrators can give any seed link to crawl that website and they also can use smart search to efficiently search across collected data. It can not search outside collected data and currently dynamic ranking is not supported.

2.3 Overall Description

- **Product Perspective**

This software is developed as a part of course work the subject "System Design Practice". The software aims to crawl/search online with ease. The web- app's main perspective is towards efficient ranking, searching and provide a user-flexible system.

- **Product Functions**

  - **Crawling**:
    When a spider is building its lists, it is called Web crawling. In this process spider crawls pages starting with seed link.

- **Ranking:**

  It is the process of giving rank to collected links based on their inter relation(tree structure), the higher the rank the greater chance of it to be displayed on first page of search result.

- **Indexing**:

  It is the process of giving index to crawled pages so that at search time they are accessed faster compare to traditional database queries.

- **Searching:**

  When user enter any keyword(s) and the result is displayed, this process is called searching where links to be displayed are searched from already indexed data and most appropriate results are displayed first.

- **User Classes and Characteristics**

Basically there are two types of end users i.e.,

- **Administrator**:

  Administrator of a firm can manage which links to crawl and which to not, also she can add links to crawl and manage crawled data. She can manage employees too.
- **Employees:**

  Employees of the firm can provide seed link to crawl and they can search for any detail related to its firm in user section of the system.

- **Operating Environment**

The system is a web-app and not an android or iOS application because development in different operating environment will take time and cross-platform development is not the base of the development team and a web-app can be used by everyone in an efficient way. Therefore, after hosting this web-app can be accessible from any web browser.

- **Design and Implementation Constraints**

The constraints are financial as well as at a corporate level. The system can only crawl allowed pages so if website owner or generally pages forbid some pages which require authentication or authorization will not be crawled.

- **Assumptions and Dependencies**

System will not work if appropriate network is not available. Mainline network of system highly needs electricity to work on. System requires to be given a seed link to crawl, it cannot crawl randomly any website found in between of any previous crawling job.

2.4 External Interface Requirements

- **User Interfaces**

Application can be accessed through any browser interface. The software will be fairly compatible with

Microsoft Internet Explorer Version 6 and above or other modern web browsers.

- **Hardware Interfaces**

  - **Server Side**:

    - Operating System: Windows.
    - Processor: Pentium 3.0 GHz or higher.
    - RAM: 2GB or higher.
    - Hard-Disk: 100GB or more.

  - **Client Side:**

    - Operating System: Windows 7,8, 8.1,10.
    - Processor: Pentium 3.0 GHz or higher.
    - RAM: 2GB or higher.

- **Software Interfaces**

  - **Client Side:**

HTML5 supported Web Browsers, Windows 7, 8, 8.1, 10, MAC OS, Linux (All Flavors).

  - **Server Side:**

Windows, MongoDB Database, Apache Solr(core), Python Interpreter.

- **Communication Interfaces**

The Web Crawler Supported Smart Search shall use the HTTP protocol for communication over the internet and for the intranet communication will be through TCP/IP protocol suite.

2.5 Non Functional Requirements

- **Performance Requirements**

The average response time for a user is 0.36 sec. The expected accuracy of output is 90%.

For faster access we have provided solr indexing.

Ranking is done in main memory so that is very fast.

- **Safety Requirements**

If any testing purpose link of more than one domain name is given then it may lead the system to crash (e.g. http://newsbytag.herokuapp.com)

If any link contains non-html page then is will be discarded immediately.

- **Software Quality Attributes**

All the software modules are developed in python, which makes the system extensible and robust. Secondly the system will provide the user with easy to use and understandable GUI interface.

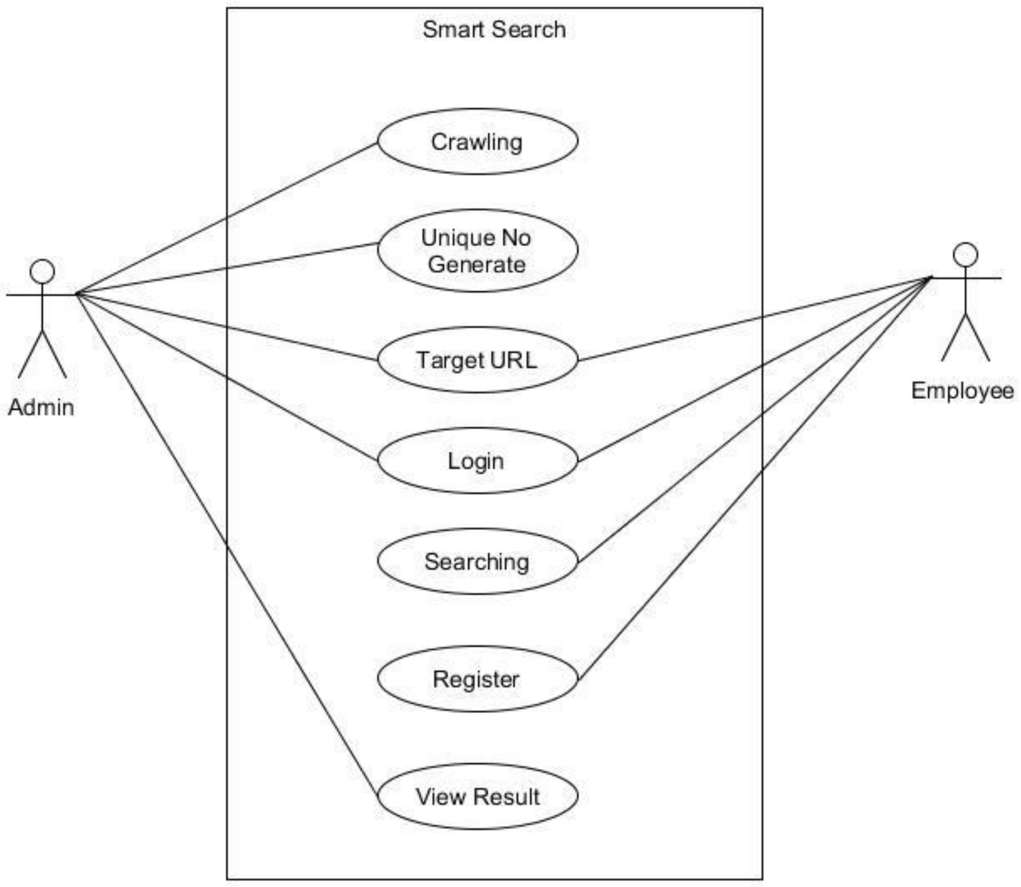
For good interface we have used standard bootstrap library which also provides consistency.

CHAPTER : 3

DESIGN

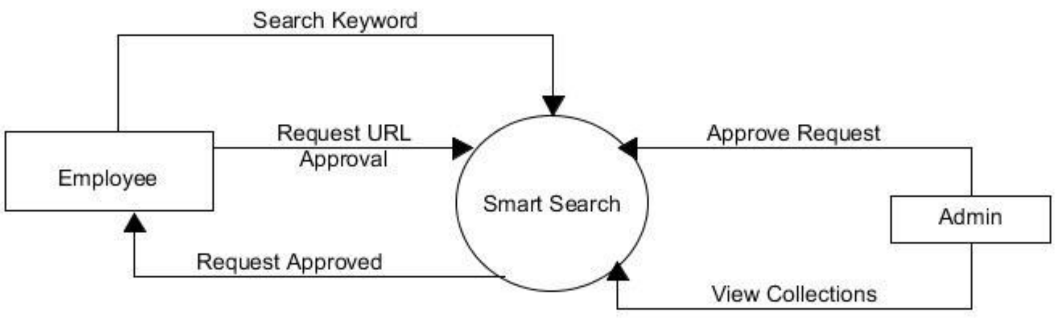
3. Design

3.1 Use-Case Diagram

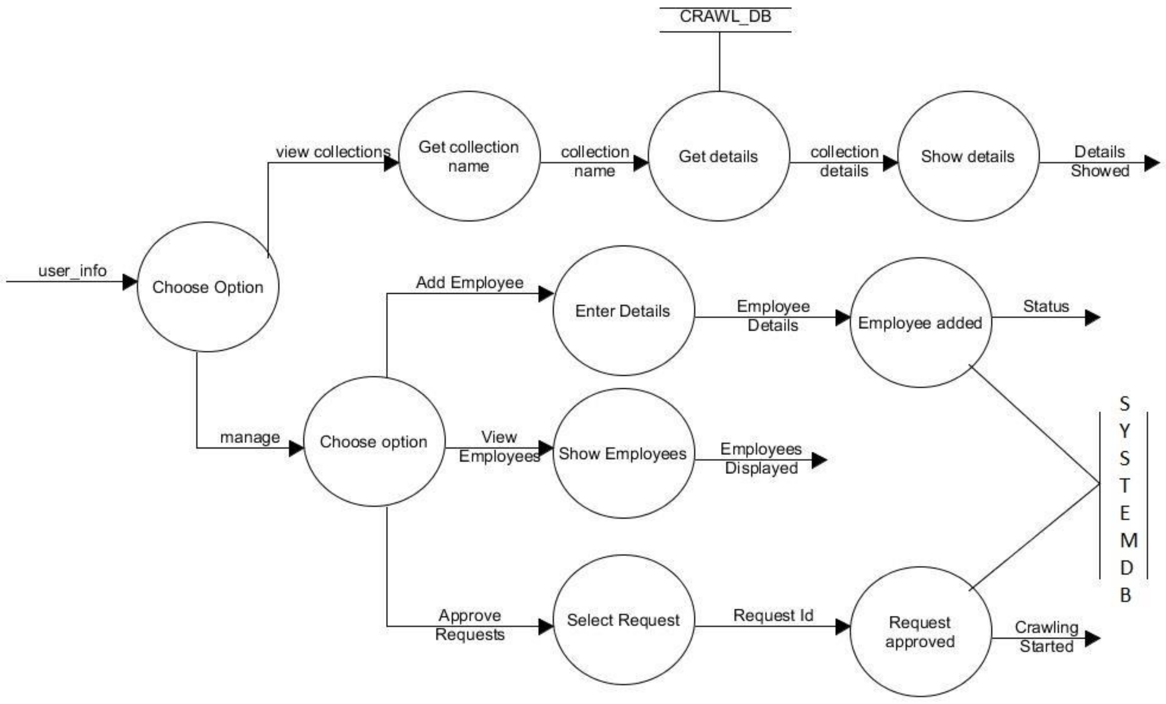


3.2 Data Flow Diagrams

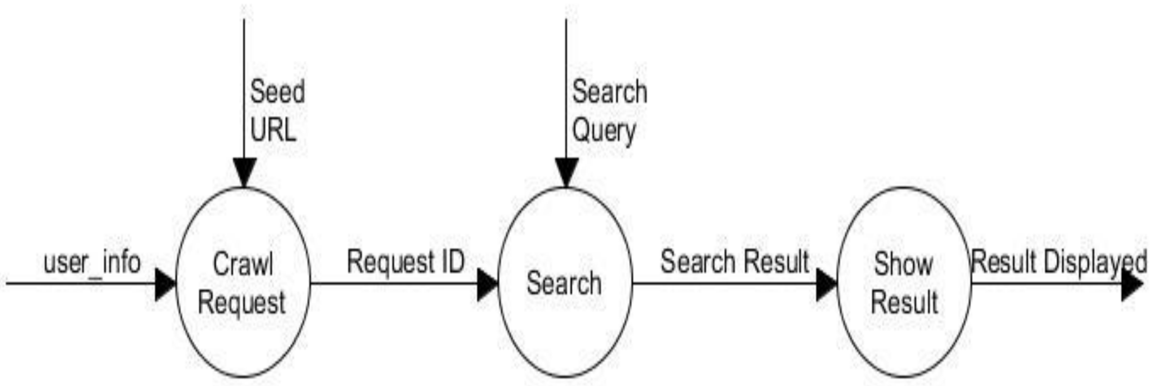
- **Level - 0**



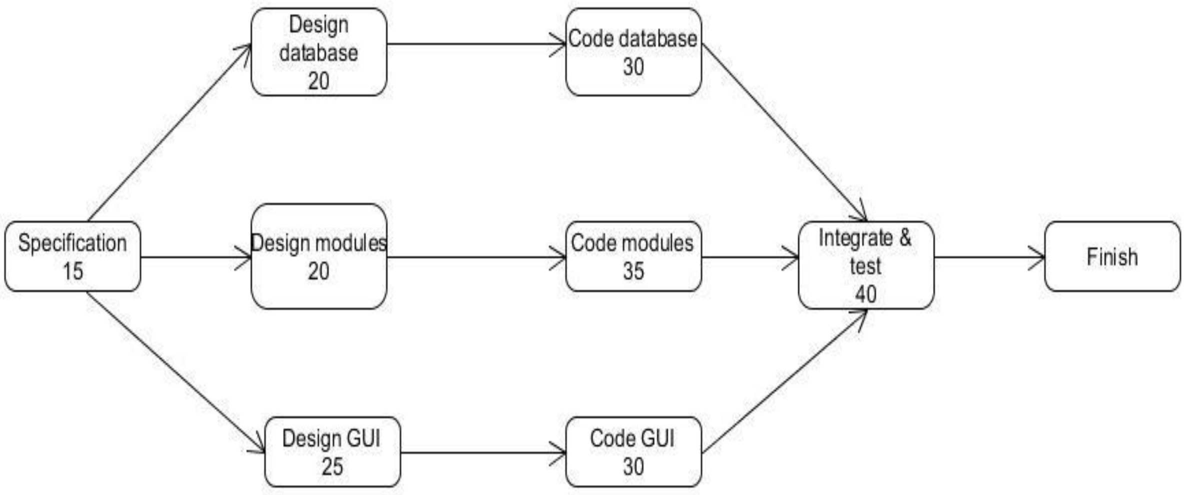
- **Level -1(a)**



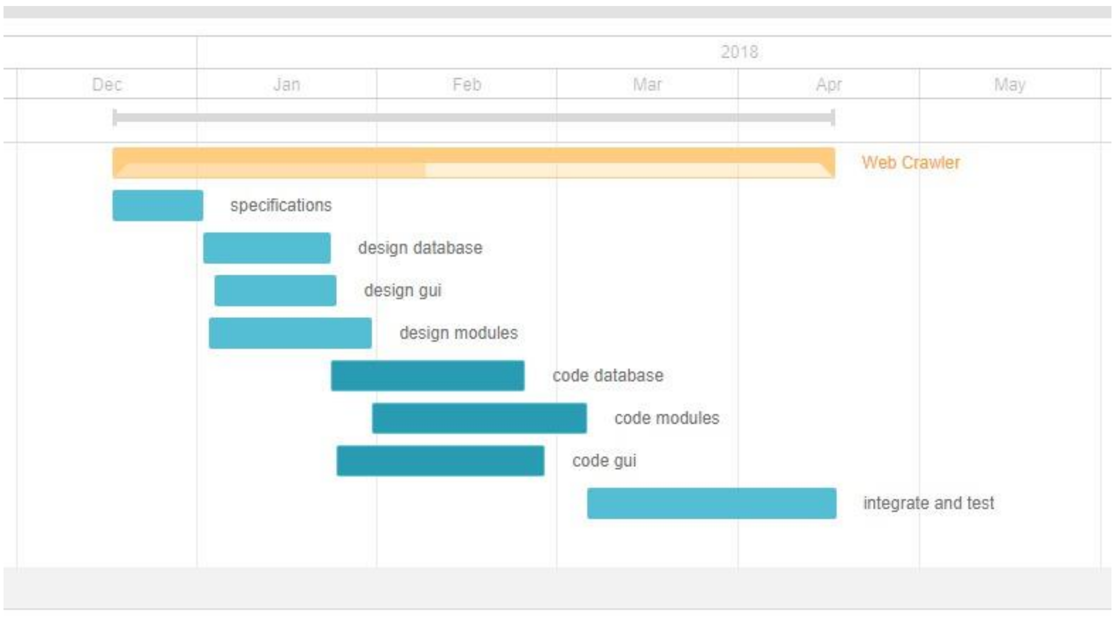
- **Level -1(b)**



3.3 Network Activity Diagram


$$\begin{tikzcd}
 & \text{Design database 20} \arrow[r] & \text{Code database 30} \arrow[dr] & & \\
\text{Specification 15} \arrow[ur] \arrow[r] \arrow[dr] & \text{Design modules 20} \arrow[r] & \text{Code modules 35} \arrow[r] & \text{Integrate \& test 40} \arrow[r] & \text{Finish} \\
 & \text{Design GUI 25} \arrow[r] & \text{Code GUI 30} \arrow[ur] & &
\end{tikzcd}$$

3.4 Gantt Chart



CHAPTER : 4

IMPLEMENTATION

4. Implementation

4.1 Implementation Environment

- Python environment in Windows 10 (version=3.6.2)
- MongoDB Client in Windows 10 (version=3.6.2)
- Apache solr in Windows 10 (version=7.2.1)

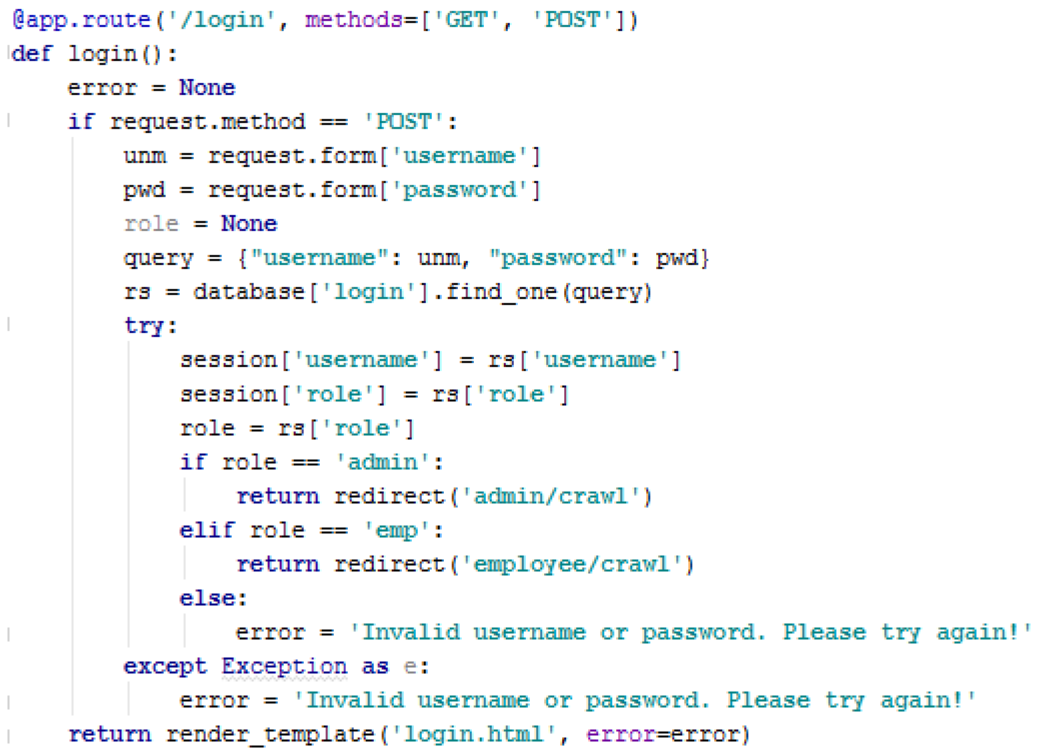
4.2 Description of Modules

- **Register / Log - In Module**

It is used to store information of employees who are new to the system and accessing the system for the first time and also authenticating the users before they can add link to the crawling job of the system.

Input : User's information or credentials
Output:Stored or Verified successfully
Processing: Check user's credentials in the database while logging in or store them in the database while registering new user.

Code Snippet



```
@app.route('/login', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        unm = request.form['username']
        pwd = request.form['password']
        role = None
        query = {"username": unm, "password": pwd}
        rs = database['login'].find_one(query)
        try:
            session['username'] = rs['username']
            session['role'] = rs['role']
            role = rs['role']
            if role == 'admin':
                return redirect('admin/crawl')
            elif role == 'emp':
                return redirect('employee/crawl')
            else:
                error = 'Invalid username or password. Please try again!'
        except Exception as e:
            error = 'Invalid username or password. Please try again!'
    return render_template('login.html', error=error)
```

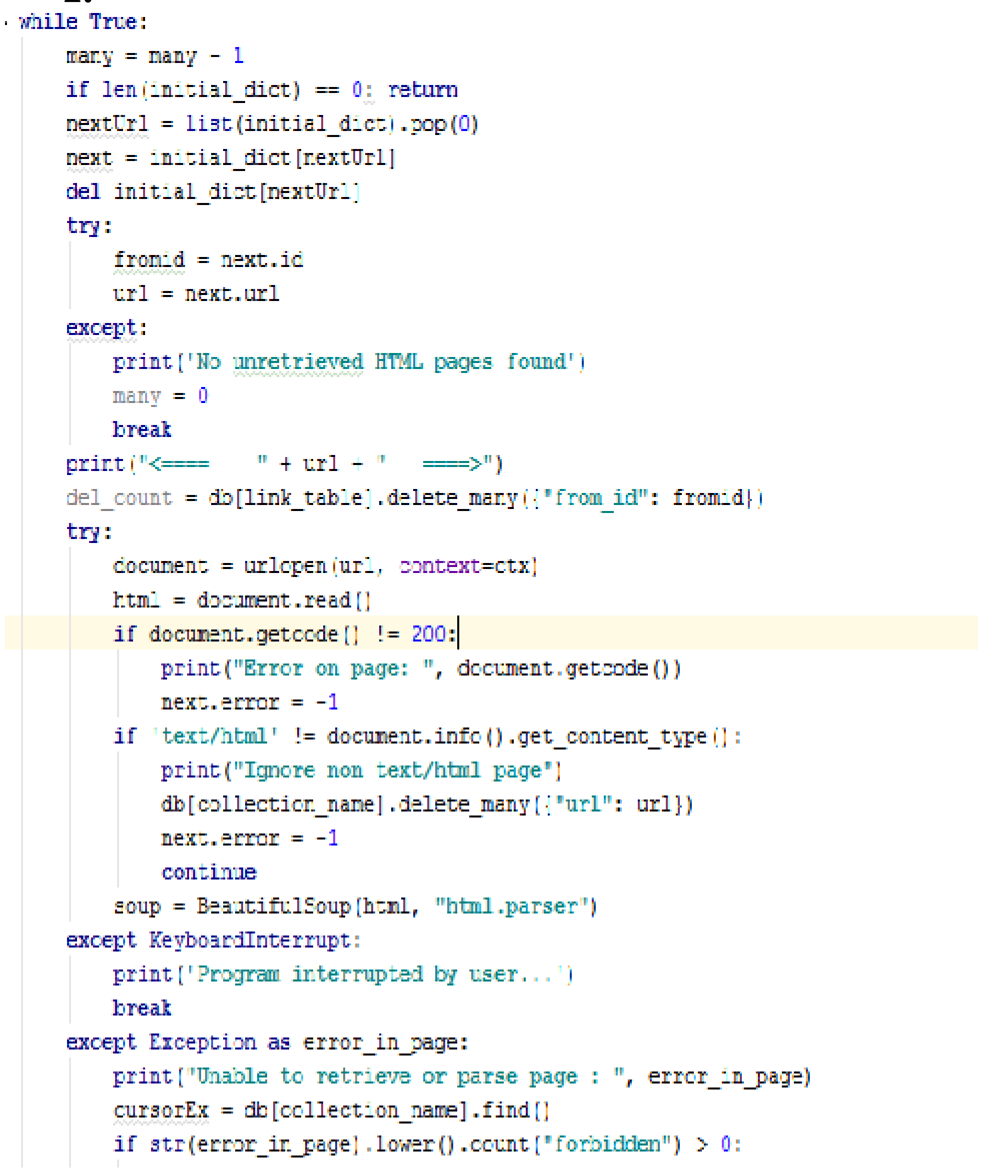
➢ Spider Module

This module does crawling of urls from queue of pending crawling jobs.

Input : seed url’s queue
Output: Crawled data stored in MongoDB
Processing: The spider first search in Db whether crawling of this project is started or is yet to be started. It then fetches the content of the current crawling page, we have configured it for fetching of tags like ‘anchor’(it gives reference of other pages from this page and gives keywords too), ‘meta’(it gives details about content of that page), and ’title’ tags. It then stores the anchor links fetched in the pending queue in MongoDb which are having domain name as the current working domain(Basically spider will not crawl out of the domain links). It then continues crawling by fetching next link from queue of current domain until maximum number of pages(links) is reached.

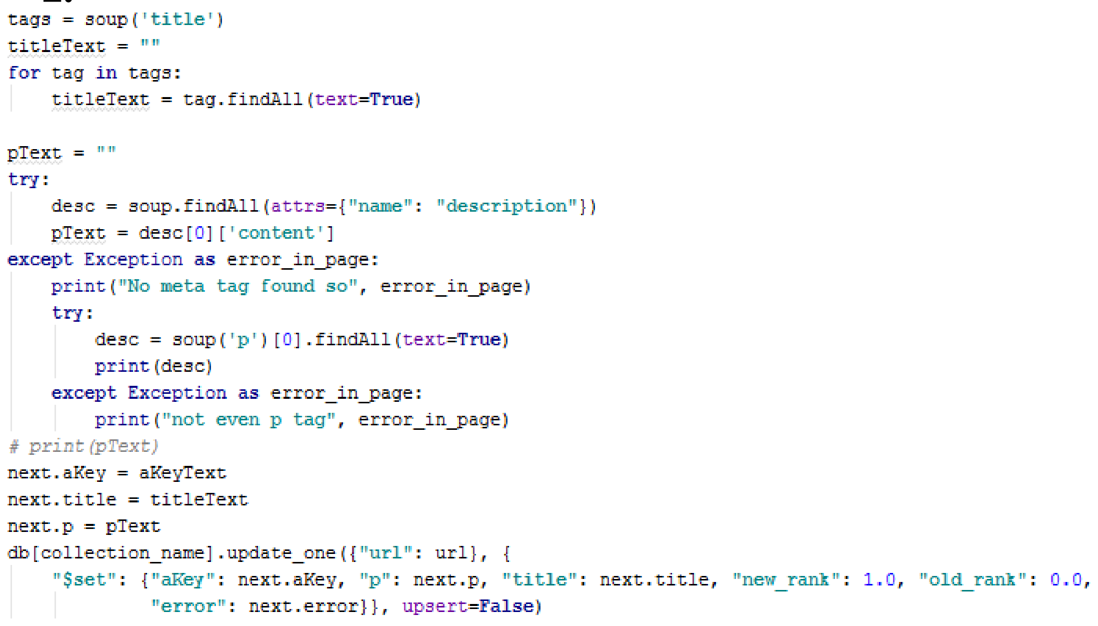
Code Snippet

1.



```
while True:
    many = many - 1
    if len(initial_dict) == 0: return
    nextUrl = list(initial_dict).pop(0)
    next = initial_dict[nextUrl]
    del initial_dict[nextUrl]
    try:
        fromid = next.id
        url = next.url
    except:
        print('No unretrieved HTML pages found')
        many = 0
        break
    print("<====      " + url + "      ====>")
    del_count = db[link_table].delete_many({"from_id": fromid})
    try:
        document = urlopen(url, context=ctx)
        html = document.read()
        if document.getcode() != 200:
            print("Error on page: ", document.getcode())
            next.error = -1
        if 'text/html' != document.info().get_content_type():
            print("Ignore non text/html page")
            db[collection_name].delete_many({"url": url})
            next.error = -1
            continue
        soup = BeautifulSoup(html, "html.parser")
    except KeyboardInterrupt:
        print('Program interrupted by user...')
        break
    except Exception as error_in_page:
        print("Unable to retrieve or parse page : ", error_in_page)
        cursorEx = db[collection_name].find()
        if str(error_in_page).lower().count("forbidden") > 0:
```

2.



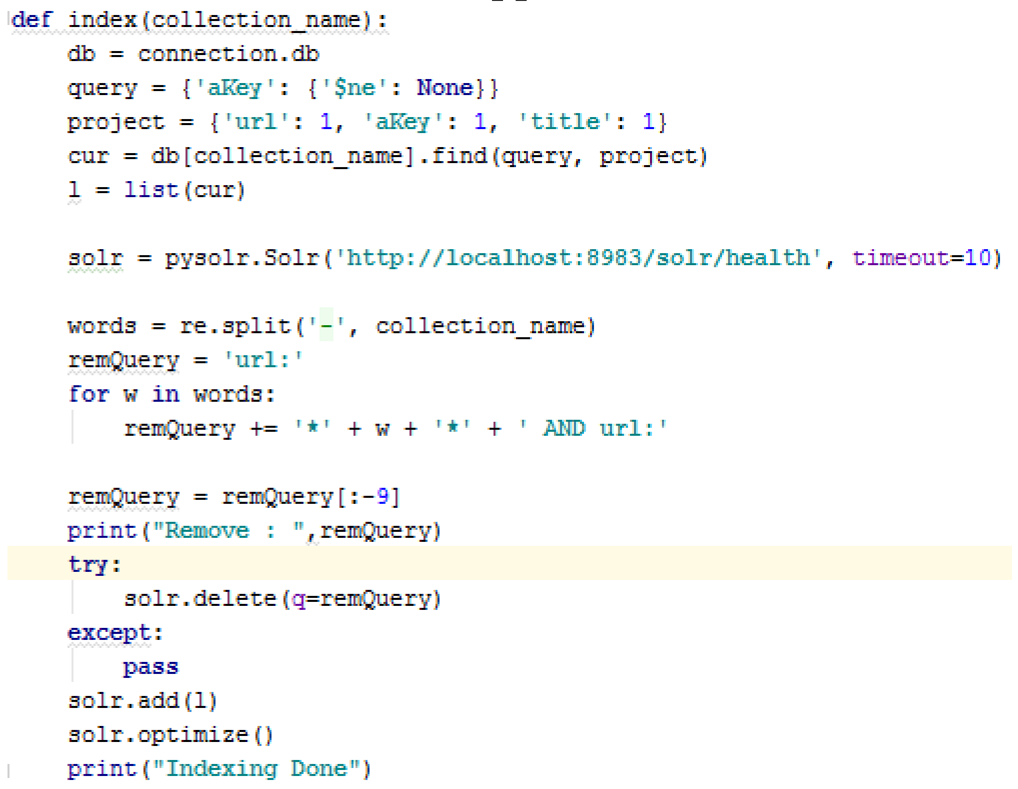
```
tags = soup('title')
titleText = ""
for tag in tags:
    titleText = tag.findAll(text=True)

pText = ""
try:
    desc = soup.findAll(attrs={"name": "description"})
    pText = desc[0]['content']
except Exception as error_in_page:
    print("No meta tag found so", error_in_page)
    try:
        desc = soup('p')[0].findAll(text=True)
        print(desc)
    except Exception as error_in_page:
        print("not even p tag", error_in_page)
# print(pText)
next.aKey = aKeyText
next.title = titleText
next.p = pText
db[collection_name].update_one({"url": url}, {
    "$set": {"aKey": next.aKey, "p": next.p, "title": next.title, "new_rank": 1.0, "old_rank": 0.0,
             "error": next.error}}, upsert=False)
```

➢ Indexer Module

Input : Crawled data
Output: Indexed data stored at Apache solr's core.
Processing:

This module will be called in between any crawling job is finished and new job is to be started, this will index the data which is collected in previous crawling job. The indexing will be taken care by pysolr which is the light-weight wrapper for Apache solr.

Code Snippet



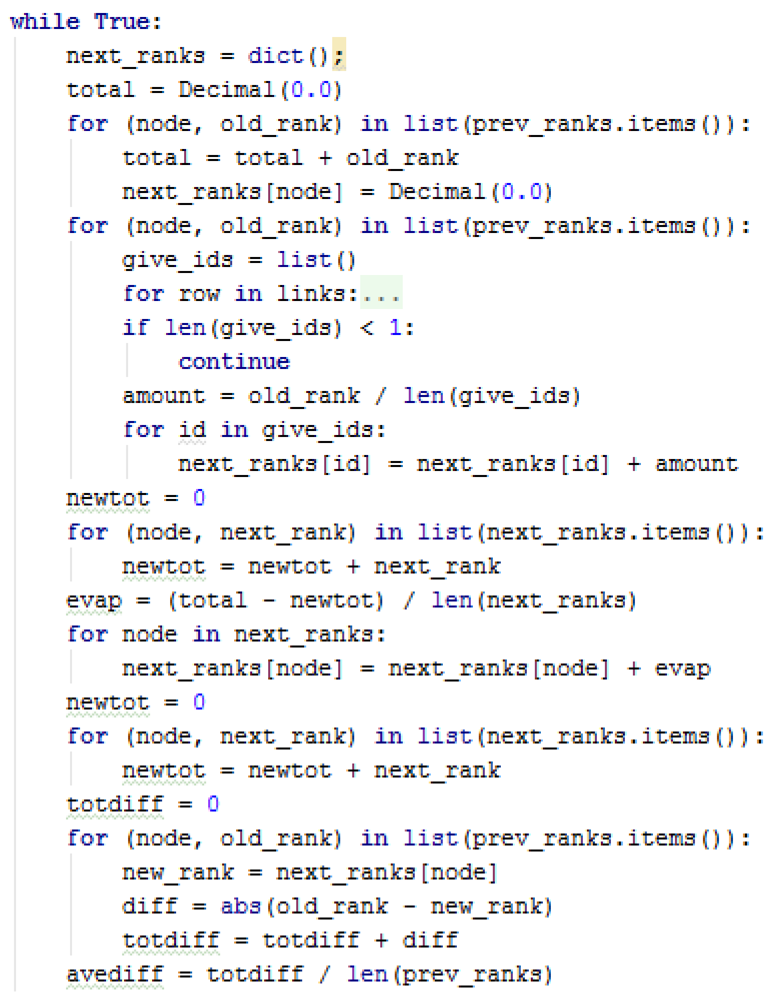
```python
def index(collection_name):
    db = connection.db
    query = {'aKey': {'$ne': None}}
    project = {'url': 1, 'aKey': 1, 'title': 1}
    cur = db[collection_name].find(query, project)
    l = list(cur)

    solr = pysolr.Solr('http://localhost:8983/solr/health', timeout=10)

    words = re.split('-', collection_name)
    remQuery = 'url:'
    for w in words:
        remQuery += '*' + w + '*' + ' AND url:'

    remQuery = remQuery[:-9]
    print("Remove : ",remQuery)
    try:
        solr.delete(q=remQuery)
    except:
        pass
    solr.add(l)
    solr.optimize()
    print("Indexing Done")
```

➢ Ranking Module

Input : Crawled data
Output: Ranked pages stored in MongoDB
Processing:

This module will be called in between any crawling job is finished and new job is to be started, this will rank the pages of the domain which is just crawled. The ranking is based on page-rank algorithm which is described below. After ranking is completed it updates it in MongoDb for permanent storage.

Code Snippet



```
while True:
    next_ranks = dict();
    total = Decimal(0.0)
    for (node, old_rank) in list(prev_ranks.items()):
        total = total + old_rank
        next_ranks[node] = Decimal(0.0)
    for (node, old_rank) in list(prev_ranks.items()):
        give_ids = list()
        for row in links:...
        if len(give_ids) < 1:
            continue
        amount = old_rank / len(give_ids)
        for id in give_ids:
            next_ranks[id] = next_ranks[id] + amount
    newtot = 0
    for (node, next_rank) in list(next_ranks.items()):
        newtot = newtot + next_rank
    evap = (total - newtot) / len(next_ranks)
    for node in next_ranks:
        next_ranks[node] = next_ranks[node] + evap
    newtot = 0
    for (node, next_rank) in list(next_ranks.items()):
        newtot = newtot + next_rank
    totdiff = 0
    for (node, old_rank) in list(prev_ranks.items()):
        new_rank = next_ranks[node]
        diff = abs(old_rank - new_rank)
        totdiff = totdiff + diff
    avediff = totdiff / len(prev_ranks)
```

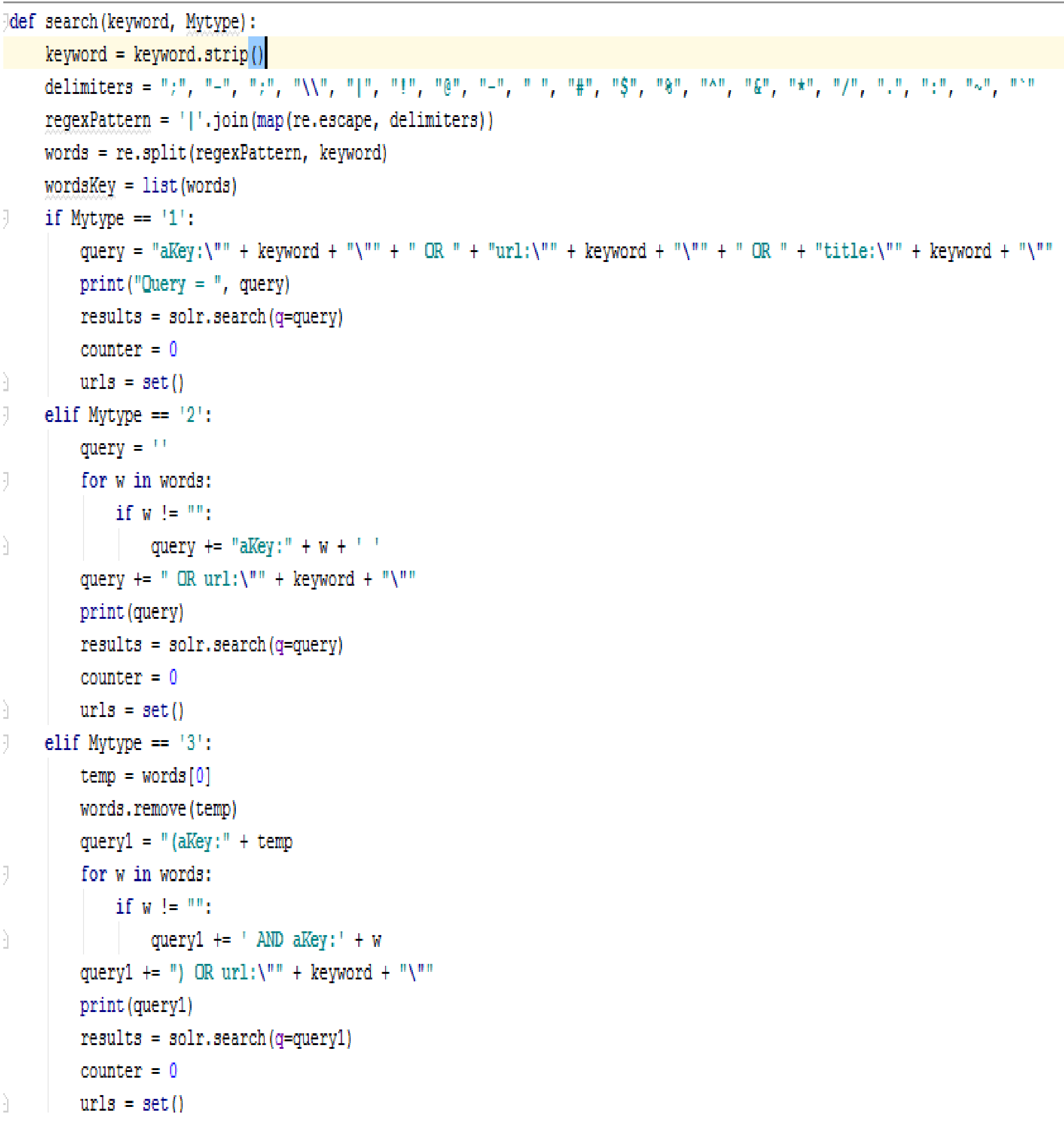
➢ Searching Module

Input : Indexed data
Output: Most matching Urls.
Processing:

This module will be called when user search for any keyword in user section. This will find the documents which have the keyword in its fields, this finding is based on one of two options “any word” or “all words” ,there are different lucene queries for both of these options. After finding those records it will sort those records based on their ranks calculated in ranking module and display those urls with some description about content of that page(which was fetched from meta tags).

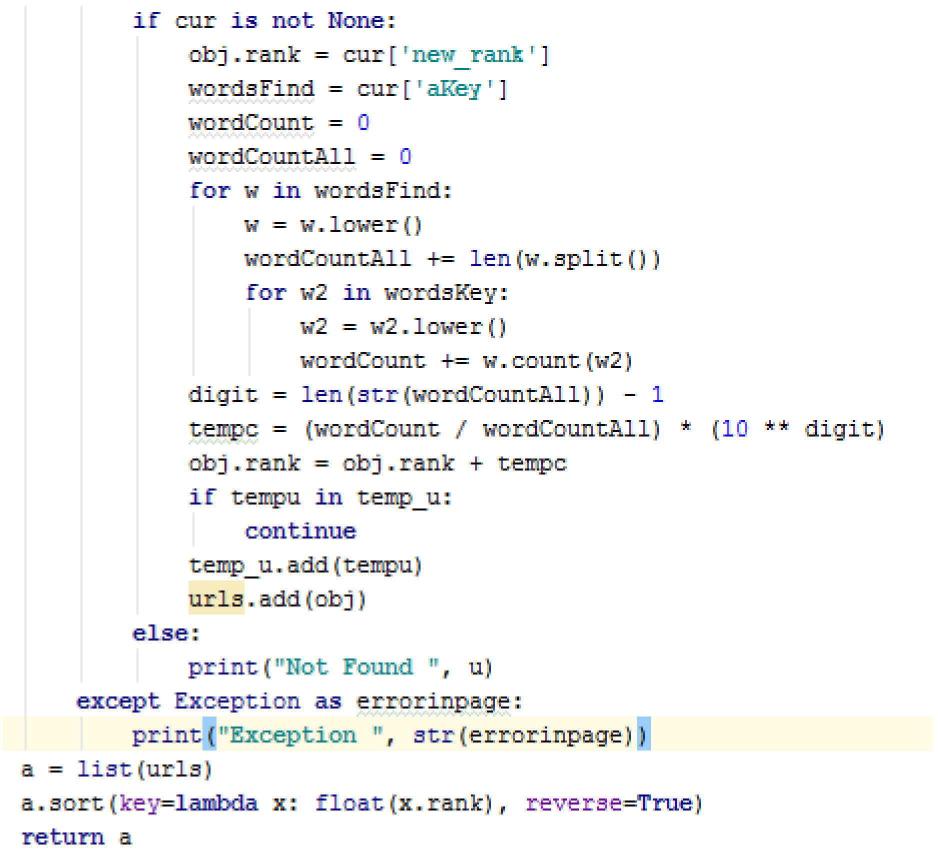
Code Snippet

1.



```
def search(keyword, Mytype):
    keyword = keyword.strip()
    delimiters = ";", "-", ";", "\\", "|", "!", "@", "-", " ", "#", "$", "%", "^", "&", "*", "/", ".", ":", "~", "`"
    regexPattern = '|'.join(map(re.escape, delimiters))
    words = re.split(regexPattern, keyword)
    wordsKey = list(words)
    if Mytype == '1':
        query = "aKey:\"" + keyword + "\"" + " OR " + "url:\"" + keyword + "\"" + " OR " + "title:\"" + keyword + "\""
        print("Query = ", query)
        results = solr.search(q=query)
        counter = 0
        urls = set()
    elif Mytype == '2':
        query = ''
        for w in words:
            if w != "":
                query += "aKey:" + w + ' '
        query += " OR url:\"" + keyword + "\""
        print(query)
        results = solr.search(q=query)
        counter = 0
        urls = set()
    elif Mytype == '3':
        temp = words[0]
        words.remove(temp)
        query1 = "(aKey:" + temp
        for w in words:
            if w != "":
                query1 += ' AND aKey:' + w
        query1 += ") OR url:\"" + keyword + "\""
        print(query1)
        results = solr.search(q=query1)
        counter = 0
        urls = set()
```

2.



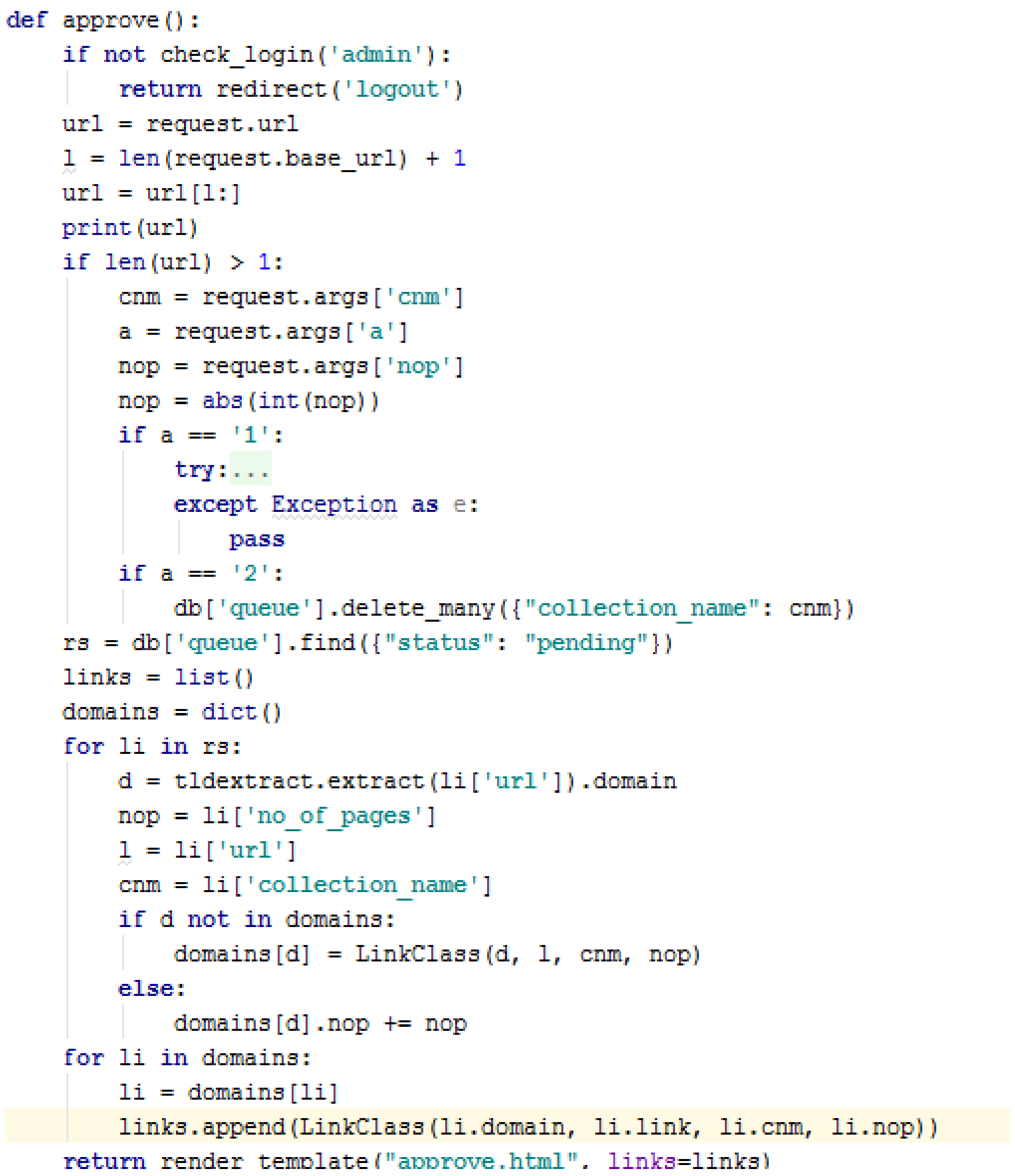
```
        if cur is not None:
            obj.rank = cur['new_rank']
            wordsFind = cur['aKey']
            wordCount = 0
            wordCountAll = 0
            for w in wordsFind:
                w = w.lower()
                wordCountAll += len(w.split())
                for w2 in wordsKey:
                    w2 = w2.lower()
                    wordCount += w.count(w2)
            digit = len(str(wordCountAll)) - 1
            tempc = (wordCount / wordCountAll) * (10 ** digit)
            obj.rank = obj.rank + tempc
            if tempu in temp_u:
                continue
            temp_u.add(tempu)
            urls.add(obj)
        else:
            print("Not Found ", u)
    except Exception as errorinpage:
        print("Exception ", str(errorinpage))
a = list(urls)
a.sort(key=lambda x: float(x.rank), reverse=True)
return a
```

➢ Admin Control Module

Input : Pending projects for approvement
Output: Status after approvement.
Processing:

This module is used by the administrator of the system in order to maintain control of the seed urls suggested by employees to add to crawling queue or not. Here, all the seed urls listed by the users come for verification .Thus, only after the admin approves the url, it can be listed for the crawling job. If admin rejects any url then it will be discarded. Admin can also modify number of pages to be crawled in particular job.

Code Snippet

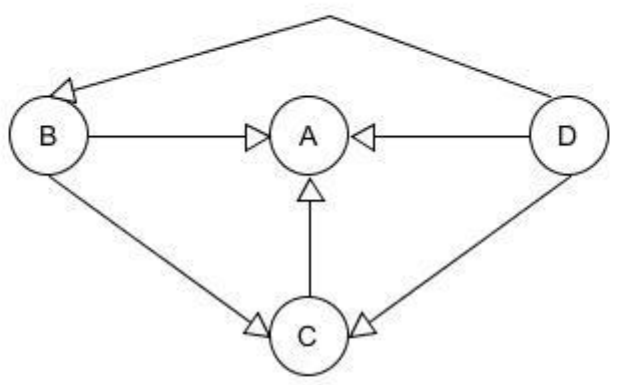


```python
def approve():
    if not check_login('admin'):
        return redirect('logout')
    url = request.url
    l = len(request.base_url) + 1
    url = url[l:]
    print(url)
    if len(url) > 1:
        cnm = request.args['cnm']
        a = request.args['a']
        nop = request.args['nop']
        nop = abs(int(nop))
        if a == '1':
            try:...
            except Exception as e:
                pass
        if a == '2':
            db['queue'].delete_many({"collection_name": cnm})
    rs = db['queue'].find({"status": "pending"})
    links = list()
    domains = dict()
    for li in rs:
        d = tldextract.extract(li['url']).domain
        nop = li['no_of_pages']
        l = li['url']
        cnm = li['collection_name']
        if d not in domains:
            domains[d] = LinkClass(d, l, cnm, nop)
        else:
            domains[d].nop += nop
    for li in domains:
        li = domains[li]
        links.append(LinkClass(li.domain, li.link, li.cnm, li.nop))
    return render_template("approve.html", links=links)
```

4.3 Algorithm

➢ **Page Rank Algorithm**

- **How Page Rank Works**:



Assume a small universe of four web pages: **A**, **B**, **C** and **D**. The initial approximation of Page Rank would be evenly divided between these four documents. Hence, each document would begin with an estimated Page Rank of 0.25.

In the original form of Page Rank initial values were simply 1. This meant that the sum of all pages was the total number of pages on the web. Later versions of Page Rank would assume a probability distribution between 0 and 1. Here we're going to simply use a probability distribution hence the initial value of 0.25.

If pages **B**, **C**, and **D** each only link to **A**, they would each confer 0.25 Page Rank to **A**. All Page Rank i.e. **PR ( )** in this simplistic system would thus gather to **A** because all links would be pointing to **A**.

$$PR(A) = PR(B) + PR(C) + PR(D).$$

This is 0.75.

Again, suppose page **B** also has a link to page **C**, and page **D** has links to all three pages. The value of the link-votes is divided among all the outbound links on a page. Thus, page **B** gives a vote worth 0.125 to page **A** and a vote worth 0.125 to page **C**. Only one third of **D**'s Page Rank is counted for A's Page Rank (approximately 0.083).

$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{1} + \frac{PR(D)}{3}.$$

In other words, the Page Rank conferred by an outbound link **L( )** is equal to the document's own Page Rank score divided by the normalized number of outbound links (it is assumed that links to specific URLs only count once per document).

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}.$$

In the general case, the Page Rank value for any page **u** can be expressed as:

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)},$$

i.e. the Page Rank value for a page **u** is dependent on the Page Rank values for each page **v** out of the set **B**$_{\mathbf{u}}$ (this set contains all pages linking to page **u**), divided by the number *L*(*v*) of links from page **v**.

CHAPTER : 5

TESTING

5. Testing

5.1 Testing Plan

The testing is a technique that is going to be used in the project is black box testing ,the expected inputs to the system are applied and only the outputs are checked .

5.2 Testing Strategy

The development process repeats this testing sub process a number of the lines for the following phases.

- Unit Testing
- Integration Testing

Unit Testing tests a unit of code after coding of that unit is completed. Integration Testing tests whether the previous programs that make up a system, interface with each other as desired. System testing ensures that the system meets its stated design specifications. Acceptance testing is testing by users to ascertain whether the system developed is a correct implementation of the software requirements specification.

Testing is carried out in such a hierarchical manner to that each component is correct and the assembly/combination of component is correct. Merely testing a whole system at end would most likely throw up errors in component that would be very costly to trace and fix. We have performed both Unit Testing and System Testing to detect and fix errors.

5.3 Testing Methods

We have performed Black-box testing for the testing purpose. A brief description is given below:

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.

5.4 Test Cases

Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Results**	**Actual Results**
T01	Sign Up	1. Go to Home page 2. Provide Information	User Information	Stored Successfully	Success
T02	Log In	1. Go to Home page 2. Enter Credentials	User Credentials	Display Main Page	Main Page Displayed
T03	Add Employee	1. Go to Main Page 2. Go to register employee 3. Enter	Employee Information	Employee information stored successfully in database Success	Success

		employee Details			
T04	View collection	1.Go to Views collection page 2.Enter Domain name as Project name	Collection of crawled data	Displays the data based on given project name.	Result Shown accordingly
T05	Crawl	1. Go to Crawl Page 2. Provide The project name, domain link and no pages to be crawl	Crawling start and request id generated	It Shows the request id Generated and Send email when crawling will done successfully.	Success
T06	Show details	1. Go to Show details page 2. Enter the domain name of the link	Display Output data	It will list all the crawled link on that domain and it's rank and titles also	Successfully Displayed

		as project name			
T07	Search Keyword	1. Home page 2. Enter any Keyword	Matched links listed	Display all the links related with input keyword from our database.	Displayed Successfully

CHAPTER : 6

SCREENSHOTS

6. Screen-shots

6.1 Home Page

This is the starting page of the system. Any user can access this page without login and she can search her queries here. From this page admin and employees can login to their accounts.

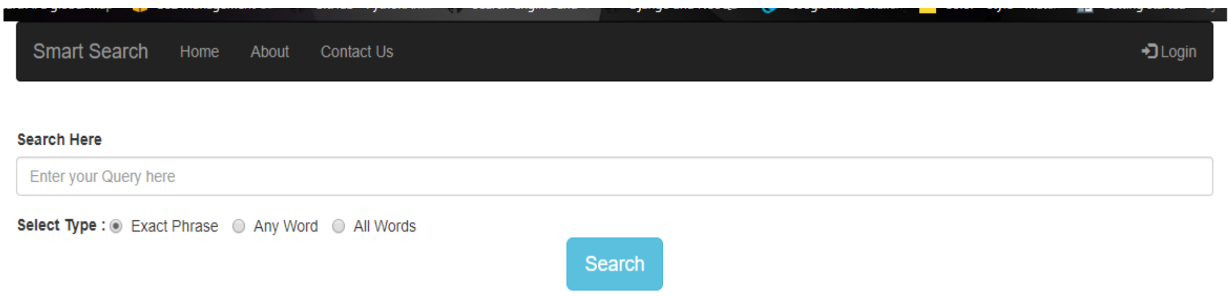
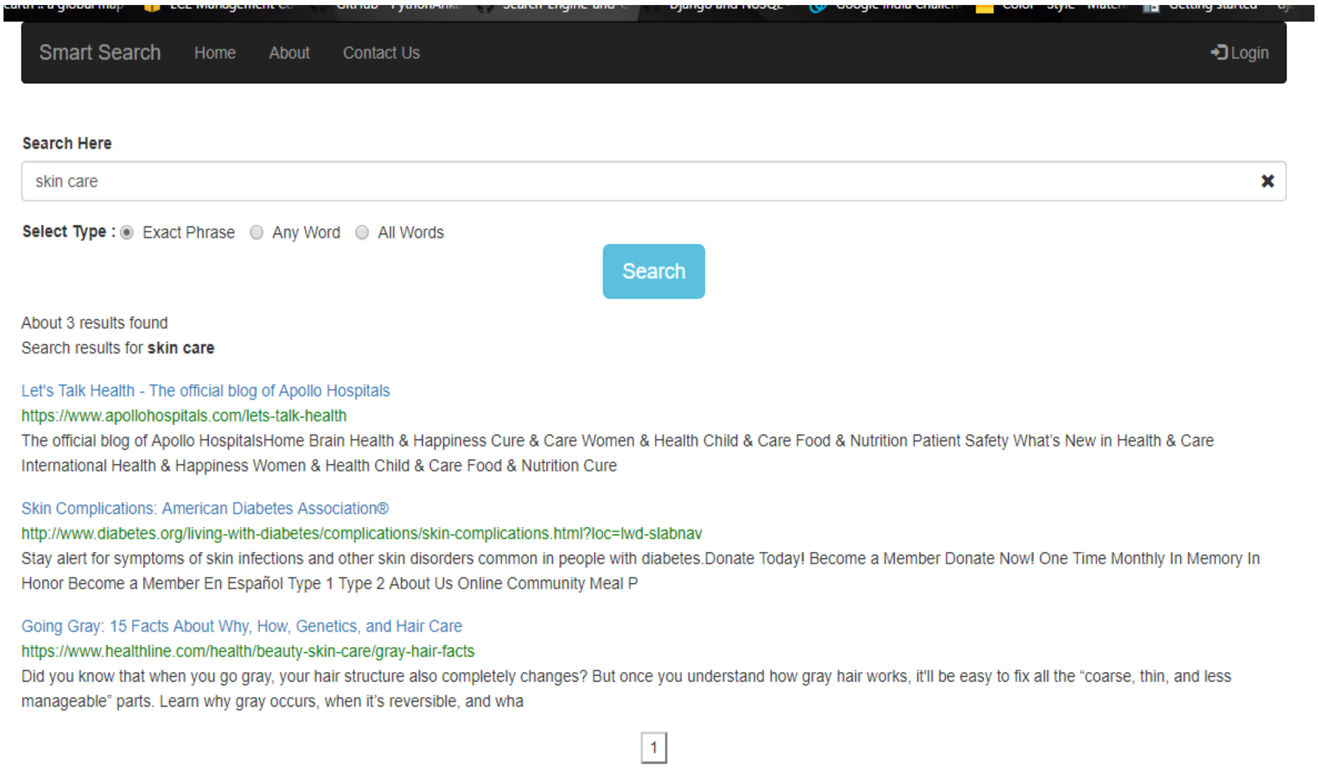


Figure 6.1.1 Search Result

After hitting any query in the search box the result of searching is displayed over here. The search result contains redirection links of the pages. Searching can be done in one of the three ways visible in the image.



Search Here

skin care

Select Type : [✓ Exact Phrase] Any Word All Words

About 3 results found
Search results for **skin care**

Let's Talk Health - The official blog of Apollo Hospitals
https://www.apollohospitals.com/lets-talk-health
The official blog of Apollo HospitalsHome Brain Health & Happiness Cure & Care Women & Health Child & Care Food & Nutrition Patient Safety What's New in Health & Care International Health & Happiness Women & Health Child & Care Food & Nutrition Cure

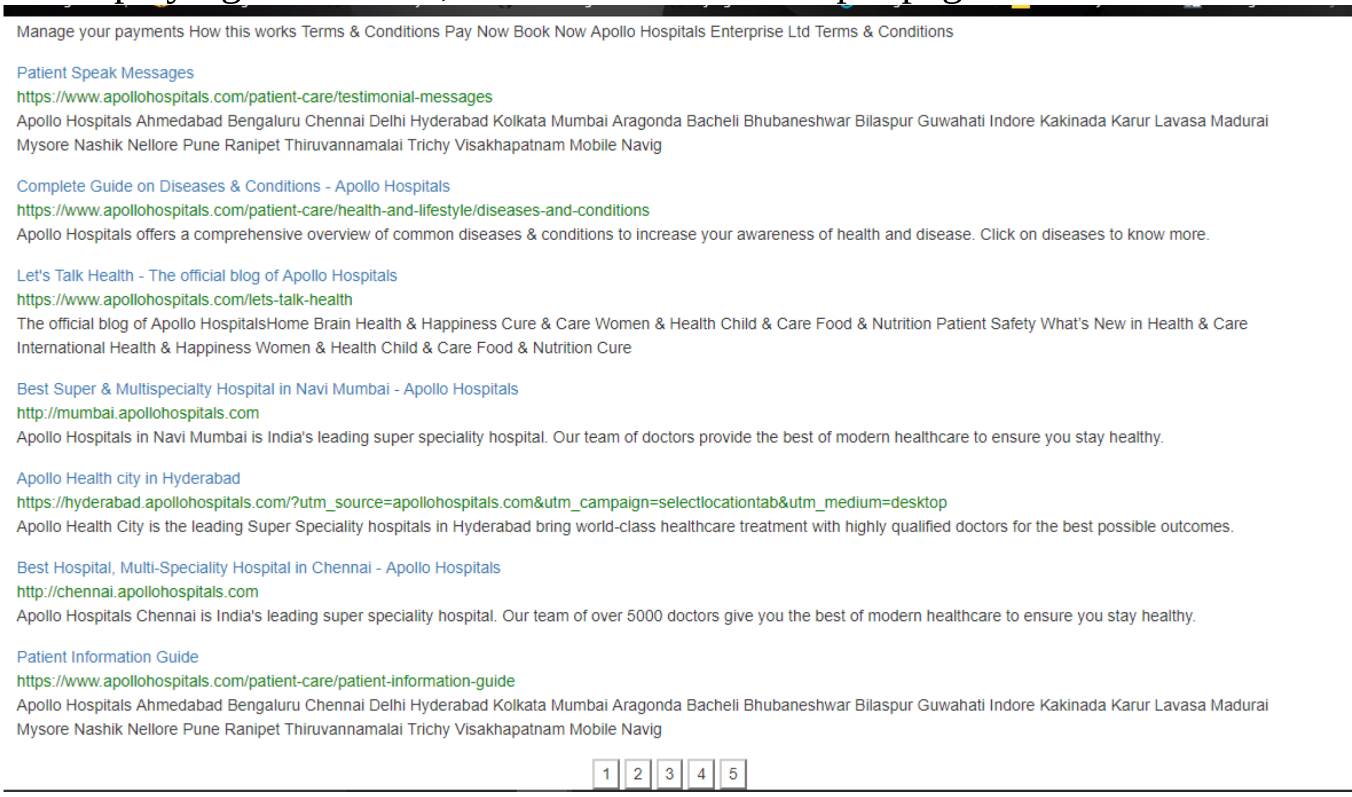
Skin Complications: American Diabetes Association®
http://www.diabetes.org/living-with-diabetes/complications/skin-complications.html?loc=lwd-slabnav
Stay alert for symptoms of skin infections and other skin disorders common in people with diabetes.Donate Today! Become a Member Donate Now! One Time Monthly In Memory In Honor Become a Member En Español Type 1 Type 2 About Us Online Community Meal P

Going Gray: 15 Facts About Why, How, Genetics, and Hair Care
https://www.healthline.com/health/beauty-skin-care/gray-hair-facts
Did you know that when you go gray, your hair structure also completely changes? But once you understand how gray hair works, it'll be easy to fix all the "coarse, thin, and less manageable" parts. Learn why gray occurs, when it's reversible, and wha

1

Figure 6.1.2 Search Result – 2(Any Word)

This figure shows the pagination links. We have used pagination for displaying the records, 10 records are shown per page.



Manage your payments How this works Terms & Conditions Pay Now Book Now Apollo Hospitals Enterprise Ltd Terms & Conditions

Patient Speak Messages
https://www.apollohospitals.com/patient-care/testimonial-messages
Apollo Hospitals Ahmedabad Bengaluru Chennai Delhi Hyderabad Kolkata Mumbai Aragonda Bacheli Bhubaneshwar Bilaspur Guwahati Indore Kakinada Karur Lavasa Madurai Mysore Nashik Nellore Pune Ranipet Thiruvannamalai Trichy Visakhapatnam Mobile Navig

Complete Guide on Diseases & Conditions - Apollo Hospitals
https://www.apollohospitals.com/patient-care/health-and-lifestyle/diseases-and-conditions
Apollo Hospitals offers a comprehensive overview of common diseases & conditions to increase your awareness of health and disease. Click on diseases to know more.

Let's Talk Health - The official blog of Apollo Hospitals
https://www.apollohospitals.com/lets-talk-health
The official blog of Apollo HospitalsHome Brain Health & Happiness Cure & Care Women & Health Child & Care Food & Nutrition Patient Safety What's New in Health & Care International Health & Happiness Women & Health Child & Care Food & Nutrition Cure

Best Super & Multispecialty Hospital in Navi Mumbai - Apollo Hospitals
http://mumbai.apollohospitals.com
Apollo Hospitals in Navi Mumbai is India's leading super speciality hospital. Our team of doctors provide the best of modern healthcare to ensure you stay healthy.

Apollo Health city in Hyderabad
https://hyderabad.apollohospitals.com/?utm_source=apollohospitals.com&utm_campaign=selectlocationtab&utm_medium=desktop
Apollo Health City is the leading Super Speciality hospitals in Hyderabad bring world-class healthcare treatment with highly qualified doctors for the best possible outcomes.

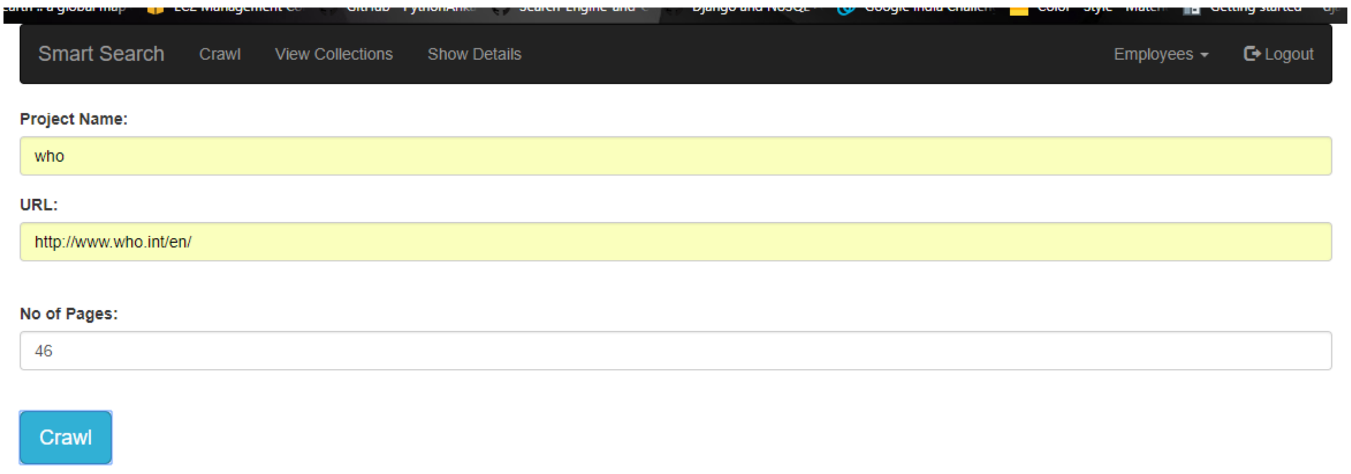
Best Hospital, Multi-Speciality Hospital in Chennai - Apollo Hospitals
http://chennai.apollohospitals.com
Apollo Hospitals Chennai is India's leading super speciality hospital. Our team of over 5000 doctors give you the best of modern healthcare to ensure you stay healthy.

Patient Information Guide
https://www.apollohospitals.com/patient-care/patient-information-guide
Apollo Hospitals Ahmedabad Bengaluru Chennai Delhi Hyderabad Kolkata Mumbai Aragonda Bacheli Bhubaneshwar Bilaspur Guwahati Indore Kakinada Karur Lavasa Madurai Mysore Nashik Nellore Pune Ranipet Thiruvannamalai Trichy Visakhapatnam Mobile Navig

6.2 Admin Crawl

This is the page where admin can enter seed url from where starting is to be crawled with number of pages to be crawled.

The crawl job submitted by admin is automatically approved.



Project Name:

who

URL:

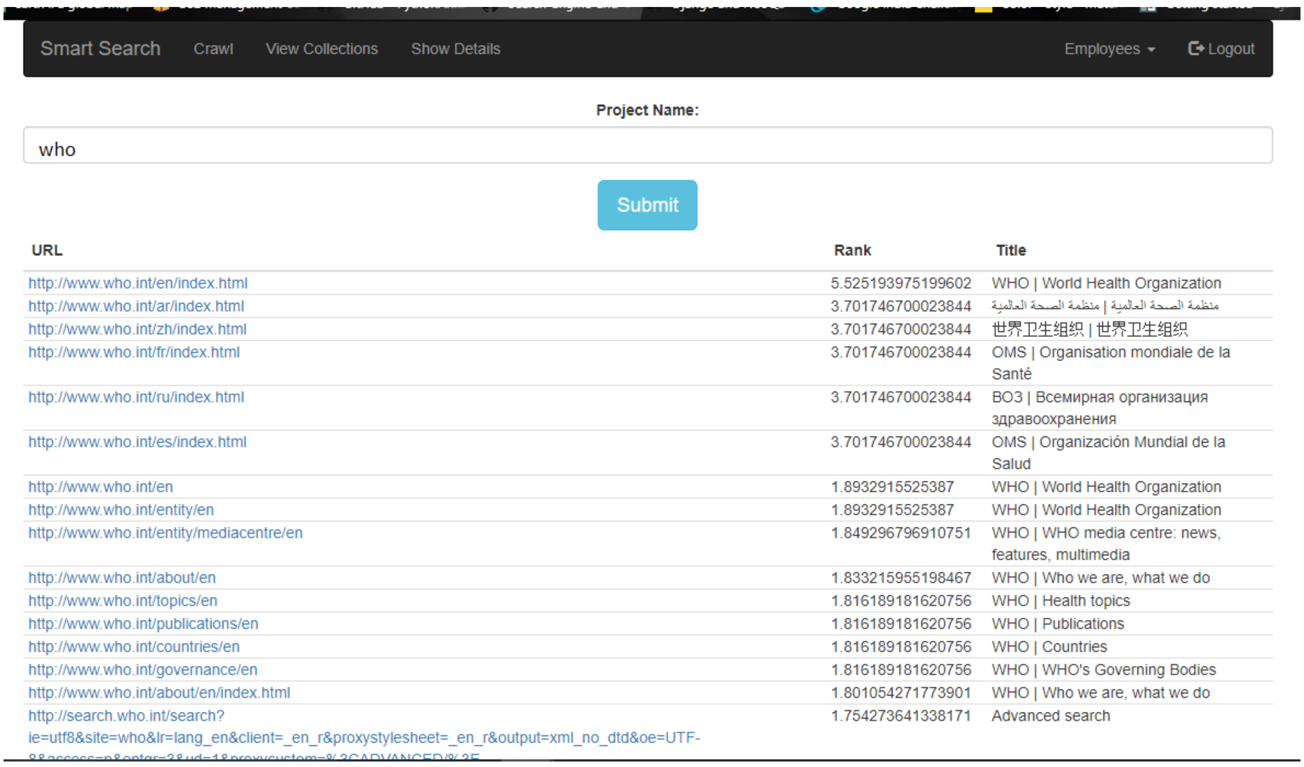
http://www.who.int/en/

No of Pages:

46

6.3 Admin View Result

Admin can view the crawling results with ranks of respective ranks counted by page rank algorithm.
Any column is sorted ascending/descending by clicking on the title of column.



Project Name:

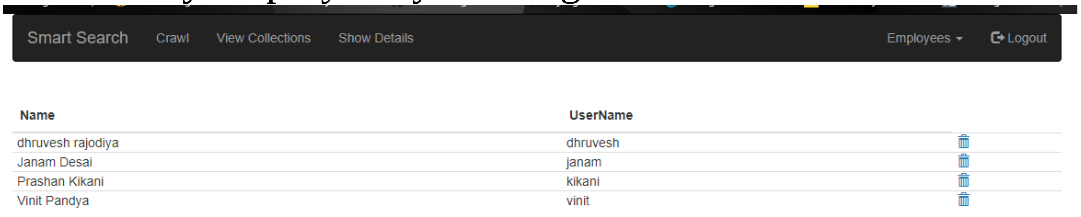
who

URL	Rank	Title
http://www.who.int/en/index.html	5.525193975199602	WHO | World Health Organization
http://www.who.int/ar/index.html	3.701746700023844	منظمة الصحة العالمية | منظمة الصحة العالمية
http://www.who.int/zh/index.html	3.701746700023844	世界卫生组织 | 世界卫生组织
http://www.who.int/fr/index.html	3.701746700023844	OMS | Organisation mondiale de la Santé
http://www.who.int/ru/index.html	3.701746700023844	ВОЗ | Всемирная организация здравоохранения
http://www.who.int/es/index.html	3.701746700023844	OMS | Organización Mundial de la Salud
http://www.who.int/en	1.8932915525387	WHO | World Health Organization
http://www.who.int/entity/en	1.8932915525387	WHO | World Health Organization
http://www.who.int/entity/mediacentre/en	1.849296796910751	WHO | WHO media centre: news, features, multimedia
http://www.who.int/about/en	1.833215955198467	WHO | Who we are, what we do
http://www.who.int/topics/en	1.816189181620756	WHO | Health topics
http://www.who.int/publications/en	1.816189181620756	WHO | Publications
http://www.who.int/countries/en	1.816189181620756	WHO | Countries
http://www.who.int/governance/en	1.816189181620756	WHO | WHO's Governing Bodies
http://www.who.int/about/en/index.html	1.801054271773901	WHO | Who we are, what we do
http://search.who.int/search? ie=utf8&site=who&lr=lang_en&client=_en_r&proxystylesheet=_en_r&output=xml_no_dtd&oe=UTF-	1.754273641338171	Advanced search

6.4 Admin Manage

6.4.1 View Employees

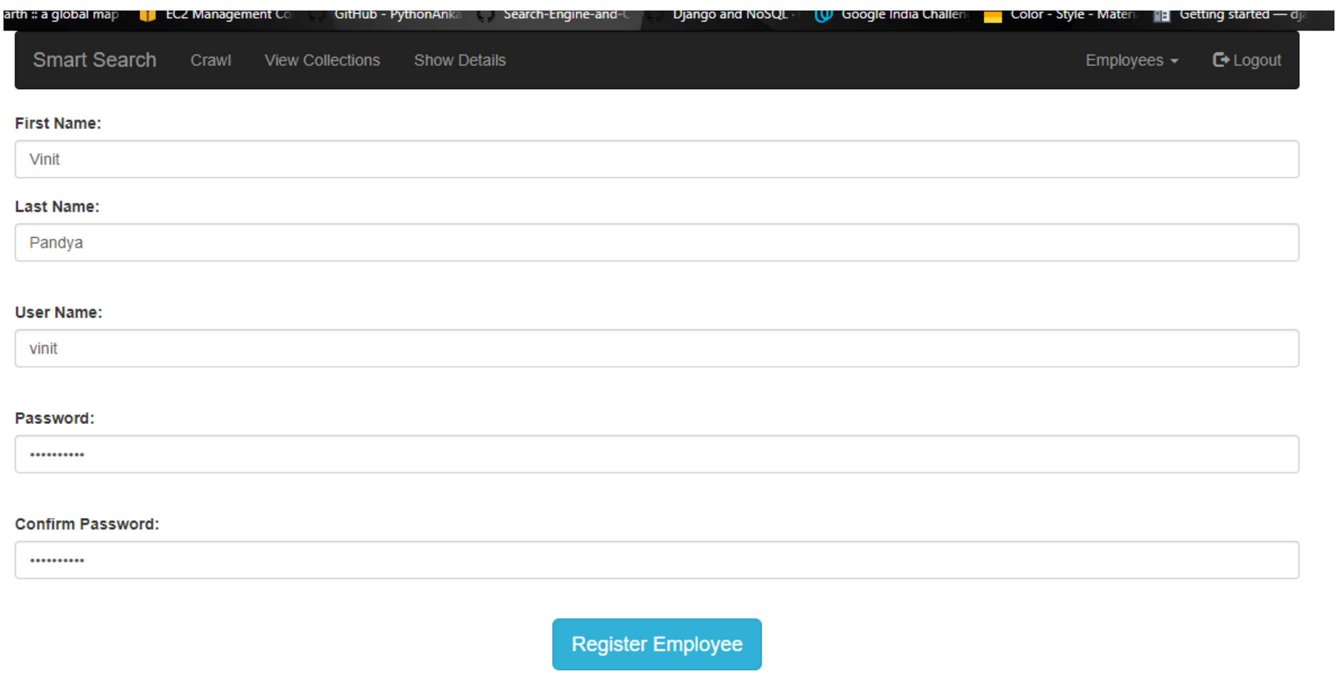
Admin can view currently registered employees here and can remove any employee by clicking on delete icon.



Name	UserName
dhruvesh rajodiya	dhruvesh
Janam Desai	janam
Prashan Kikani	kikani
Vinit Pandya	vinit

6.4.2-Add Employee

Here Admin can register any new employee.
Employee can not register by herself, it must be done by Admin.



First Name: Vinit

Last Name: Pandya

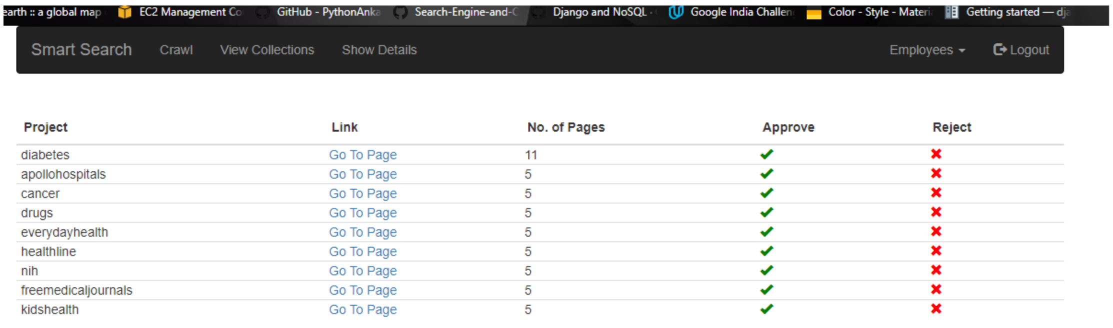
User Name: vinit

Password: ••••••••••

Confirm Password: ••••••••••

6.4.3 Approve Request

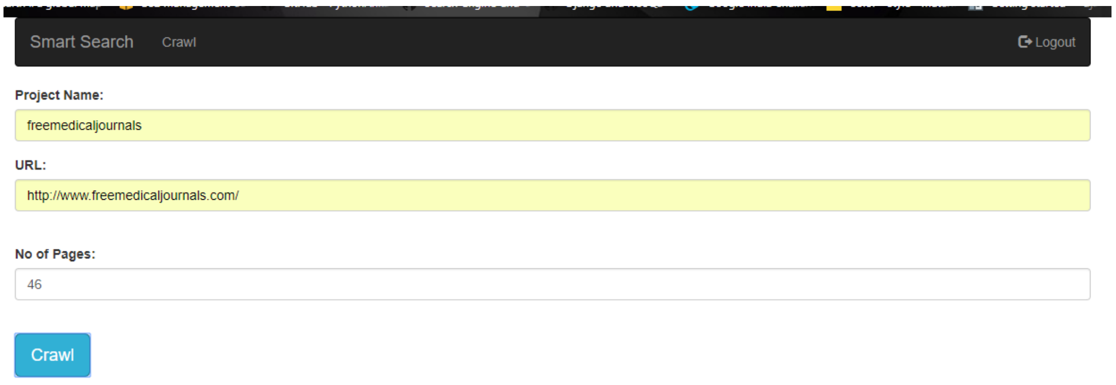
This is the page where admin can approve/reject pending requests. Admin can change No. of pages of any request by editing that column.



Project	Link	No. of Pages	Approve	Reject
diabetes	Go To Page	11	✓	✗
apollohospitals	Go To Page	5	✓	✗
cancer	Go To Page	5	✓	✗
drugs	Go To Page	5	✓	✗
everydayhealth	Go To Page	5	✓	✗
healthline	Go To Page	5	✓	✗
nih	Go To Page	5	✓	✗
freemedicaljournals	Go To Page	5	✓	✗
kidshealth	Go To Page	5	✓	✗

6.5 Request Crawl by Employee

Employee can add new crawl request by adding it over here. Request will be added with status as pending.



Project Name: freemedicaljournals

URL: http://www.freemedicaljournals.com/

No of Pages: 46

CHAPTER : 7

LIMITATIONS AND FUTURE ENHANCEMENTS

7. Limitations and Future Enhancements

7.1 Limitations

- System is not able to provide the dynamic ranking based.

- If re-crawling of any website from scratch is needed then, admin has to delete previous record manually.

7.2 Future Enhancements

- Dynamic ranking of pages based on user's interest.

- “You may also like” feature in search section.

CHAPTER : 8

CONCLUSION

8. Conclusion

Hereby, we declare that the functionality implemented in Web Crawler Supported Smart Search was performed by understanding all the modules.

We have implemented crawling with solr indexing and ranking of pages.

The searching facility is also provided with three options (Exact phrase, Any Word, All Words).

After the coding was completed, comprehensive testing was performed and the results were provided in the report. Unit Testing of all modules were done and later ,Integration Testing was also performed.

CHAPTER : 9

BIBLIOGRAPHY

9. Bibliography

9.1 Web Tutorials

- https://docs.python.org/
- https://docs.mongodb.com/
- http://lucene.apache.org/solr/
- https://www.py4e.com/
- http://flask.pocoo.org/docs/0.12/

9.2 Books

- The Def Guide to MongoDB
- MongoDB in action : Kyle Banker
- Core Python Programming
- Flask Web Development

CHAPTER : 10

SOFTWARE VERSION DEPLOYMENT DOCUMENT

10.1 Installing

➢ MongoDB

- ➢ Install MongoDB Client from
  https://www.mongodb.com/download-center
- ➢ Install pymongo package by
  pip install pymongo OR
  https://github.com/mongodb/mongo-python-driver

➢ Apache Solr

- ➢ Install Solr
  http://lucene.apache.org/solr/downloads.html
- ➢ Install pysolr package by
  pip install pysolr (or)
  https://github.com/djangohaystack/pysolr

10.2 Create Collections

- ➢ Create collection of your requirement in Mongo Compass or by Mongo Shell.

The following example shows the syntax of **createCollection()** method with few important options in Mongo Shell –

```
>db.createCollection("mycol", { capped : true, autoIndexId : true, size :
   6142800, max : 10000 } )
```

- ➢ create solr core of your requirement in command prompt by typing below command :

  bin/solr create -c <name>

10.3 Starting Server

- MongoDB :
  For mongodb start local server by typing :

  “$mongod” in command prompt.

- Solr :

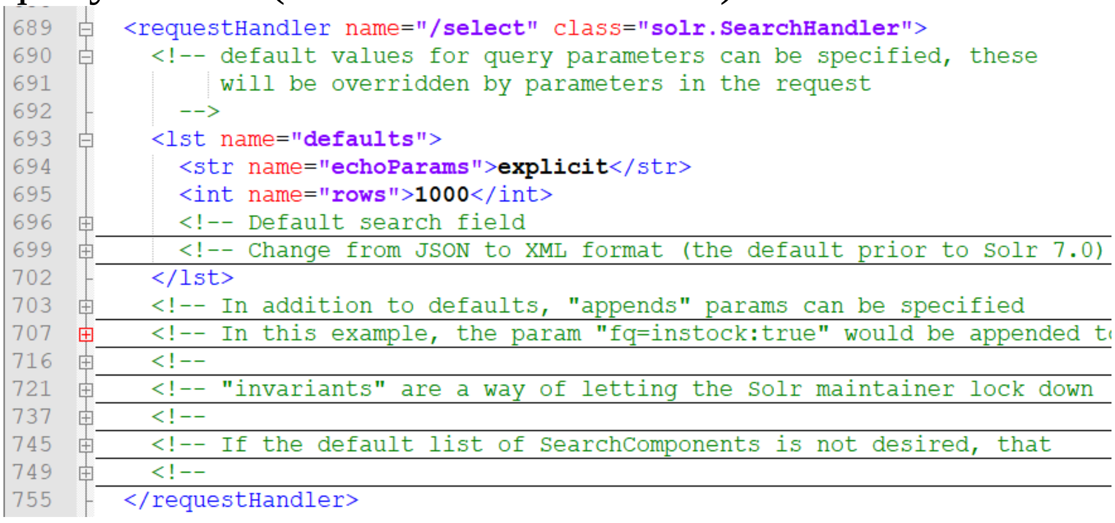
  For Windows, you can start Solr by running bin\solr.cmd .

```
bin\solr.cmd start
```

  This will start Solr in the background, listening on port 8983.

- Solr configuration :
  At location
  <ROOT-DRIVE>\solr-7.2.1\solr-7.2.1\server\solr\<CORE-NAME>\conf

  You have to specify number of results in query result : (here line number : 695)



```
689  <requestHandler name="/select" class="solr.SearchHandler">
690    <!-- default values for query parameters can be specified, these
691         will be overridden by parameters in the request
692      -->
693    <lst name="defaults">
694      <str name="echoParams">explicit</str>
695      <int name="rows">1000</int>
696      <!-- Default search field
699      <!-- Change from JSON to XML format (the default prior to Solr 7.0)
702    </lst>
703    <!-- In addition to defaults, "appends" params can be specified
707    <!-- In this example, the param "fq=instock:true" would be appended t
716    <!--
721    <!-- "invariants" are a way of letting the Solr maintainer lock down
737    <!--
745    <!-- If the default list of SearchComponents is not desired, that
749    <!--
755  </requestHandler>
```

CHAPTER : 11

MISCELLANEOUS

- **Challenge** :
  In Search result we were always getting only 10 rows as output no matter what the keyword was, we were not sure about how to solve this problem.

  Solution :
  We have to configure "solrconfig.xml" file's RequestHandler tag to give output as 1000 rows which is sufficient in our project.

- **Challenge:**
  In submitting job of crawling we wanted to do asynchronous function call in python so that user need not to wait for crawling has completed.

  Solution:
  We have not implemented asynchronous function call because that method will work only for one caller method which needed to be the main method of program, which was not supported in our program so what we have done is that whenever any new request arrives store it to the database in waiting queue and an another process is fetching requests from that queue.

- **Challenge:**
  At beginning we have thought of adding 3 filters in search module, but at implementation time we could not implement "Exact Phrase"

filter because we could not find query syntax for that.

Solution:
We were writing wrong syntax at beginning but at the end we found correct syntax so we have added that feature in version-1 itself.