# NWHoogle

## Workshop 2 Project

Jay Ning[1] Wendy Wang[2] Beatrice Hou[3]

UIC Data Science

2022

# Table of Contents

## Target parsing

https://www.gutenberg.org/cache/epub/ **10000** /pg **10000** .txt

```
The Project Gutenberg eBook of The Magna Carta

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this eBook or online at
www.gutenberg.org. If you are not located in the United States, you
will have to check the laws of the country where you are located before
using this eBook.

Title: The Magna Carta

Author: Anonymous

Release Date: October 15, 2003 [eBook #10000]
[Most recently updated: December 20, 2021]

Language: English


Produced by: Michael Hart

*** START OF THE PROJECT GUTENBERG EBOOK THE MAGNA CARTA ***



The Magna Carta


Contents

 The Text of Magna Carta
 Magna Carta 1215
 The text of THE MAGNA CARTA



A note from Michael Hart, preparer of the 0.1 version.


This file contains a number of versions of the Magna Carta, some of
which were a little mangled in transit. I am sure our volunteers will
find and correct errors I didn't catch, and that version 0.2 - 1.0 will
have significant improvements, as well as at least one more version in
Latin.
```

# Data acquisition and processing

## Crawl data

Use requests and define function to get book files

```python
def storageToLocalFiles(storagePath, data):#store in the path
    fhandle = open(storagePath,"w")
    fhandle.write(data)
    fhandle.close()
```

```python
import requests
headers={'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.87 Safari/537.36'}
for i in range(10044,11001):#It should larger than 200MB
    storagePath = 'books/'+str(i)+".txt"
    link='https://www.gutenberg.org/cache/epub/'+str(i)+'/pg'+str(i)+'.txt'
    r=requests.get(link,headers=headers)
    storageToLocalFiles(storagePath,r.content)
```

## Observed Data

Found that some books do not have TXT format, the content is HTML. It is characterized as beginning with <!DOCTYPE html>

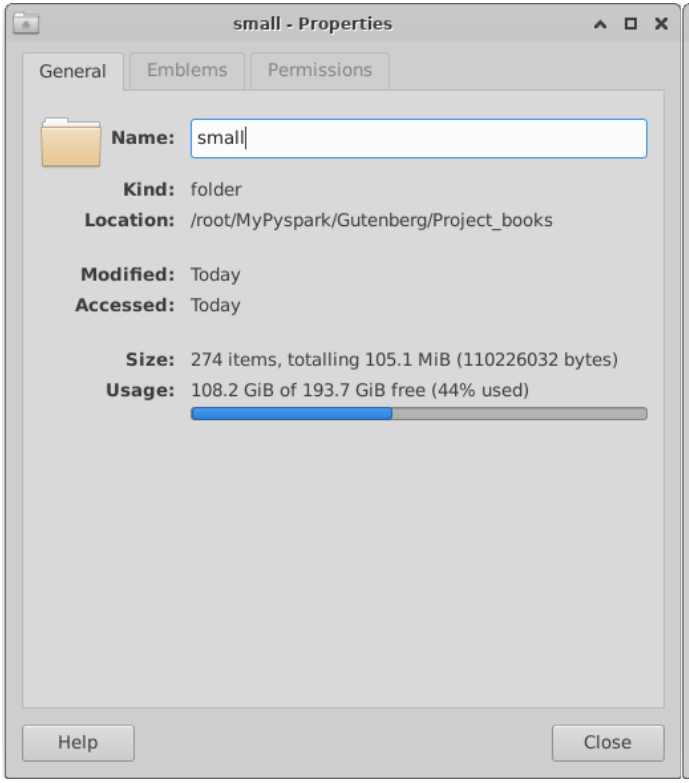
```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
 <meta charset="UTF-8"/>

<title>404 | Project Gutenberg</title>
 <link rel="stylesheet" href="/gutenberg/style.css?v=1.1">
 <link rel="stylesheet" href="/gutenberg/collapsible.css?1.1">
 <link rel="stylesheet" href="/gutenberg/new_nav.css?v=1.321231">
<link rel="stylesheet" href="/gutenberg/pg-desktop-one.css">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <meta name="keywords" content="books, ebooks, free, kindle, android, iphone, ipad"/>
 <meta name="google-site-verification" content="wucOEvSnj5kP3Ts_36OfP64laakK-1mVTg-ptrGC9io"/>
 <meta name="alexaVerifyID" content="4WNaCljsE-A82vP_ih2H_UqXZvM"/>
 <link rel="copyright" href="https://www.gnu.org/copyleft/fdl.html"/>
 <link rel="shortcut icon" href="/gutenberg/favicon.ico?v=1.1"/>

 <meta property="og:title"        content="Project Gutenberg" />
 <meta property="og:type"         content="website" />
 <meta property="og:url"          content="https://www.gutenberg.org/" />
 <meta property="og:description"  content="Project Gutenberg is a library of free eBooks." />
 <meta property="fb:admins"       content="615269807" />
 <meta property="fb:app_id"       content="115319388529183" />
 <meta property="og:site_name"    content="Project Gutenberg" />
 <meta property="og:image"        content="https://www.gutenberg.org/gutenberg/pg-logo-144x144.png" />
 </head>
<body>
  <div class="container"><!-- start body --><nav>
  <!--<div id="main_logo"> -->
  <a id="main_logo" href="/" class="no-hover">
```

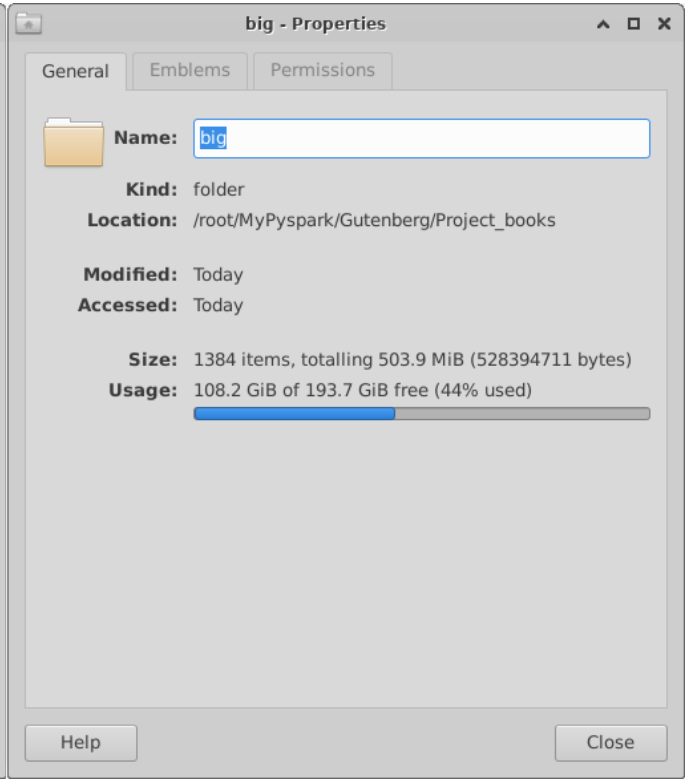
Use readline() function to check first line. And use os module to delete wrong file

```
import os
path1 = "books"
files= os.listdir(path1)
d_list=[]
for file in files:
    f = open("books/"+file,"r",encoding='utf-8')
    if(f.readline()=="<!DOCTYPE html>\n"):
        d_list.append(file)
# f.readline()
for file in d_list:
    os.remove("books/"+file)
```

# Data acquisition and processing

## Result



## Upload Files

Firstly we divided the books obtained by the crawler into two parts: 100 megabytes and 500 megabytes, and upload them to 2 HDFS folders respectively (project small and project big) then check the file in the Hadoop browse directory.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ☐ | drwxr-xr-x | root | supergroup | 0 B | May 22 11:11 | 0 | 0 B | project_big | 🗑 |
| ☐ | drwxr-xr-x | root | supergroup | 0 B | May 21 19:06 | 0 | 0 B | project_small | 🗑 |

# Indexing and Ranking

## Preprocessing

In the preprocessing part, we delete all punctuation, numbers and other symbols, only retain English characters. In addition, we quoted a text processing toolkit called nltk to delete the common stopwords, and restore the part of speech of English words with deformation.

```
import re
import math
def removeSomeTexts(text):
    return re.sub(r"[^a-z-'\s]",'', text)
stop_word = stopwords.words("english")
new_words=["aa","aaa","aaaahh"]
for i in new_words:
    stop_word.append(i)

def lemmatization(text):
    lemmatizer = WordNetLemmatizer()
    return lemmatizer.lemmatize(text)
def stem(text):
    lancaster_stemmer = LancasterStemmer()
    return lancaster_stemmer.stem(text)
```

## Use the local session

```
from pyspark import SparkConf, SparkContext
import os, re
from pyspark.sql.session import SparkSession
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
import os, re
import time

spark = SparkSession.builder.master('local').appName('timetest').getOrCreate()

sc = spark.sparkContext
#sc = SparkContext.getOrCreate(SparkConf())
sc._conf.set('spark.executor.memory', '8g')
sc._conf.set('spark.driver.memory', '8g')
sc._conf.set('spark.driver.maxResultsSize', '0')
```

Use the local session to test the time when doing the indexing part with small document (105M)

```
#Use local session to test small document
start = time.time()
data = sc.wholeTextFiles('hdfs://10.20.0.151:9000/project_small')
numFiles = data.count()
wordcount = data.flatMap(lambda x: [((os.path.basename(x[0]), removeSomeTexts(i.lower())), 1) for i in re.split('\W+', x[1])]) \
        .filter(lambda x: x[0][1] not in stop_word).filter(lambda x: x[0][1] !='').map(lambda x:((x[0][0],lemmatization(x[0][1])),x[1])) \
        .reduceByKey(lambda a,b:a+b).map(lambda x:(x[0][1],(x[0][0],x[1]))) \
        .reduceByKey(lambda a,b:a+b).sortByKey()
#wordcount.take(10)
wordcount.saveAsTextFile('hdfs://10.20.0.151:9000/output/InvIndex_small')
end = time.time()
print(end-start, " seconds")

[Stage 3:>                                                          (0 + 1) / 1]

168.81149792671204  seconds
```

168.81149792671204 seconds

Use the local session to test the time when doing the indexing part with big document (503M)

```
#Use local session to test big document
start = time.time()
data = sc.wholeTextFiles('hdfs://10.20.0.151:9000/project_big')
numFiles = data.count()
wordcount = data.flatMap(lambda x: [((os.path.basename(x[0]), removeSomeTexts(i.lower())), 1) for i in re.split('\W+', x[1])]) \
        .filter(lambda x: x[0][1] not in stop_word).filter(lambda x: x[0][1] !='').map(lambda x:((x[0][0],lemmatization(x[0][1])),x[1])) \
        .reduceByKey(lambda a,b:a+b).map(lambda x:(x[0][1],(x[0][0],x[1]))) \
        .reduceByKey(lambda a,b:a+b).sortByKey()
#wordcount.take(10)
wordcount.saveAsTextFile('hdfs://10.20.0.151:9000/output/InvIndex_big')
end = time.time()
print(end-start, " seconds")
```

```
[Stage 7:>                                                          (0 + 1) / 1]

814.2438313961029  seconds
```

814.2438313961029 seconds

Use the cluster session

```
from pyspark import SparkConf, SparkContext
import os, re
import nltk
from pyspark.sql.session import SparkSession
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
import os, re
import time
spark1 = SparkSession.builder.master('spark://DPWII-JayNing:7077').appName('MyInvIndex').getOrCreate()
sc1 = spark.sparkContext
sc1._conf.set('spark.executor.memory', '8g')
sc1._conf.set('spark.driver.memory', '8g')
sc1._conf.set('spark.driver.maxResultsSize', '0')
```

Use the cluster session to test the time when doing the indexing part with small document(105M)

```
#Use cluster session to test small document
start = time.time()
data = sc1.wholeTextFiles('hdfs://10.20.0.151:9000/project_small')
numFiles = data.count()
wordcount = data.flatMap(lambda x: [((os.path.basename(x[0]), removeSomeTexts(i.lower())), 1) for i in re.split('\W+', x[1])]) \
        .filter(lambda x: x[0][1] not in stop_word).filter(lambda x: x[0][1] !='').map(lambda x:((x[0][0],lemmatization(x[0][1])),x[1])) \
        .reduceByKey(lambda a,b:a+b).map(lambda x:(x[0][1],(x[0][0],x[1]))) \
        .reduceByKey(lambda a,b:a+b).sortByKey()
#wordcount.take(10)
wordcount.saveAsTextFile('hdfs://10.20.0.151:9000/output/InvIndex_small_cluster')
end = time.time()
print(end-start, " seconds")
```

```
80.18091893196106  seconds
```

80.18091893196106 seconds

Use the cluster session to test the time when doing the indexing part with big document(503M)

```
#Use local session to test big document
start = time.time()
data = sc1.wholeTextFiles('hdfs://10.20.0.151:9000/project_big')
numFiles = data.count()
wordcount = data.flatMap(lambda x: [((os.path.basename(x[0]), removeSomeTexts(i.lower())), 1) for i in re.split('\W+', x[1])]) \
        .filter(lambda x: x[0][1] not in stop_word).filter(lambda x: x[0][1] !='').map(lambda x:((x[0][0],lemmatization(x[0][1])),x[1])
        .reduceByKey(lambda a,b:a+b).map(lambda x:(x[0][1],(x[0][0],x[1]))) \
        .reduceByKey(lambda a,b:a+b).sortByKey()
#wordcount.take(10)
wordcount.saveAsTextFile('hdfs://10.20.0.151:9000/output/InvIndex_big_cluster1')
end = time.time()
print(end-start, " seconds")
```

```
[Stage 51:============================>                          (1 + 1) / 2]

439.3812403678894  seconds
```

439.3812403678894 seconds

Summary:

| Session/FileSize | small | big |
|---|---|---|
| local | 168 seconds | 814 seconds |
| cluster | 80 seconds | 439 seconds |

# Indexing and Ranking

## Index

In addition to the file index, we also set another index for the line of each word in each file

```
#add line index
data = sc.wholeTextFiles('hdfs://10.20.0.151:9000/project_small')
numFiles = data.count()
wordcount = data.flatMap(lambda x:[(os.path.basename(x[0]),i.lower())for i in re.split('\\n',x[1])])
wordcount1=wordcount.zipWithIndex()
wordcount3=wordcount1.flatMap(lambda x:[((x[0][0],x[1]),removeSomeTexts(i.lower()))for i in re.split('\W+',x[0][1])]) \
        .filter(lambda x: x[1] !='').filter(lambda x: x[1] not in stop_word).map(lambda x:((x[0][0],lemmatization(x[1])),x[0][1])) \
        .map(lambda x:(x[0][0],(x[0][1],x[1])))
wordcount3.take(15)
```

We first calculate the RDD which have the document id along with its document term frequency for each token

```
tf1 = wordcount.reduceByKey(lambda a,b:a+b).map(lambda x:(x[0][1],(x[0][0],x[1])))
#tf2 = tf1.map(lambda x:(x[0],(x[1][0],1+math.log10(x[1][1]))))
tf = tf1.map(lambda x:(x[0],(x[1][0], 1+math.log10((x[1][1]) ))))
tf.take(10)
```

Another one is the RDD which has the IDF score

```
idf1 = tf1.map(lambda x:(x[0],1)).reduceByKey(lambda x,y:x+y)
idf = idf1.map(lambda x:(x[0],math.log10(numFiles / (x[1]))))
idf.take(10)
```

Perform an inner join to assign each token

```
tfidf1 = tf.join(idf)
tfidf = tfidf1.map(lambda x: (x[1][0][0], (x[0], x[1][0][1] * x[1][1]))).sortByKey()
index = tfidf.map(lambda x:(x[1][0], (x[0], x[1][1]))).filter(lambda x: x[0]!='')
index = index.sortByKey()
index.take(10)
```

Save lines and tfidf to MySQL

```
user = 'root'
pw = '12345678'
## Database information
table_name = 'books'
url = 'jdbc:mysql://localhost:3306/project?user='+user+'&password='+pw
properties ={ 'password': pw,'user': user}
lines.write.jdbc(url=url, table=table_name, mode='append', properties=properties)
```

Check if the tables with row indexes and TFIDF indexes have been imported into SQL workbench respectively.

```
1 •  SELECT * FROM project.line_index;
```

t Grid  Filter Rows:  Expor

| term | fileName | rowNum |
|---|---|---|
| project | 10000.txt | 0 |
| gutenberg | 10000.txt | 0 |
| ebook | 10000.txt | 0 |
| magna | 10000.txt | 0 |
| carta | 10000.txt | 0 |
| ebook | 10000.txt | 2 |
| use | 10000.txt | 2 |
| anyone | 10000.txt | 2 |
| anywhere | 10000.txt | 2 |
| united | 10000.txt | 2 |
| state | 10000.txt | 2 |
| part | 10000.txt | 3 |
| world | 10000.txt | 3 |

```
1 •  SELECT * FROM project.books
```

t Grid  Filter Rows:

| term | fileName | TFIDF |
|---|---|---|
| abashed | 10008.txt | 0.9705248794454127 |
| abashed | 10017.txt | 0.7459665670122424 |
| abashed | 10038.txt | 0.7459665670122424 |
| abashed | 10041.txt | 0.7459665670122424 |
| abashed | 10047.txt | 0.7459665670122424 |
| abashed | 10048.txt | 0.7459665670122424 |
| abashed | 10057.txt | 0.7459665670122424 |
| abashed | 10062.txt | 1.195083191878583 |
| abashed | 10064.txt | 1.4919331340244848 |
| abashed | 10068.txt | 0.7459665670122424 |
| abashed | 10083.txt | 0.9705248794454127 |
| abashed | 10094.txt | 0.7459665670122424 |
| abashed | 10095.txt | 1.195083191878583 |

# Indexing and Ranking

## Table Join

To speed up the query for corresponding terms, we use natural joins to automatically match the records of columns with the same name in both tables and merge the two tables.

```
CREATE TABLE tfitfidfdf AS SELECT * FROM project.books NATURAL JOIN project.line_index;
```

Filter Rows: Edit: Export/Import: Wrap Cell Content: Fetch rows:

| term | fileName | TFIDF | rowNum |
|---|---|---|---|
| abbot | 10000.txt | 2.094431030211474 | 59 |
| abbot | 10000.txt | 2.094431030211474 | 180 |
| abbey | 10000.txt | 1.564524419258904 | 344 |
| abbot | 10000.txt | 2.094431030211474 | 346 |
| abolished | 10000.txt | 0.993710655438553 | 356 |
| abandon | 10000.txt | 0.538086675634899 | 385 |
| abbey | 10000.txt | 1.564524419258904 | 392 |
| abandon | 10000.txt | 0.538086675634899 | 394 |
| abandon | 10000.txt | 0.538086675634899 | 426 |
| abbot | 10000.txt | 2.094431030211474 | 540 |
| able | 10000.txt | 0.0822909851809811 | 630 |
| abbot | 10000.txt | 2.094431030211474 | 665 |
| abbey | 10000.txt | 1.564524419258904 | 835 |
| abolished | 10000.txt | 0.993710655438553 | 848 |
| abbey | 10000.txt | 1.564524419258904 | 886 |
| abbot | 10000.txt | 2.094431030211474 | 1034 |
| able | 10000.txt | 0.0822909851809811 | 1123 |

Add a self-incrementing ID to the new table and import the table into the booktfidf table created in Django

```
INSERT INTO project.booktfidf SELECT * FROM project.tfidf;
SELECT * FROM project.booktfidf;
```

t Grid  Filter Rows:  Edit:  Export/Import:  W

| id | term | fileName | TFIDF | rowNum |
|---|---|---|---|---|
| 1 | abbot | 10000.txt | 2.094431030211474 | 59 |
| 2 | abbot | 10000.txt | 2.094431030211474 | 180 |
| 3 | abbey | 10000.txt | 1.564524419258904 | 344 |
| 4 | abbot | 10000.txt | 2.094431030211474 | 346 |
| 5 | abolished | 10000.txt | 0.993710655438553 | 356 |
| 6 | abandon | 10000.txt | 0.538086675634899 | 385 |
| 7 | abbey | 10000.txt | 1.564524419258904 | 392 |
| 8 | abandon | 10000.txt | 0.538086675634899 | 394 |
| 9 | abandon | 10000.txt | 0.538086675634899 | 426 |
| 10 | abbot | 10000.txt | 2.094431030211474 | 540 |

In Django sever we define the model with unique_index:

```
class BookTFIDF(models.Model):
    term = models.CharField(max_length = 40)
    fileName = models.CharField(max_length = 10)
    TFIDF = models.FloatField()
    #rowNum = models.IntegerField()
    class Meta:
        #unique_together = ["term", "fileName"]
        managed = True
        db_table = 'booktfidf'
        constraints = [models.UniqueConstraint(fields=["term", "fileName"], name = 'unique_token')]
```

Which make a index in MySQL session to speed up search:

**Indexes in Table**

| Visibl | Key | Type | Uniq | Columns |
|---|---|---|---|---|
| ☑ | PRIMARY | BTREE | YES | id |
| ☑ | unique_token | BTREE | YES | term, fileName |

**Index Details**

Key Name: **unique_token**
Index Type: **BTREE**
Allows NULL:
Cardinality: **205530**
Comment:
User Comment:
Packed:
Unique: **YES**

**Columns in table**

| Column | Type | Nullal | Indexes |
|---|---|---|---|
| id | bigint | NO | PRIMARY |
| term | varchar(40) | NO | unique_token |
| fileName | varchar(10) | NO | unique_token |
| TFIDF | double | NO | |

# Server and UI

## Search page

## index Function

```
def index(request):
    context={'results':{},'count':0,'search_term':'empty'}
    # get keyword of searching.
    if request.method == "POST":
        # just submit the result to the result url.
        return sql_result(request)
    else:
        #If it is get, just stay in this page.
        return render(request, 'minisearch/header.html', context)
```

## sql_result Function

```python
def sql_result(request):
    # spilt the result, and find the preview in hdfs.
    #time counter and variable initialization.
    start_time = time.time()
    results = []    # store the filename.
    search_term = ''    # store the term that user search.
    context = dict()    # used to render html.
    # read in the index and filename related to input.
    if request.method == "POST":
        search_term=request.POST['keyword']
        # call function in the search_call.py at current path.
        results=sql_search(query=search_term)
        results_list = list(results[1])
        context={'results':results_list,'count':results[0],'search_term':search_term}
        # acquire filename, TFIDF, rowNum and preview of text.
        context['fileName'] = [list(results_list[i])[0] for i in range(results[0])]
        context['TFIDF'] = [list(results_list[i])[1] for i in range(results[0])]
        # context['rowNum'] = [list(results_list[i])[2] for i in range(results[0])]
        # get the english meaning.
        context['title1'] = "English Explantion"
        context['explanation'] = get_meaning(search_term)
        # get chinese translation.
        context['title2'] = "Synonyms"
        context['synonyms']= get_synonyms(search_term)

        # get final result of the html page
        context['final'] = ""
        count = 1
        fileName = list(context['fileName'])
```

# Server and UI

## Link to UI

```
        for i in range(results[0]):
            # get final results, with a special form.
            # we can preview 3 lines besides the article.
            context['final'] += "<table><th><tr>"
            content,bookName = get_finalRes(fileName[i])
            weblink = "<a href=http://localhost:8000/static/tfidf-index/"+fileName[i]+">"
            context['final'] += weblink
            context['final'] += "<h5>"
            context['final'] += str(count)
            context['final'] += ". "
            context['final'] += bookName
            context['final'] += "</h5></a></tr>"
            context['final'] += content
            context['final'] += "</h6></th></table>"
            context['final'] += '<br>'
            count+=1

        # use amount to store the num of true term in the result.
        # maybe the num of result < 20.
        context['amount'] = count - 1
        # end of time calculation, add time into context dictionary.
        end_time = time.time()
        context['time'] = str(round(end_time-start_time,2))
    return render(request, 'minisearch/result.html', context)
```

## sql_search Function

```
# here we use mysql statement to select the index.
def sql_search(query):
    with connection.cursor() as cursor:
        cursor.execute("SELECT * FROM (SELECT filename, TFIDF FROM booktfidf WHERE term = '" + str(query) + "'" + ") AS a ORDER BY TFIDF DESC LIMIT 2
        row = cursor.fetchall()
        num = len(row)
    return num, row
```

Here, we connect to MySQL database to fetch index, including file name, value of TFIDF which correspond to search term, and we set only take 20 of result which TFIDF is large. Pass number of result and data in database to sql_result function.

## get_finalRes Function

```
def get_finalRes(filename):
    bookName=''
    fileFetcher = Process_Data_Hdfs(filename)
    content = ""
    local_path = "/root/MyPyspark/Gutenberg/Django/dpwII_project/static/tfidf-index/"
    if(os.path.exists(local_path + filename) == False):
        fileFetcher.get_from_hdfs(local_path)

    # after download the file, we open it to fetch info.
    with open(local_path + filename) as f:
            for lines in f.readlines():
                # get the title of book.
                if 'Title: ' in lines:
                    bookName=lines[7:]
                if 'Author: 'in lines:
                    print(lines)
                    content += '<tr><h6>'
                    content += lines
                    content += '</h6></tr>'
                    break
    return content,bookName
```

L.

## Process_Data_Hdfs Class

```python
# define a class used to fetch files from hdfs.
class Process_Data_Hdfs():
    # initialize an object.
    def __init__(self, filename):
        self.client = Client('http://localhost:9870')
        self.filename = "/project_big/" + filename

    # download file from hdfs.
    def get_from_hdfs(self, local_path):
        self.client.download(self.filename, local_path, overwrite=False)
        return

    # return file under directory on hdfs.
    def list(self):
        return self.client.list(self.filename, status=False)
```

## get_meaning & get_synonyms Function

```python
#get the english meanings of the keywords.
def get_meaning(post_data):
    # which means the input is a phrase,
    # simply show a warning.
    if(len(post_data.split())>1):
        return "Sorry, A phrase is not supported to get the English explanation for now!"
    # otherwise, create a pydictionary object.
    dictionary = PyDictionary()
    dict = dictionary.meaning(post_data)
    try:
        # try to get the lenth of the dictionary.
        length = len(dict[list(dict.keys())[0]])
    except:
        # which means there is no explanation in the pydictionary.
        return "The keyword you input has no explanation in the dictionary!"
    # if try successful, then get the meaning of keywords.
    value = list(dict.values())[0]
    string = ""
    count = 1
    for i in range(length):
        if count > 5:
            break
        # output all meanings of keywords.
        string += str(i+1)
        string += ". "
        string += value[i]
        # a new line.
        string += "<br>"
        count += 1
```

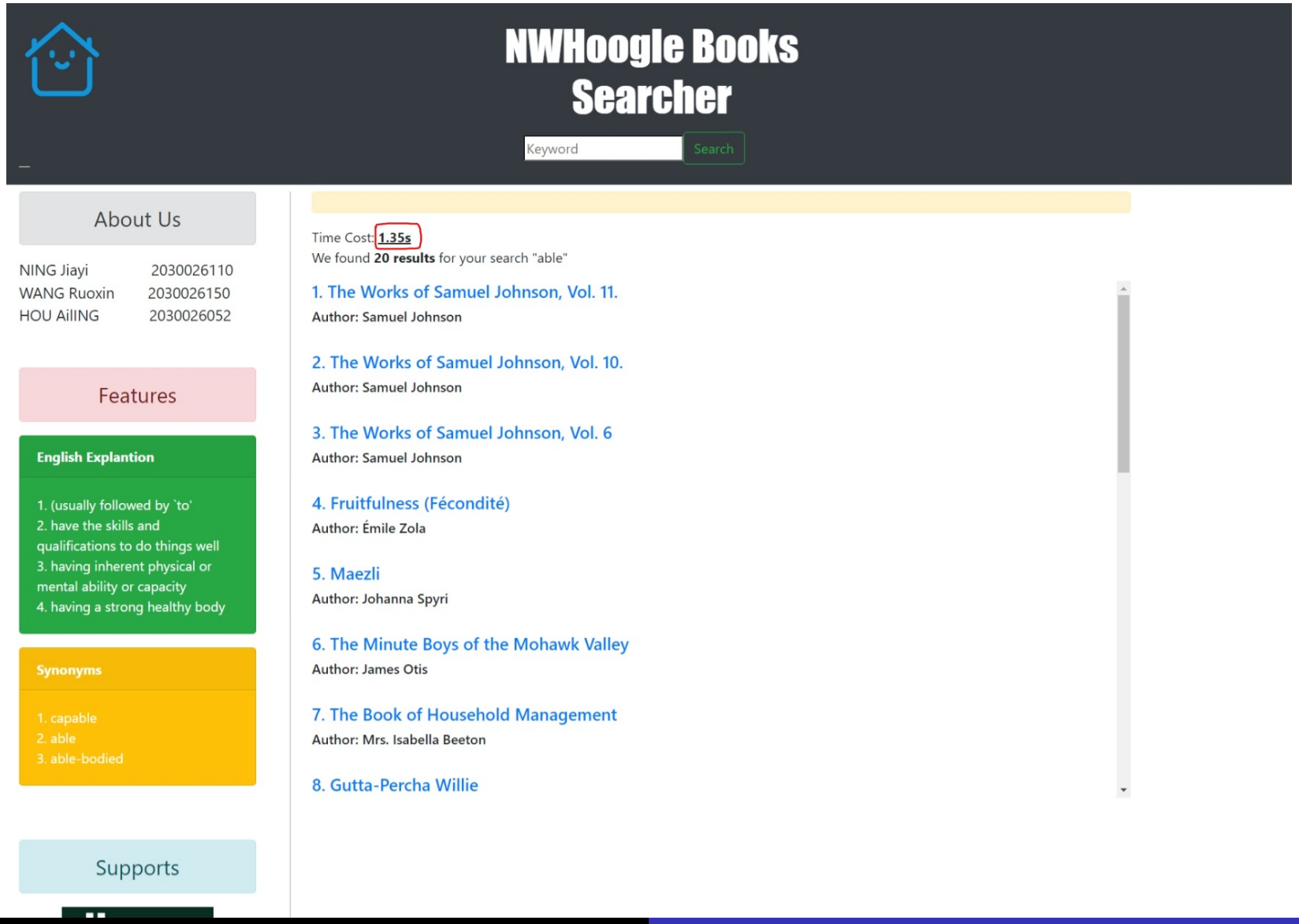
# Server and UI

## Link to UI

```python
#get the synonyms of the keywords.
def get_synonyms(post_data):
    synonyms = []
    try:
        for syn in wordnet.synsets(post_data):
            for l in syn.lemmas():
                synonyms.append(l.name())
    except:
        return "The keyword you input has no synonyms in the dictionary!"
    if(len(synonyms)==0):
        return "The keyword you input has no synonyms in the dictionary!"
    str1 = ""
    count = 1
    synonyms = list(set(synonyms))
    for i in synonyms:
        if count > 5:
            break
        str1 += (str(count)+". "+i+"<br>")
        count += 1
    return str1
```

Call methods in PyDictionary package, to get further information and explanation of search term. We only select top 5 item in package. And pass them in specific form combining with HTML.

## search result



## Preview Book

```
The Project Gutenberg EBook of The Poetical Works of William Wordsworth
Edited by William Knight

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: The Poetical Works of William Wordsworth
       Volume 1 of 8

Author: (Edited by William Knight)

Release Date: November 23, 2003 [EBook #10219]

Language: English


*** START OF THIS PROJECT GUTENBERG EBOOK POETRY OF WORDSWORTH ***



Produced by Jonathan Ingram, Clytie Siddall and the Online Distributed
Proofreading Team.
```

## Meaning and Synonyms

Features

**English Explantion**

1. (usually followed by `to'
2. have the skills and qualifications to do things well
3. having inherent physical or mental ability or capacity
4. having a strong healthy body

**Synonyms**

1. capable
2. able
3. able-bodied

# DEMO !