Implementation of the whoosh with the flask

~ Abhishek Kaushik

Steps:1

Installation of the whoosh
installation of the flask

Step 2:

Creating the whoosh index for the data

The steps to be followed

Step 3:

This is assumed that we have the index prepared from the whoosh

Now we start with the whoosh

In the main folder: we will create three folders

Main folder:
Static:
Template:
Whoosh Index:
main.py

The static folder :
will be contain all the css and javascript. You can download it from boot strap or you can make your on css and javascript or java query

Template folder:
This folder will contain all the html files.

Whoosh folder:
This folder contain whoosh index.

Main.py
This is the main python file which will activate the server and run the flask.

Step 4:
Configuring the flask with whoosh

The main.py standard code

```
from flask import Flask,render_template,request
# This is for importing the flask and rendering template
from whoosh.index import open_dir
from whoosh.qparser import MultifieldParser
from whoosh.fields import *
#The above three will the import from the whoosh

app = Flask(__name__)
This is the app and it is the instant for flask class with the parameter named as __name__
@app.route('/')#this will setup the path of the root of the following the functions.
def hello_world():
    return render_template('index.html')


@app.route('/search',methods=['GET', 'POST'])
```

This app route method mean that it will receive the values from the search page into the variable text which will pass into query variable with the function request.args.get("text")

```
def search():
 query=request.args.get('text')
 #return query


 index_path = r"C:\Users\Abhi\Desktop\Index"
 ix = open_dir(index_path)
 mparser = MultifieldParser(["title","docno","text_data", "author_data","bibilo_text"],
schema=ix.schema)
 q = mparser.parse(str(query))

 with ix.searcher() as searcher:
        result = searcher.search(q)
        if len(result)!=0:
```

This pass the result dictionary into the search html This is the way to transfer the by rendering the template

```
            return render_template("search.html", results=result)
        else:
            return render_template("NotFound.html")

    #count = count + 1
     # for hit in result:
```

```
    # print(len(hit))
#print(count)

#return render_template('search.html')
if __name__ == '__main__':
  app.run(port=80,debug=True)
```

The app will run after calling the run function with app instant.

Html----------

This html search page where we send our results .
The first two lines will be an example of jinja templating

```
{% extends "layout.html" %}
{% block body %}

<h1 align="center">Search Results</h1>
```

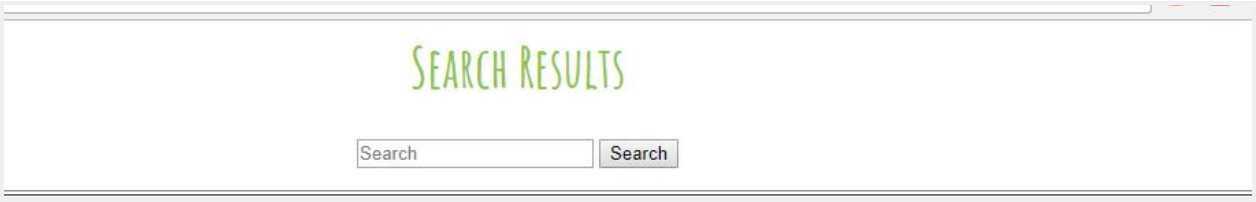
This is title in the page


```
        </ul>
        <form class="form-inline my-2 my-lg-0" method="get", action="/search">
          <input class="form-control mr-sm-2" align="center" type="text" name="text"
placeholder="Search" aria-label="Search">
          <button class="btn btn-outline-success my-2 my-sm-0"
type="submit">Search</button>
        </form>
      </div>
    </nav>
```

This is the search fox with the button

The first step is you have to define the form and it class then what would be the action and method. Details would be given html forms in any html tutorials.

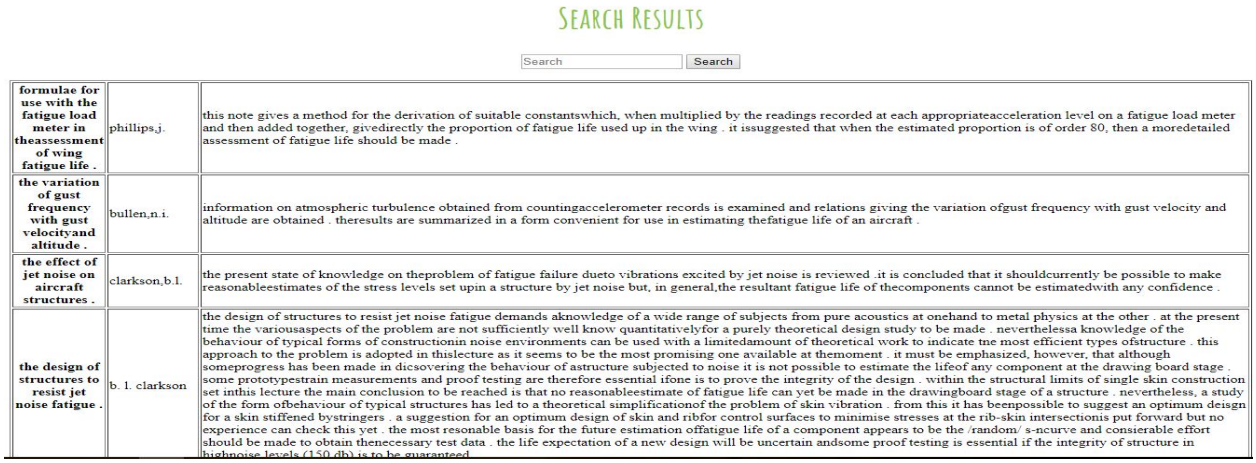
This is creating the box and sending the query written over search black box into /search page using get method after the submission of the button.



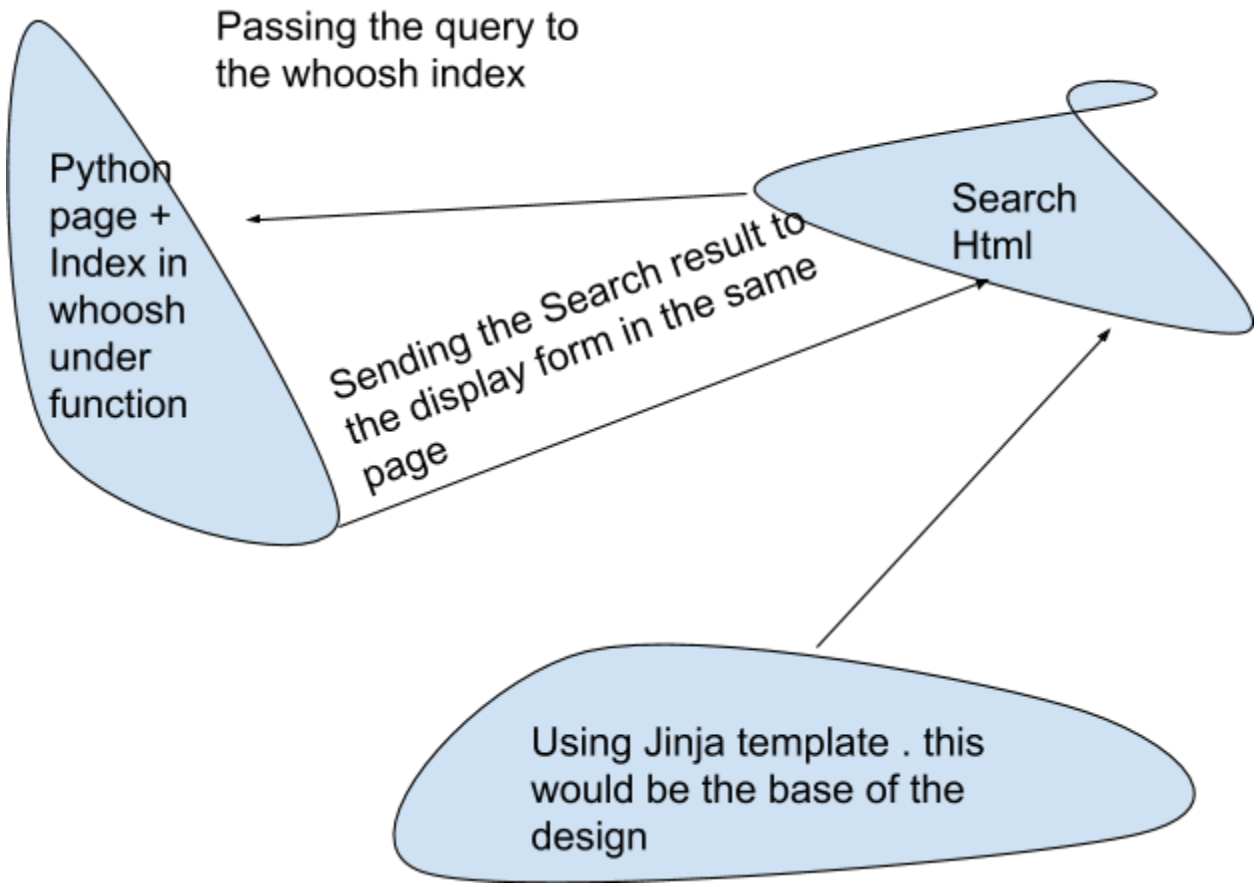
This is showing the result into the following way

```
<table border = 1>
   {% for hit in results %}

      <tr>
         <th> {{ hit['title'] }} </th>
          <td> {{ hit['author_data'] }} </td>
          <td> {{ hit['text_data'] }} </td>
      </tr>

   {% endfor %}
</table>


{% endblock %}
```

This is table form from which we can represent our data in the same page where we have the search block.



formulae for use with the fatigue load meter in theassessment of wing fatigue life .	phillips,j.	this note gives a method for the derivation of suitable constantswhich, when multiplied by the readings recorded at each appropriateacceleration level on a fatigue load meter and then added together, givedirectly the proportion of fatigue life used up in the wing . it issuggested that when the estimated proportion is of order 80, then a moredetailed assessment of fatigue life should be made .
the variation of gust frequency with gust velocityand altitude .	bullen,n.i.	information on atmospheric turbulence obtained from countingaccelerometer records is examined and relations giving the variation ofgust frequency with gust velocity and altitude are obtained . theresults are summarized in a form convenient for use in estimating thefatigue life of an aircraft .
the effect of jet noise on aircraft structures .	clarkson,b.l.	the present state of knowledge on theproblem of fatigue failure dueto vibrations excited by jet noise is reviewed .it is concluded that it shouldcurrently be possible to make reasonableestimates of the stress levels set upin a structure by jet noise but, in general,the resultant fatigue life of thecomponents cannot be estimatedwith any confidence .
the design of structures to resist jet noise fatigue .	b. l. clarkson	the design of structures to resist jet noise fatigue demands aknowledge of a wide range of subjects from pure acoustics at onehand to metal physics at the other . at the present time the variousaspects of the problem are not sufficiently well know quantitativelyfor a purely theoretical design study to be made . nevertheless a knowledge of the behaviour of typical forms of constructonin noise environments can be used with a limitedamount of theoretical work to indicate tne most efficient types ofstructure . this approach to the problem is adopted in thislecture as it seems to be the most promising one available at themoment . it must be emphasized, however, that although someprogress has been made in dicsovering the behaviour of astructure subjected to noise it is not possible to estimate the lifeof any component at the drawing board stage . some prototypestrain measurements and proof testing are therefore essential ifone is to prove the integrity of the design . within the structural limits of single skin construction set inthis lecture the main conclusion to be reached is that no reasonableestimate of fatigue life can yet be made in the drawingboard stage of a structure . nevertheless, a study of the form ofbehaviour of typical structures has led to a theoretical simplificationof the problem of skin vibration . from this it has beenpossible to suggest an optimum deisgn for a skin stiffened bystringers . a suggestion for an optimum design of skin and ribfor control surfaces to minimise stresses at the rib-skin intersectionis put forward but no experience can check this yet . the most resonable basis for the future estimation offatigue life of a component appears to be the /random/ s-ncurve and consierable effort should be made to obtain thenecessary test data . the life expectation of a new design will be uncertain andsome proof testing is essential if the integrity of structure in highnoise levels (150 db) is to be guaranteed .



Python code

```
from flask import Flask,render_template,request
from whoosh.index import open_dir
from whoosh.qparser import MultifieldParser
from whoosh.fields import *
app = Flask(__name__)

@app.route('/')
def hello_world():
   return render_template('index.html')


@app.route('/search',methods=['GET', 'POST'])
def search():
query=request.args.get('text')
#return query
```

```
index_path = r"C:\Users\Abhi\Desktop\Index"
ix = open_dir(index_path)
mparser = MultifieldParser(["title","docno","text_data", "author_data","bibilo_text"],
schema=ix.schema)
q = mparser.parse(str(query))

with ix.searcher() as searcher:
        result = searcher.search(q)
        if len(result)!=0:
           return render_template("search.html", results=result)
        else:
           return render_template("NotFound.html")

   #count = count + 1
    # for hit in result:
    # print(len(hit))
#print(count)

#return render_template('search.html')
if __name__ == '__main__':
 app.run(port=80,debug=True)
```

Search Index

-

```
{% extends "layout.html" %}
{% block body %}
<h1 align="center">Search Results</h1>
        </ul>
        <form class="form-inline my-2 my-lg-0" method="get", action="/search">
          <input class="form-control mr-sm-2" align="center" type="text" name="text"
placeholder="Search" aria-label="Search">
          <button class="btn btn-outline-success my-2 my-sm-0"
type="submit">Search</button>
        </form>
      </div>
    </nav>

      <table border = 1>
        {% for hit in results %}

            <tr>
               <th> {{ hit['title'] }} </th>
                <td> {{ hit['author_data'] }} </td>
                <td> {{ hit['text_data'] }} </td>
```

```
            </tr>

        {% endfor %}
    </table>
{% endblock %}
```

Jinja Template

```
<html>
<head>
    <title>Website</title>
<style>
@import url(http://fonts.googleapis.com/css?family=Amatic+SC:700);

body{
    text-align: center;
}
h1{
    font-family: 'Amatic SC', cursive;
    font-weight: normal;
    color: #8ac640;
    font-size: 2.5em;
}

</style>

</head>
<body >
{% block body



%}


{% endblock %}

</body>
</html>
```

Important link :

http://programminghistorian.github.io/ph-submissions/lessons/published/creating-apis-with-python-and-flask