Project Documentation

For

Intelligent Known Error Database Search Engine

Version 1.0.0

Table of Contents

Table of Contents ii

Revision History ii

1. Introduction 1

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Product Scope 1

1.5 References 1

2. Overall Description 2

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 2

2.5 Design and Implementation Constraints 2

2.6 User Documentation 2

2.7 Assumptions and Dependencies 3

3. External Interface 3

3.1 User Interfaces 3

3.2 Hardware Interfaces 3

3.3 Software Interfaces 3

3.4 Communications Interfaces 3

4. System Features or Functional Features 4

4.1 System Feature 1 4

4.2 System Feature 2 (and so on) 4

5. Other Nonfunctional Features 4

5.1 Performance Requirements 4

5.2 Safety Requirements 5

5.3 Security Requirements 5

5.4 Software Quality Attributes 5

5.5 Business Rules 5

6. Logical Modules 5

Appendix A: Glossary 5

Appendix B: Analysis Models 5

Appendix C: To Be Determined List 6

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

This document is for the Intelligent Known Error Database version 1.0.0.

This documentation describes the functions and performance requirements of the Intelligent KEDB. The IKEDB allows performing search about known errors in unstructured data and getting a valid workaround.

## Document Conventions

Throughout this document the following conventions have been used:-

* Font : Times New Roman
* Main Headings : Size 18
* Sub-Headings : Size 16
* Rest of the document : Size 14
* Words in Bold are important and have been formatted to grab the attention of the reader.

## Intended Audience and Reading Suggestions

This document is meant for users, developers, project managers, testers and documentation writers. The document aims to explain in an easy manner, the basic idea behind the IKEDB and how the developers aim to achieve their goals.

It also aims to introduce to the users the main features of the IKEDB and what makes it different from other KEDB.

## Product Scope

We students have looked into the analysis of the KEDB search engine and its design and implementation (integration of modules too).

Animation based search engine and Natural Language Processing are the specific areas we will be dealing in the next few days prior to the implementation details. A survey of the existing KEDBs has been made in order to understand the in-addition expectations from the current IKEDB.

## References

Python 2.7: <https://www.python.org/download/releases/2.7/>

Django 1.11.4: <https://www.djangoproject.com/download/>

NLTK 3.2.4: <http://www.nltk.org/>

Openpyxl 2.5.Oa3: <https://pypi.python.org/pypi/openpyxl>

# Overall Description

## Product Perspective

The IKEDB is a standalone system. It provides user to search for known errors and receive a workaround with highest accuracy possible.

## Product Functions

The main function of the IKEDB is to allow its user to perform search in natural language and receive workaround for a valid known error from unstructured data. The product also allows managers to add known error database at any moment of time.

## User Classes and Characteristics

The major user classes that are expected to use the product are as follows:

* + - Enterprises
    - Reception authorities
    - General user

## Operating Environment

* Client Side :

OS: Linux, Windows

Software Packages: Browser

* Server Side :

OS: Linux, Windows

Software Packages: Python 2.7, Django 1.11.4, Natural Language Toolkit 3.2.4, Openpyxl 2.5.Oa3

## Design and Implementation Constraints

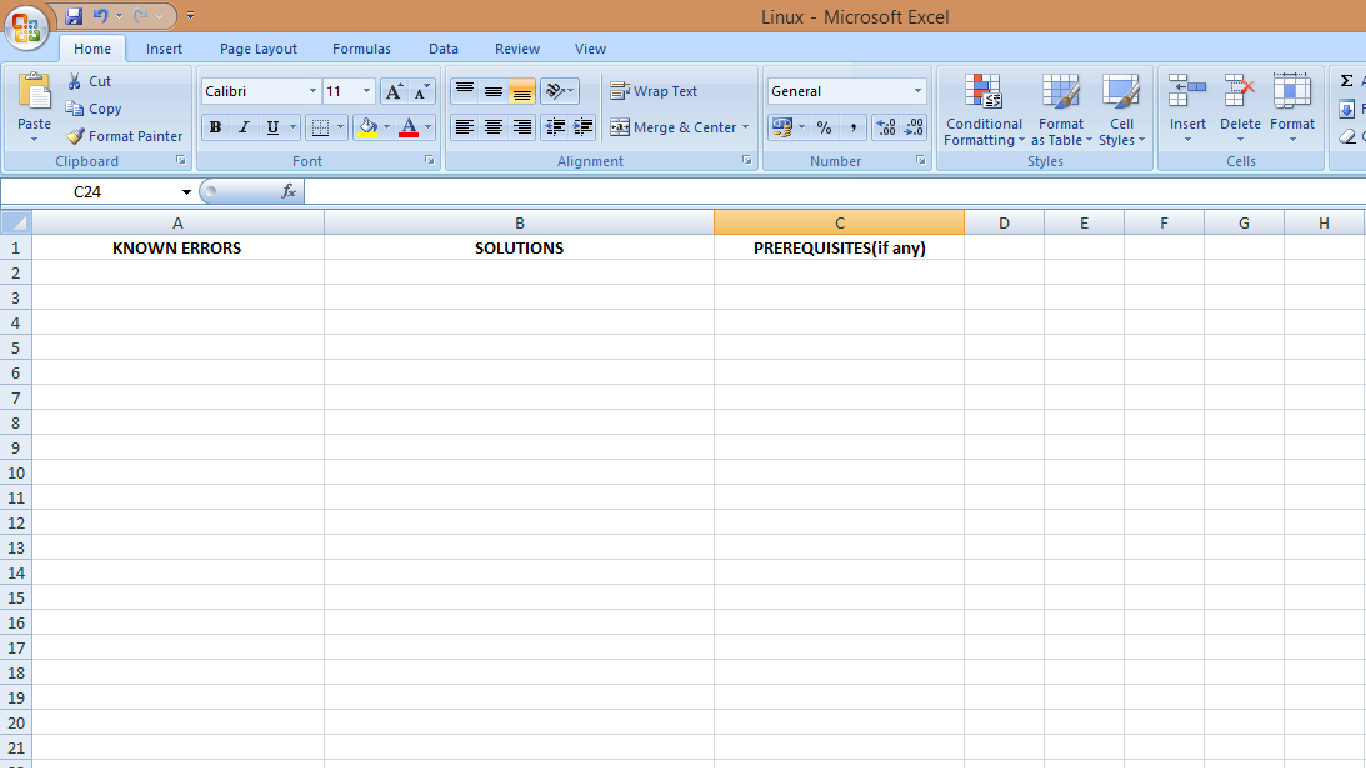
* 100% accuracy was not obtained.
* Search result will be displayed only if more than the 0% of the input is matched.
* File filtration can be done based on only one file.

## User Documentation

* Go to homepage of the IKEDB search engine.
* Enter query. (The IKEDB allows query in Natural Language).
* Receive solution based on entry present on Known Error Database server.
* Filtration possible using radio buttons provided on left panel.
* From the result page further query can be searched.

## Assumptions and Dependencies

* The KEDB must have .xls extension.
* The KEDB file format must be of the form :



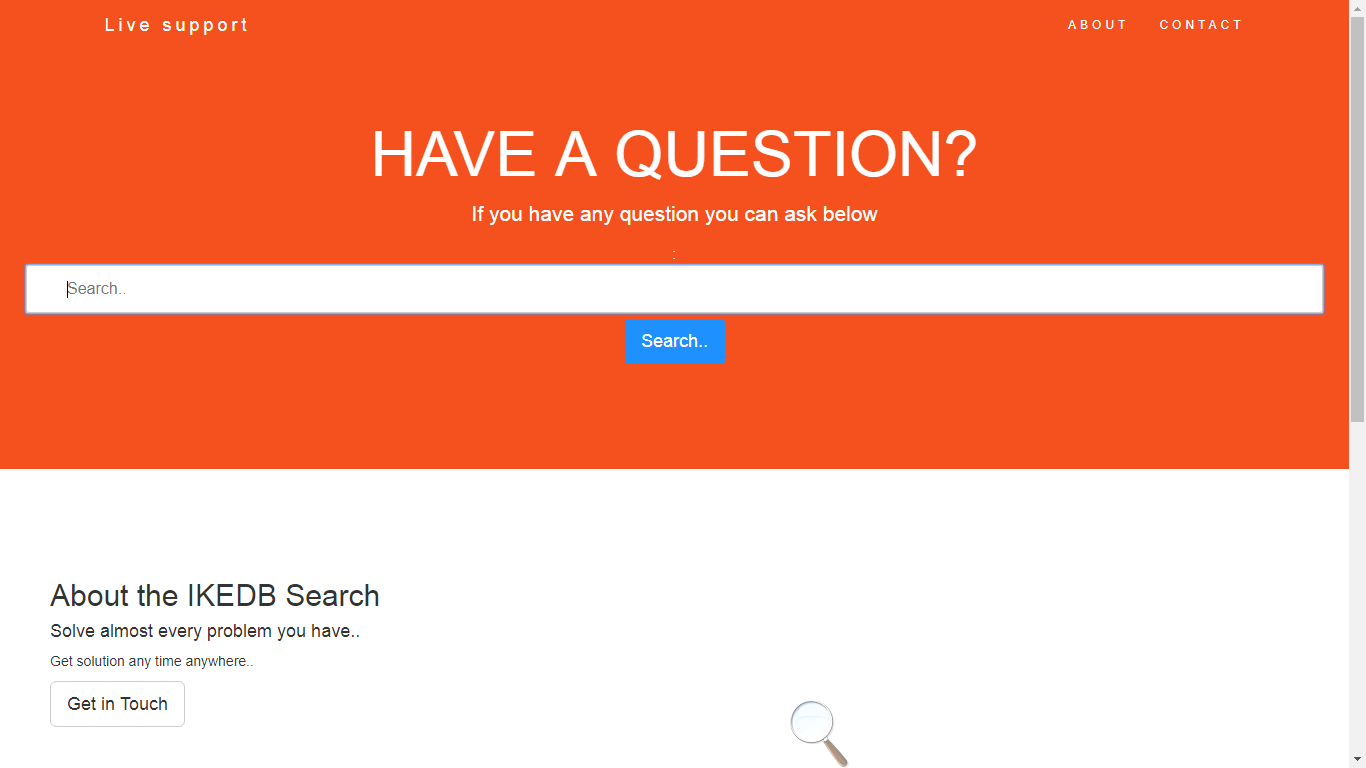
* Spelling correction is done based on the input in the KEDB files. So user’s entry may get corrected to match the most probable answer.
* Path must be set for KEDB files in access\_to\_kedbfiles.txt.

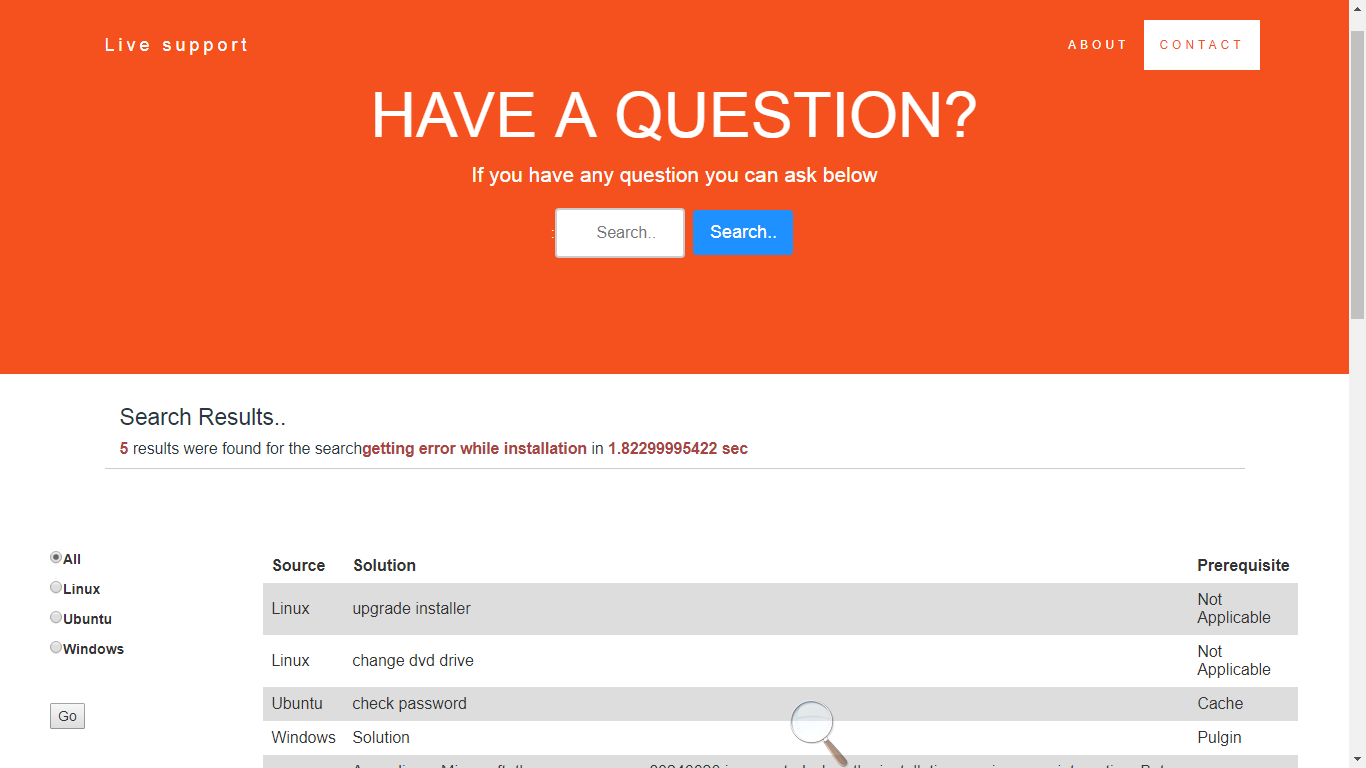
# External Interface Requirements

## User Interfaces

* A screen containing a search panel providing area for the user to input his search query.
* A result page with all the matched results from various KEDB files along with file names and a left panel with radio buttons for filtration.

Sample User Interface:





## Hardware Interfaces

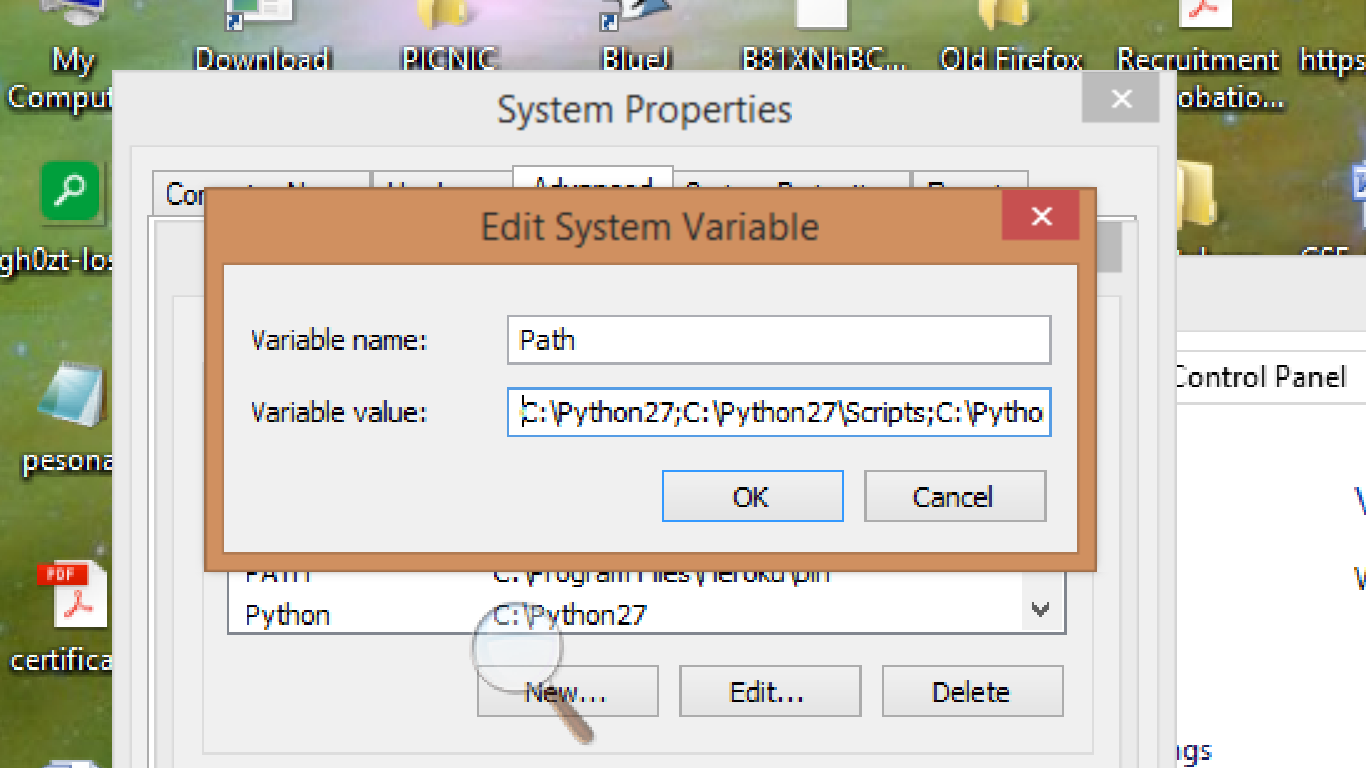
* The user uses the search engine through a browser.
* The manager uploads the KEDB files to a repository.

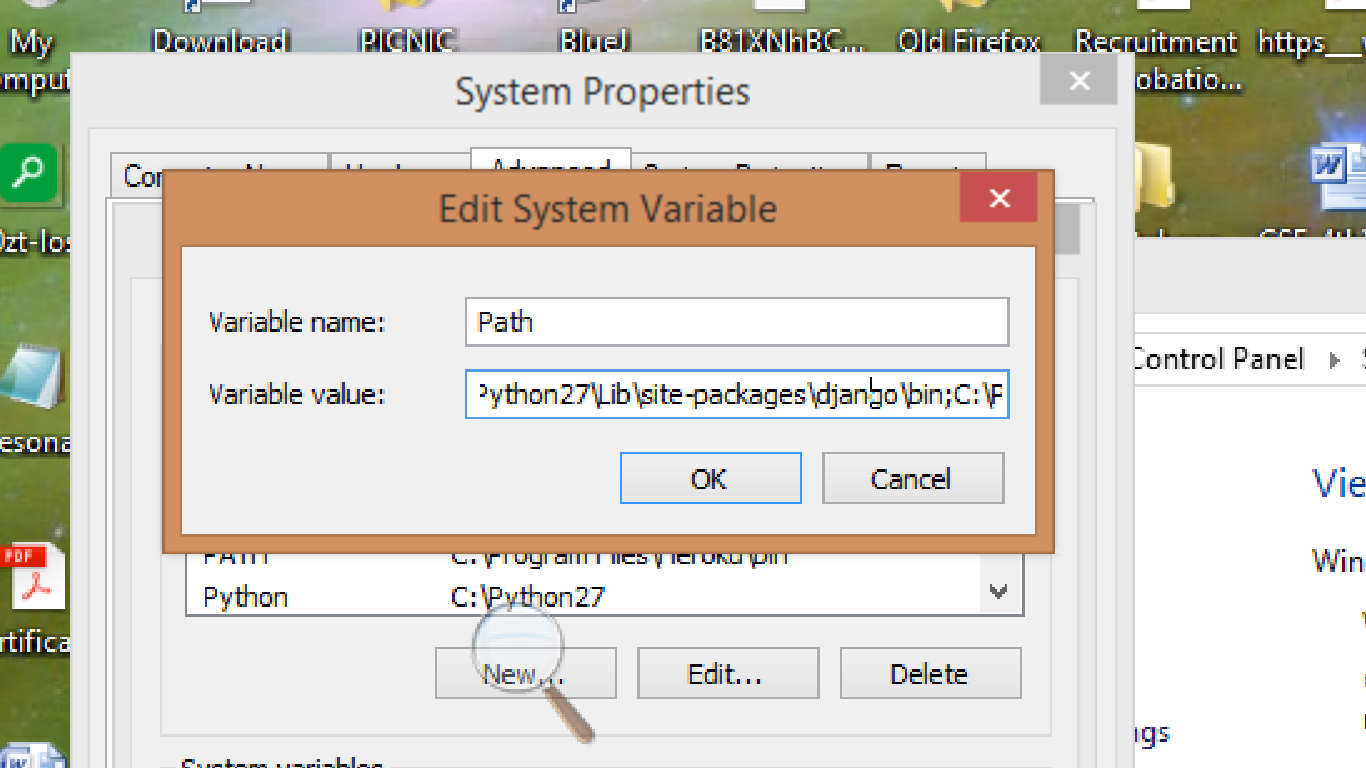
## Software Interfaces

* Django-Python based web framework is used for development and deployment for our project.
* Local deployment and testing is done on the Django provided default server using cmd :

$ python project\_name runserver local\_ip:port\_number

* HTML, CSS, JavaScript, Bootstrap used for the templates of Django that is the UI display provided to the user.
* Path must be set in Environment Variables as follows :





## Communications Interfaces

* Connection between user machine and server is done using HTTP protocol.

# System Features or Functional Features

* Query processing in natural language.
* Spell Correction of the query entered by the end user.
* Modification of already existing KEDB files, addition of new KEDB files and deletion of existing KEDB files.
* Display of solution based on ranking if there are more than one search results.
* The end user gets to know the process time.
* Filtration by a particular file.

# Other Nonfunctional Features

## Performance

The number of files in which the searching is to be performed is dynamic. The search result should be ranked based on accuracy. The KEDB files will have unstructured data. The search query will be in natural language. 20% of the input query should match with the KEDB for valid result.

## Software Quality Attributes

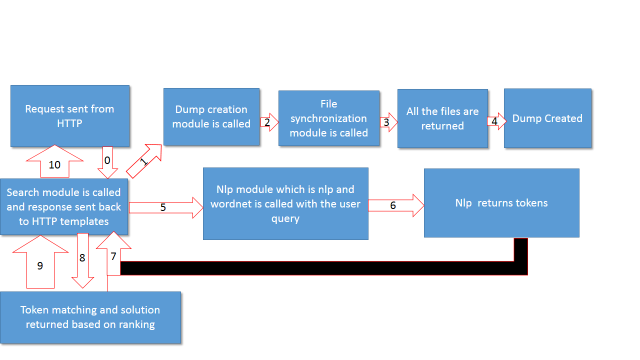
All the modules are built using Python making the software platform independent. The software provides easy to use and understandable user interface.

## Business Rules

The administrator can use the administrator GUI to upload the Known Error Database Files.

# Logical Modules

The implementation and linking of various modules in architectural design are as follows:



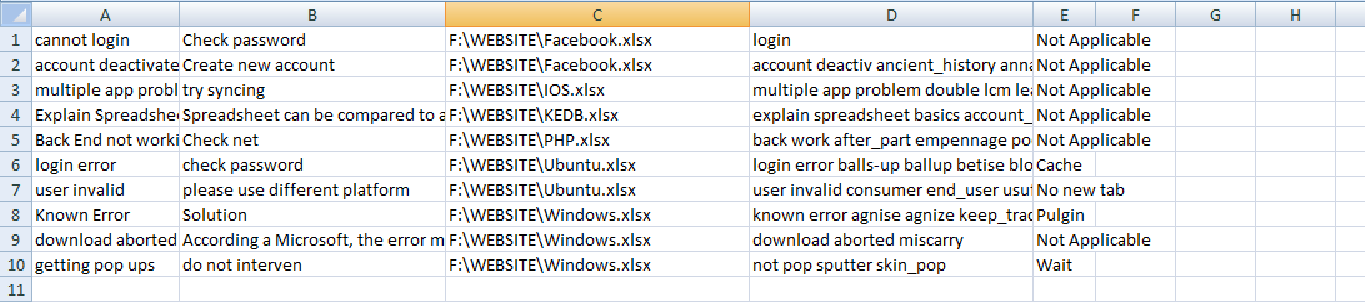
## File Synchronization

* This module uses access\_to\_kedbfiles.txt to collect path of all the .xls KEDB files in the repository.
* The file synchronization module checks for any new KEDB files or any modification to already existing files or any deletion of KEDB files. If any, it will synchronize all the available files and returns all the files along with their paths.
* This module reads /access\_to\_kedbfiles.txt where path to all KEDB files is stored and then checks all the KEDB files with extension .xls files available at that time and returns a list containing all the files along with their full path.
* The Python code for this module –

**import** os  
**import** time  
**import** re  
**def** syncfile():  
 fp=open(**"c:\\Python27\\myproject\\myapp\\access\_to\_kedbfiles.txt"**)  
 data=fp.read()  
 fp.close()  
 find=**r'%s'**%data  
 **global** full,full1  
 os.chdir(find)  
 file1=[]  
 full=[]  
 full1=[]  
 **for** root,folders,files **in** os.walk(**'.'**):  
  
 file1=files  
 **for** i **in** file1:  
 **if** re.search(**'.xls'**,i):  
 j= os.path.abspath(i)  
 full.append(j)  
 **return** full

## Dump Creation

* A Master.xlsx file is created when the app is first run.
* First column: Known Errors from all available KEDB files.
* Second column: Solution from all available KEDB files.
* Third column: File names with path from where known error and solution is taken.
* Fourth column: All possible tokens generated after passing the first column entry in nlp module.
* Fifth column: Prerequisite if any or “Not Applicable” if no prerequisite.
* Master.xlsx is recreated after first run only if the modification time of master.xlsx is greater than modification time of any other KEDB files.
* Sample MASTER.xlsx file.



* Code for Dump Creation module :

import filesync

from openpyxl import load\_workbook

from openpyxl import Workbook

import nlp

import os

def dump():

l = filesync.syncfile()

wb = Workbook()

ws=wb.active

rng=1

for i in l:

wb2=load\_workbook(i)

ws2=wb2.active

row=2

col=1

#counting total number of entries in all KEDB files

while True:

cr=ws2.cell(row=row,column=col).value

if cr is None:

break

else:

rng+=1

row+=1

#creating master KEDB files with total no of rows counted and 6 columns

for i in range(1, rng):

for j in range(1, 7):

ws.cell(row=i, column=j)

m=1

for i in l:

row\_no=2

wb1=load\_workbook(i)

ws1=wb1.active

while True:

c=ws1.cell(row=row\_no,column=1).value

c1=ws1.cell(row=row\_no,column=2).value

c2=i

c3=ws1.cell(row=row\_no,column=3).value

if c3 is None:

#if no prerequisite then adding "not applicable in prerequisite section"

c3="Not Applicable"

if c is None:

break

else:

ws.cell(row=m,column=1).value=c

ws.cell(row=m,column=2).value=c1

ws.cell(row=m,column=3).value=c2

ws.cell(row=m,column=5).value=c3

m+=1

row\_no+=1

#create KEDB.txt with only the known error column required for spell check

fp=open("c:\\Python27\\myproject\\myapp\\KEDB.txt",'w')

r=1

while True:

c = ws.cell(row=r, column=1).value

if c is None:

break

else:

c=c+'\n'

fp.write(c)

r += 1

fp.close()

r=1

while True:

c1 = ws.cell(row=r, column=1).value

c=str(c1)

tkn=nlp.create\_keyword(c)

if c1 is None:

break

else:

tkn1=tkn

ws.cell(row=r, column=4).value=tkn1

r += 1

for i in l:

tm=os.path.getmtime("c:\\Python27\\myproject\\myapp\\MASTER.xlsx")

if tm < os.path.getmtime(i):

wb.save("c:\\Python27\\myproject\\myapp\\MASTER.xlsx")

break

dump()

## Natural Language Processing

**Requirements Specifications**

To make our search more effective and intelligent we have used the concept of natural language processing. Here in our project we have used **NLTK**(Natural Language Toolkit), a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language.

So to proceed with our project we first need to install **NLTK** in our system. And for the installation of NLTK we have to previously install python.

**Analysis**

A known error is a fault in a Configuration Item (CI) identified by the successful diagnosis of a Problem and for which a temporary work-around or a permanent solution has been identified. Therefore a known error is an already identified solution to an existing or a new issue.

The Known Error Database is a repository of information that describes all of the conditions in the IT systems that might result in an incident for the customers and users. The creation of this intelligent KEDB(Known Error Database) focuses on the natural language processing(NLP) to process the user questions and retrieve the answer. NLP is the backbone of this application that involves text processing and data chunking concepts. NLP deals with processing and analyzing the human language to communicate with machine.

**Design Specification**

Architectural Design

To extract the keywords from the query entered by the users we need to follow various steps which are indicated below:

**Spell correction**

The spell correction module uses a text file KEDB.txt which has all the entries from Known error column of the MASTER.xlsx file.It finds probability of all the words from the KEDB.txt file.Then it edits wrongly spelled word based on insertion, deletion, transpose, replace and finds minimum edits required to match the most probable word in the KEDB.txt file for the wrongly spelled word, and accordingly it is corrected.It is dependent on the fact that even if the user is entering correct spelled word the query will get corrected based on KEDB.txt as it is taken as reference. In this case user may not convey its actual meaning.

**Stemming and Lemmatization**

Stemming and Lemmatization are the basic text processing methods for English text. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Stemming has been done for reducing inflected (or sometimes derived) words to their stem, base or rootform-generally a written word form. The stem need not to be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not itself a valid root. We have used here the best-known and most popular stemming approach for English- **Porter Stemming Algorithm**, also known as Porter Stemmer. Here, the Porter stemmer chops both apple and apples to appl, and it stems berry and berries to berri.

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.It does not simply chop off the inflections, but instead relies on lexical knowledge base like **WordNet**to obtain the correct form of the words.It converts words like goose to geese and lay to lie. But the WordNet lemmatizer requires specifying the word’s part of speech otherwise it assumes word is a noun.

**Removal of punctuation**

Before tokenization into words we will remove all the **punctuation** marks from the text. We have defined a tokenizer that picks out a sequences of alphanumeric characters as tokens and drops everything else.

**Tokenization**

The text now which we obtain does not contain punctuation and stopwords. Then the text is tokenized into words using word\_tokenize.

**Removal of Stopwords**

Stopwords are set of commonly used words which are eliminated from many text processing applications because these words may be distracting, non-informative and are additional memory overhead. Stopwords removal is one of the most commonly used preprocessing steps across different NLP application. Typically, articles and pronouns are generally classified as stopwords. Hence removal of stopwords reduces memory overheads, noise and false positives and can potentially improve the power of prediction.

After performing all the steps now we have obtained the keywords.

**Implementation**

**Spell Check Code**

import re

from collections import Counter

def words(text): return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open('kedb.txt').read()))

def P(word, N=sum(WORDS.values())):

"Probability of `word`."

return WORDS[word] / N

def correction(word):

"Most probable spelling correction for word."

return max(candidates(word), key=P)

def candidates(word):

"Generate possible spelling corrections for word."

return (known([word]) or known(edits1(word)) or known(edits2(word)) or [word])

def known(words):

"The subset of `words` that appear in the dictionary of WORDS."

return set(w for w in words if w in WORDS)

def edits1(word):

"All edits that are one edit away from `word`."

letters = 'abcdefghijklmnopqrstuvwxyz'

splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]

deletes = [L + R[1:] for L, R in splits if R]

transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]

replaces = [L + c + R[1:] for L, R in splits if R for c in letters]

inserts = [L + c + R for L, R in splits for c in letters]

return set(deletes + transposes + replaces + inserts)

def edits2(word):

"All edits that are two edits away from `word`."

return (e2 for e1 in edits1(word) for e2 in edits1(e1))

if \_\_name\_\_=="\_\_main\_\_":

s1=list()

search=text

s=search.split(' ')

for i in s:

s1.append(correction(i))

text=' '.join(s1)

returntext

**Keyword Extraction code**

import nltk

from nltk.tokenize import RegexpTokenizer

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from nltk.tokenize import PunktSentenceTokenizer

def create\_keyword(t):

text1=t.lower()

ps=PorterStemmer()

tokens = nltk.word\_tokenize(text1)

text=[ps.stem(t) for t in tokens]

wnl = nltk.WordNetLemmatizer()

text = [wnl.lemmatize(t) for t in text]

text=' '.join(text)

tokenizer = RegexpTokenizer(r'\w+')

tokens = tokenizer.tokenize(text)

filtered\_words = [w for w in tokens if not w in stopwords.words('english')]

text=' '.join(filtered\_words)

return text

## Search

* The user entered query is spell corrected and passed to search module for searching and finding solution.
* Then the query is sent to nlp module to generate keywords and these keywords are stored as a string separated by blank space in text1.
* Then one by one we take the entry of the first column of MASTER.xlsx which contains the questions of the kedb file and generate their respective keywords using the nlp module which in turn is also stored as string separated by blank space.
* The string which is generated is stored in the fourth column of MASTER.xlsx against their respective row of questions.
* Now we calculate the cosine similarity between the two strings i.e. text1 and the strings stored in the fourth column, one by one.
* The string which gives the maximum cosine similarity is our best result.
* We now print all the possible solutions whose cosine similarity is greater than zero. First the best search result is printed followed by the next best result and so on (in descending order).

**Implementation**

import re, math

from collections import Counter

import openpyxl

import nlp

WORD = re.compile(r'\w+')

def get\_cosine(vec1, vec2):

intersection = set(vec1.keys()) & set(vec2.keys())

numerator = sum([vec1[x] \* vec2[x] for x in intersection])

sum1 = sum([vec1[x]\*\*2 for x in vec1.keys()])

sum2 = sum([vec2[x]\*\*2 for x in vec2.keys()])

denominator = math.sqrt(sum1) \* math.sqrt(sum2)

if not denominator:

return 0.0

else:

return float(numerator) / denominator

def text\_to\_vector(text):

words = WORD.findall(text)

return Counter(words)

def search(text1):

text1=nlp.create\_keyword(text1)

global l,l1,l3,l2,l4,l5,l7,l6

vector1 = text\_to\_vector(text1)

max=0

i=1

l=list()

l2=list()

l4=list()

l6=list()

l1=list()

l3=list()

l5=list()

l7=list()

wb = openpyxl.load\_workbook("c:\\Python27\\myproject\\myapp\\MASTER.xlsx")

first\_sheet = wb.get\_sheet\_names()[0]

worksheet = wb.get\_sheet\_by\_name(first\_sheet)

#here you iterate over the rows in the specific column

for row in range(1,worksheet.max\_row+1):

for column in "D": #Here you can add or reduce the columns

cell\_name = "{}{}".format(column, row)

cell\_name1="{}{}".format("B",row)

cell\_name2="{}{}".format("C",row)

cell\_name3="{}{}".format("E",row)

text2=worksheet[cell\_name].value # the value of the specific cell

text3=worksheet[cell\_name1].value

text4=worksheet[cell\_name2].value

text5=worksheet[cell\_name3].value

vector2 = text\_to\_vector(text2)

cosine = get\_cosine(vector1, vector2)

print cosine

l.append(cosine)

l2.append(text3)

l4.append(text4)

l6.append(text5)

if max<cosine:

temp=cosine

cosine=max

max=temp

print 'Cosine:', max

l1=[x for (x,y,z,w) in sorted(zip(l,l2,l4,l6), reverse=True) if x>0 ]

l3=[y for (x,y,z,w) in sorted(zip(l,l2,l4,l6), reverse=True) if x>0 ]

l5=[z for (x,y,z,w) in sorted(zip(l,l2,l4,l6),reverse=True) if x>0 ]

l7=[w for (x,y,z,w) in sorted(zip(l,l2,l4,l6),reverse=True) if x>0 ]

print l1

print l3

def get\_search\_result(s):

search(s)

return l3

def pre():

return l7

def get\_file\_name():

fl=[]

print l5

for j in l5:

k=j.split("\\")

print k

k1=k[-1]

leng=len(k1)

i1=k1[0:leng-5:]

fl.append(i1)

print fl

return fl

## Django Framework

* The app is developed and hosted locally using Django.
* Django form with char field is created and when requested is passed into this form Django view is executed.
* After executing our various modules we pass our results into our templates and display the results.
* Various Django files and their codes :

Views.py – This is the heart of our Django framework where we accept HTTP request and render HTTP response to our HTML templates. Code:

**from** django.http **import** HttpResponse  
**from** django.template **import** loader  
**from** django.shortcuts **import** render,redirect  
**from** django.http **import** HttpResponseRedirect  
**import** search  
**import** filesync  
**import** spellcheck  
**import** time  
*#forms***from** forms **import** NameForm  
  
  
**def** get\_name(request):  
  
 *# if this is a POST request we need to process the form data* **if** request.method == **'POST'**:  
 *# create a form instance and populate it with data from the request:* form = NameForm(request.POST)  
 *# check whether it's valid:* **if** form.is\_valid():  
 *#start of search processing time* start\_time=time.time()  
 pre=search.predictive()  
 your\_name = request.POST.get(**'your\_name'**)  
 your\_name1=spellcheck.spell\_correction(your\_name)  
 s=search.get\_search\_result(your\_name)  
 s1=search.get\_file\_name()  
 s3=[]  
 s4=search.pre()  
 **for** i **in** s1:  
 **if** i **not in** s3:  
 s3.append(i)  
 s2=zip(s,s1,s4)  
 c=len(s)  
 tim=time.time()-start\_time  
 tim=str(tim)+**" sec"** *#end of search process time* form=NameForm()  
  
 **return** render(request,**'myapp/thanks.html'**,{**'name'**:your\_name1,**'file'**:s3,**'filename'**:s2,**'number'**:c,**'result'**:s,**'form'**:form,**'soln'**:s,**'pre'**:pre,**'ti'**:tim})  
 **else**:  
 form = NameForm()  
 *#template = loader.get\_template("myapp/name.html")* context = {**'form'**: form}  
 *#return HttpResponse(template.render(context, request))* **return** render(request, **"myapp/name.html"**, context)

Urls.py- This is the file where the url is matched and proper function in views is executed. Code:

**from** django.conf.urls **import** url  
**from** . **import** views  
  
*#app\_name = 'app'*urlpatterns = [  
 url(**r'^$'**, views.get\_name, name = **'get\_name'**),  
 url(**r'^getname/$'**, views.get\_name, name = **'get\_name'**)  
 ]

Settings.py-this is file where we set the basic framework like ALLOWED\_HOSTS, INSTALLED\_APPS and MIDDLEWARES. Full Code for our project :

**import** os  
  
*# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)*BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
  
  
*# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/1.11/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!*SECRET\_KEY = **'secret\_key\_generated from django install’***# SECURITY WARNING: don't run with debug turned on in production!*DEBUG = True  
 *#hosting ip*ALLOWED\_HOSTS = [**'192.168.0.100'**,**'localhost'**,**'127.0.0.1'**]  
  
  
*# Application definition*INSTALLED\_APPS = [  
 **'myapp'**,  
 **'django.contrib.admin'**,  
 **'django.contrib.auth'**,  
 **'django.contrib.contenttypes'**,  
 **'django.contrib.sessions'**,  
 **'django.contrib.messages'**,  
 **'django.contrib.staticfiles'**,  
]  
  
MIDDLEWARE = [  
 **'django.middleware.security.SecurityMiddleware'**,  
 **'django.contrib.sessions.middleware.SessionMiddleware'**,  
 **'django.middleware.common.CommonMiddleware'**,  
 *#'django.middleware.csrf.CsrfViewMiddleware',* **'django.contrib.auth.middleware.AuthenticationMiddleware'**,  
 **'django.contrib.messages.middleware.MessageMiddleware'**,  
 **'django.middleware.clickjacking.XFrameOptionsMiddleware'**,  
]  
  
ROOT\_URLCONF = **'myproject.urls'**TEMPLATES = [  
 {  
 **'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,  
 **'DIRS'**: [],  
 **'APP\_DIRS'**: True,  
 **'OPTIONS'**: {  
 **'context\_processors'**: [  
 **'django.template.context\_processors.debug'**,  
 **'django.template.context\_processors.request'**,  
 **'django.contrib.auth.context\_processors.auth'**,  
 **'django.contrib.messages.context\_processors.messages'**,  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = **'myproject.wsgi.application'***# Database  
# https://docs.djangoproject.com/en/1.11/ref/settings/#databases*DATABASES = {  
 **'default'**: {  
 **'ENGINE'**: **'django.db.backends.sqlite3'**,  
 **'NAME'**: os.path.join(BASE\_DIR, **'db.sqlite3'**),  
 }  
}  
  
  
*# Password validation  
# https://docs.djangoproject.com/en/1.11/ref/settings/#auth-password-validators*AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.MinimumLengthValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.CommonPasswordValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.NumericPasswordValidator'**,  
 },  
]  
  
  
*# Internationalization  
# https://docs.djangoproject.com/en/1.11/topics/i18n/*LANGUAGE\_CODE = **'en-us'**TIME\_ZONE = **'UTC'**USE\_I18N = True  
  
USE\_L10N = True  
  
USE\_TZ = True  
  
  
*# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/1.11/howto/static-files/*STATIC\_URL = **'/static/'**

Appendix A: Glossary

<Define all the terms necessary to properly interpret the DOCUMENTATION, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each DOCUMENTATION.>

Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

Appendix C: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the DOCUMENTATION so they can be tracked to closure.>