1.

client = MongoClient('localhost'**, 27017**)  
  
dbname = config.dbname  
collectionname = config.glossarycn  
filespath = config.filespath  
extentions = [".jsp"**,** ".java"**,** ".css"**,** ".js"]

Connection to MongoDB, file path, database name and collection name from config file.

2.

def getallfiles(filespath**,**extentions):#function to return the file paths in the given directory with given extention  
 filelist = []  
 for root**,** dirs**,** files in os.walk(filespath):  
 for file in files:  
 if file.lower().endswith(tuple([item.lower() for item in extentions])):  
 filelist.append(os.path.join(root**,** file))  
 return filelist

function to fetch all the files with given extensions.

3.

def glossary(filespath**,** extentions):  
 *"""  
 This function is to fetch variables in java files* ***:return****:  
 """* files = getallfiles(filespath**,** extentions)  
 screenfield = get\_screenfield\_data()  
 METADATA = []  
 variable\_name = ''  
  
 for file in files:  
 if file.endswith('.java'):  
 f = open(file**,** 'r')  
 for line in f.readlines():  
 # if line.\_\_contains\_\_('new') and line.\_\_contains\_\_('(') and line.\_\_contains\_\_(')') and line.\_\_contains\_\_(  
 # '='):  
 if all(x in line for x in ['new'**,** '('**,** ')'**,** '=']):  
 # print(line)  
 variable\_name = line.split('=')[**0**].split()[-**1**]  
 # print(variable\_name)  
 modeldict = {  
 "File\_name": file.split('\\')[-**1**]**,** "Variable": variable\_name**,** "Business\_Meaning": ""  
 }  
 METADATA.append(copy.deepcopy(modeldict))  
 modeldict.clear()  
 if line.strip().startswith('int') or line.strip().startswith('float') or line.strip().startswith(  
 'boolean'):  
 # print(line)  
 if line.strip().endswith(';'):  
 line = line.replace(';'**,** ''**, 1**)  
  
 # print(line)  
 variable\_name = line.split('=')[**0**].split()[-**1**]  
 # print(variable\_name)  
 modeldict = {  
 "File\_name": file.split('\\')[-**1**]**,** "Variable": variable\_name**,** "Business\_Meaning": ""  
 }  
 METADATA.append(copy.deepcopy(modeldict))  
 modeldict.clear()  
 # if line.strip().startswith('static') or line.strip().startswith('private') or line.strip().startswith(  
 # 'public'):  
 if line.strip().startswith(('private'**,** 'public'**,** 'static')):  
  
 if line.strip().endswith(';') and not (line.\_\_contains\_\_('=') or line.\_\_contains\_\_('(')):  
 # print(line,file)  
 line = line.replace(';'**,** ''**, 1**)  
 # print(line)  
 variable\_list = line.split()  
 # print(variable\_list)  
 for i in variable\_list[**2**:]:  
 variable\_list = i.split(',')  
 # print(variable\_list)  
 for variable in variable\_list:  
 if variable != '':  
 # print(variable)  
 modeldict = {  
 "File\_name": file.split('\\')[-**1**]**,** "Variable": variable**,** "Business\_Meaning": ""  
 }  
 METADATA.append(copy.deepcopy(modeldict))  
 modeldict.clear()  
  
  
 if line.strip().startswith('public') or line.strip().startswith('private') or line.strip().startswith(  
 'protected') or line.strip().startswith('static'):  
 if line.strip().endswith(';') and line.\_\_contains\_\_('='):  
 line = line.replace(';'**,** ''**, 1**)  
 # print(line)  
 variable\_name = line.split("=")[**0**].split("private")[-**1**]  
 variable\_name = variable\_name.split()[-**1**]  
 # print(variable\_name)  
  
 modeldict = {  
 "File\_name": file.split('\\')[-**1**]**,** "Variable": variable\_name**,** "Business\_Meaning": ""  
 }  
 METADATA.append(copy.deepcopy(modeldict))  
 modeldict.clear()  
 if line.strip().startswith('public') or line.strip().startswith(  
 'private') or line.strip().startswith(  
 'protected'):  
 if line.strip().endswith(';') and line.\_\_contains\_\_('(') and line.\_\_contains\_\_(')'):  
 # print(line)  
 variables = line.split('(')[**1**].split(')')[**0**]  
 # print(variable\_name)  
 if variables.strip() != '':  
  
 variables = variables.split(',')  
 for var in variables:  
 variable\_name = var.split()[-**1**]  
 if variable\_name.strip() != '':  
 # print(variable\_name)  
  
 # variable\_name = line.split(')')[0].split()[-1]  
 modeldict = {  
 "File\_name": file.split('\\')[-**1**]**,** "Variable": variable\_name**,** "Business\_Meaning": ""  
 }  
 METADATA.append(copy.deepcopy(modeldict))  
 modeldict.clear()  
 # print(variable\_name)  
 # print(line)  
 for record in screenfield:  
 # print(record)  
 # print(file.split('\\')[-1])  
 if record['screenfield'] != '':  
 if record['filename'] == file.split('\\')[-**1**]:  
 # print(record)  
 modeldict = {  
 "File\_name": file.split('\\')[-**1**]**,** "Variable": record['screenfield']**,** "Business\_Meaning": ""  
 }  
 METADATA.append(copy.deepcopy(modeldict))  
 modeldict.clear()  
  
 # print(json.dumps(Metadata,indent=4))  
 metadata = []  
 for doc in METADATA:  
 if doc not in metadata:  
 metadata.append(doc)  
  
 return metadata

function to fetch variables from required patterns in java files.

5.

def dbinsertfunction(dbname**,** collectionname**,**filespath):  
 *"""  
 this function is to update database by calling show code and getfiles functions* ***:param*** *dbname: database name from config file* ***:param*** *collectionname: collectionname from config file  
 """* output = glossary(filespath**,**extentions)  
  
 # print(output)  
 col = client[dbname][collectionname]  
 if output != []:  
 if col.count\_documents({}) != **0**:  
 col.drop()  
 print("Deleted the old"**,** dbname**,** collectionname**,** "collection")  
  
 col.insert\_one({"type": "metadata"**,** "headers": [  
 ""  
 "File\_name"**,** "Variable"**,** "Business\_Meaning"  
  
 ]})  
 col.insert\_many(output)  
 print("Inserted the list of jsons of"**,** dbname**,** collectionname)  
  
 else:  
 print("There are no jsons in the output to insert in the DB"**,** dbname**,** collectionname)

function to update output to database, only if output is not empty.

6.

def get\_screenfield\_data():  
 *"""  
 1.In this function we are getting screenfield data from database  
 2.If screenfield is not empty then we are creating json for glossary report* ***:return****: returns glossary report json's for non empty screen fields  
 """* col = client[dbname]['screenfields']  
 validation\_dict = {}  
 output = []  
 screenfields\_list = []  
 screenfileds\_data = list(col.find({'type': {"$ne": "metadata"}}**,** {'\_id': **0,** 'filename': **1,** 'screenfield': **1**}))  
  
 return screenfileds\_data

Fetching screenfields collection from database.

7.

def variables\_count():  
 *"""  
 This function is to update number of variables in master inventory report  
 """* master\_col=client[dbname]['master\_inventory\_report']  
 master\_inventory\_data = list(master\_col.find({'type': {"$ne": "metadata"}}**,** {'\_id': **0,**'component\_name':**1**}))  
 variable\_jsons=glossary(filespath**,**extentions)  
 files=getallfiles(filespath**,**['.java'])  
 variables\_count=[]  
 for file in files:  
 for record in variable\_jsons:  
 if record['File\_name']==file.split('\\')[-**1**]:  
 variables\_count.append(record)  
 count=len(variables\_count)  
 if count!=**0**:  
 for item in master\_inventory\_data:  
 if item['component\_name']==file.split('\\')[-**1**]:  
 data=({"component\_name": file.split('\\')[-**1**]}**,** {"$set": {"no\_of\_variables": count}})  
 #print(file.split('\\')[-1], count)  
 variables\_count.clear()  
 try:  
 master\_col.update\_one(\*data)  
 except Exception as e:  
 exc\_type**,** exc\_obj**,** exc\_tb = sys.exc\_info()  
 fname = os.path.split(exc\_tb.tb\_frame.f\_code.co\_filename)[**1**]  
 print(exc\_type**,** fname**,** exc\_tb.tb\_lineno)

Function to update number of variables in master inventory collection.

8.

if \_\_name\_\_ == '\_\_main\_\_':  
 output=glossary(filespath**,**extentions)  
 dbinsertfunction(dbname**,** collectionname**,** filespath)  
 pd.DataFrame(output).to\_excel("outputs\\glossary.xlsx"**,** index=False)  
 json.dump(output**,** open('outputs\\glossary.json'**,** 'w')**,** indent=**4**)  
 glossary(filespath**,**extentions)

code to create json file and excel file for output.