**LOW LEVEL DOCUMENTATION**

**RULES REPORT**

BHAVYA JS

1.Create database connection.

client=MongoClient("localhost",port=27017)  
dbname=config.dbname  
collectionname=config.brecn

2. Function to fetch all files.

def getallfiles(self): # function to return the file paths in the given directory with given extention  
 filespath = config.filespath  
 extensions = [".jsp", ".java", ".css", ".js"]  
  
 filelist = []  
 for root, dirs, files in os.walk(filespath):  
 for file in files:  
 if file.lower().endswith(tuple([item.lower() for item in extensions])):  
 filelist.append(os.path.join(root, file))  
 return filelist

3. Function to fetch the component type of file.

def getcomponenttype(self,file):  
 *"""  
 this function is to find extension type for files* ***:param*** *filename: passing filename to fetch extension type* ***:return****: we are returning extension type based on file type  
 """* colname = config.componenttypecn  
 col = client[dbname][colname]  
 component\_type = list(col.find({"component\_name":file.split('\\')[-1]}, {'\_id': 0}))  
 component\_type = component\_type[0]['component\_type']  
 return component\_type

4. Function to create dictionary for each function for a given file.

def createdict(self,file):  
  
 *"""This function reads all files and captures each function from the files  
 and creates dictionary for each function with key as function name and value as  
 list of lines in that function* ***:param*** *filename:passing file name to get the dictionary of functions* ***:return****:different dictionary will be returned for each function  
 """* functionname=""  
 flag=False  
 flag1=False  
 count=0  
 funlist=[]  
 dictionary={}  
 f=open(file,"r")  
 fi=file.split("\\")[-1]  
 if fi.endswith (".java"): ##opens files of java extension and runs through the functions.  
 for line in f.readlines():  
 if (line.strip().startswith("protected") or line.strip().startswith("public") or line.strip().startswith("private")) and line.strip().\_\_contains\_\_("(") and line.strip().\_\_contains\_\_(")"):  
 functionname=line.split("(")[0].split()[-1]  
 flag=True  
 if flag and line.strip().\_\_contains\_\_("{"): ## count is used inorder to capture the entire function from start till end eventhough "{" found inbetween  
 count+=1  
 flag1 = True  
 if flag and line.strip().\_\_contains\_\_("}"):  
 count-=1  
  
 if flag and flag1 and count==0:  
 flag=False  
 flag1=False  
 funlist.append(line)  
 dictionary[functionname]=copy.deepcopy(funlist) ## this creates dictionary with key function name and value as list of lines.  
 funlist.clear()  
 if flag:  
 funlist.append(line) ##when flag is true it appends lines to list  
 return dictionary

5. creates conditions list to store all lines of conditional statements.

for item in f: ##iterating through dictionary  
 for line in f[item]: ##reading each line from each dictionary  
  
 if (line.strip().startswith("protected") or line.strip().startswith("public") or line.strip().startswith("private")) and line.strip().\_\_contains\_\_("(") and line.strip().\_\_contains\_\_(")"):  
 functionname = line.split("(")[0].split()[-1]  
  
 if line.strip().startswith("default") or line.strip().startswith("if") or line.strip().startswith("else if") or line.strip().startswith("else") or line.strip().startswith("while") or line.strip().startswith("for") or line.strip().startswith("switch") or line.strip().startswith("case") or line.strip().startswith("do "):  
 flag = True  
 if line.strip().endswith("\*/"): ##ignoring block comments and single line comments  
 comment\_flag = False  
 flag=False  
 continue  
 if line.strip().startswith("//"):  
 comment\_flag=False  
 continue  
 if line.strip().startswith("/\*"):  
 comment\_flag = True  
  
  
  
 if comment\_flag==False and flag and line.strip().\_\_contains\_\_("{"): ##captures entire conditional statement block  
 condition\_count += 1  
 flag1 = True  
 if comment\_flag==False and flag and line.strip().\_\_contains\_\_("}"):  
 condition\_count -= 1  
 if comment\_flag==False and flag and flag1 and condition\_count == 0:  
 condition\_list.append(line.replace("\t", " "))  
 flag = False  
 flag1 = False  
 if flag and comment\_flag==False:  
 condition\_list.append(line.replace("\t", " "))

6. reads the condition list and finds count

if not flag and not flag1 and condition\_list!=[]: ##iterating through list again and finding count of conditional statements  
 if\_count = 0  
 for line in condition\_list:  
 if line.strip().startswith("default") or line.strip().startswith("if") or line.strip().startswith("else if") or line.strip().startswith("else") or line.strip().startswith("while") or line.strip().startswith("for") or line.strip().startswith("switch") or line.strip().startswith("case") or line.strip().startswith("do "):  
 if\_count += 1 ##after reading the condition list we get the count of th conditional statements

7. creates json when count is one.

if not flag and not flag1 and condition\_list!=[]: ##iterating through list again and finding count of conditional statements  
 if\_count = 0  
 for line in condition\_list:  
 if line.strip().startswith("default") or line.strip().startswith("if") or line.strip().startswith("else if") or line.strip().startswith("else") or line.strip().startswith("while") or line.strip().startswith("for") or line.strip().startswith("switch") or line.strip().startswith("case") or line.strip().startswith("do "):  
 if\_count += 1 ##after reading the condition list we get the count of th conditional statements

8. If count is two separates the nested statements and creates json for each conditions.

else: ##if if\_count becomes 2 then splits all the nested conditons and adds to seperate json  
 parent\_rules = []  
 for line in condition\_list:  
  
 if line.strip().startswith("default") or line.strip().startswith("if") or line.strip().startswith("else if") or line.strip().startswith("else") or line.strip().startswith("while") or line.strip().startswith("for") or line.strip().startswith("switch") or line.strip().startswith("case") or line.strip().startswith("do "):  
 count += 1  
 brac\_flag = True  
  
 if brac\_flag and line.strip().\_\_contains\_\_("{"):  
 brac\_counter += 1  
  
 if brac\_flag and line.strip().\_\_contains\_\_("}"):  
 brac\_counter -= 1  
 if brac\_counter == 0:  
 conditions = conditions + line.strip().replace("\t", " ")+"\n"  
 brac\_flag = False  
 condition\_list.clear()  
  
 if count == 2 or line.strip().\_\_contains\_\_("}"):  
 if conditions != "":  
  
 """the below code creates json and also generates parent rule id for each conditions.  
 1.parent rule id remains same as rule id for first conditional statement  
 2.if second nested condition is found it appends the current rule id to the exsisting parent rule id  
 3.if third nested condition is found it appends the current rule id to the exsisting parent rule id   
 4.if the third nested condition ends then it removes its rule id from the parent rule id.  
 5.similarly for second and first conditions."""  
  
 if conditions.startswith("if") or conditions.startswith("else if") or conditions.startswith("else") or conditions.startswith("switch") or conditions.startswith("case") or conditions.startswith("default") or conditions.startswith("while") or conditions.startswith("do") or conditions.startswith("for"):  
 countRule += 1  
 Rule="RULE-" + str(countRule)  
 else:  
 Rule=""  
  
 if conditions.startswith("}") and (conditions.\_\_contains\_\_("else") or conditions.\_\_contains\_\_("case") or conditions.\_\_contains\_\_("if") or conditions.\_\_contains\_\_("switch") or conditions.\_\_contains\_\_("else") or conditions.\_\_contains\_\_("for") or conditions.\_\_contains\_\_("else if") or conditions.\_\_contains\_\_("switch") or conditions.\_\_contains\_\_("while") or conditions.\_\_contains\_\_("do")):  
 parent\_rules.pop()  
 parent\_rules.append("RULE-" + str(countRule))  
  
 elif conditions.startswith("}")or conditions.endswith("}") or conditions.endswith("break"):  
 if conditions.endswith("break"):  
 parent\_rules.pop()  
 parent\_rules.append("RULE-" + str(countRule))  
 else:  
 parent\_rules.pop()  
  
 else:  
 parent\_rules.append("RULE-" + str(countRule))  
  
 parent = str(parent\_rules)  
  
 output = ({"s\_no": "", "component\_name": file.split("\\")[-1],  
 "component\_type": self.getcomponenttype(file),  
 "Function\_name":functionname,  
 "Rule\_Id": Rule,  
 "Rule\_statements": conditions,  
 "parent\_rule\_id": (parent.replace("[", "").replace("]", "")),  
 "statement\_group": "",  
 "rule\_category": "",  
 "business\_documentation": "",  
 "rule\_description": ""})  
 if conditions.\_\_contains\_\_("default") and conditions.\_\_contains\_\_("break"):  
 parent\_rules.pop()  
  
 METADATA.append(copy.deepcopy(output))  
 conditions = ""  
 output.clear()  
  
 if line.strip().startswith("if") or line.strip().startswith("else if") or line.strip().startswith("else") or line.strip().startswith("while") or line.strip().startswith("for") or line.strip().startswith("switch") or line.strip().startswith("case") or line.strip().startswith("do "):  
 count = 1  
 else:  
 count = 0  
  
 if count == 1 or (brac\_counter != 0 and brac\_flag):  
 conditions = conditions +line.strip().replace("\t", " ")+"\n"  
return METADATA

9. Function to insert into database.

def dbinsertfunction(self,filespath, dbname, collectionname):  
 *"""  
 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  
 """* col = client[dbname][collectionname]  
 output = self.getallreports()  
 if output != []:  
 if col.count\_documents({}) != 0:  
 col.drop()  
 print("Deleted the old", dbname, collectionname, "collection")  
  
 col.insert\_one({"type": "metadata",  
 "headers": ["s\_no","component\_name", "component\_type", "Function\_name","Rule\_Id",  
 "Rule\_statements","parent\_rule\_id","statement\_group","rule\_category",  
 "business\_documentation","rule\_description"]})  
 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)

10. Create excel and function call .

if \_\_name\_\_ == '\_\_main\_\_':  
 # output = getallreports(filespath)  
 # if not os.path.exists("outputs//"):  
 # os.makedirs("outputs//")  
 # #json.dump(output , open('outputs\\variable\_impact report.json', 'w'), indent=4)  
 # pd.DataFrame(output).to\_excel("outputs\\variable\_impact report.xlsx", index=False)  
 reports=bre()  
 reports.dbinsertfunction(filespath, dbname, collectionname)