**Master Inventory Documentation**

1. Read the Folders the **filepath** given as input
2. Read folders of different component types one by one
3. Open RPG folder first and start reading the RPG files one by one and capture the master inventory details for each file
4. Capture the component name and type
5. Count the total Lines Of Code
6. Count blank lines
7. Count the comment lines

* RPG is having two different types of comments

1. Containing the “ \* ” at 7th character of the line.
2. Containing “// ” in lines (Free Format)

* CL file comments pattern

1. Characters between “/\*” and “\*/” are comments in CL files.
2. We can have commented characters and functional code in the same line.
   * + Example - **/\* VS01 \*/** PGM PARM(&NBRSTRMC &FIRSTRRNC &RCDCOUNTC)
3. Count the SLOC

* To find SLOC subtract Black lines and Comment lines from the LOC

1. Once after calculating/capturing all the required details, pass the values to web service.

Using (requests.post('http://localhost:5001/api/v1/update/masterInventory')

1. Along with details pass the headers information to the Web services in Json format

( requests.post('http://localhost:5001/api/v1/update/masterInventory', json={"data":METADATA,"headers":["component\_name","component\_type","Loc","commented\_lines","blank\_lines","Sloc","application","no\_of\_dead\_lines", "cyclomatic\_complexity","dead\_para\_count","total\_para\_count"]}) )

1. Repeat step 3 to 10 for other Folders which are meant for different file formats and update the data into Master table

**CRUD Report**

1. Read the Folders the **filepath** given as input
2. Read folders of different component types one by one
3. Open RPG folder first and start reading the RPG files one by one and capture the CRUD related details for each file .
4. Skip reading the Comment Lines.
5. Capture the component name and type
6. Capture the DB2 related queries from the file using keywords ([**write** for Create],[‘**setll', 'setgt', 'read', 'readp', 'readpe', 'reade', 'chain'** for Read ],[**'update**' for Update] , **[‘delete'** for Delete].
7. Read the lines for above mention keywords and capture the function name next to it.
8. Once after Capturing all the details Update the mongo DB using

* db.crud\_report.insert\_many(METADATA) metadata here is CRUD details in each file

1. METADATA contains values of component\_name, component\_type,Table,SQL,CRUD

* component\_name : input filename , component\_type:input file type

Table : table name,SQL:DB2 Keywords like (write,setll,settgt,read,update,delete,etc.)

CRUD: Type of db2 keyword(Create,read,update,delete)

1. Along with CRUD details upload Headers to the mongo DB using

* db.crud\_report.insert\_one({"type": "metadata", "headers": [ "component\_name", "component\_type","Table","SQL","CRUD"]})

1. Repeat step 3 to 10 for other Folders which are meant for different file formats and update the data into CRUD Collection in DB.

**Process Flow and Dead Code**

1. Read the Folders the **filepath** given as input.
2. Read folders of different component types one by one.
3. Open RPG folder first and start reading the RPG files one by one and capture the Dead code and process flow related details for each file.
4. Expand the Copy Books for the input files.

* Read the Folder of copybooks
* Read the in input file for the Keyword /COPY
* Capture the Copy name after Copy Keyword
* Copy the lines of code from the file having name as captured copy keyword
* Past those lines in the file at the position where copy word was found
* Ignore the comment lines and lines having “ d ”at the 7th position while reading.

1. Read the lines of expended copy file one by one and capture the subroutines.

* Sub routine starts with begsr and it will ends with endsr.
* We have Two different formats of Sub routines
  + - * + Fixed Format pattern : “Sub Routine name” begsr
        + Free Format pattern : begsr “Sub Routine name”
* Capture the subroutine names.

1. Read the lines to capture subroutine call, Sub routines are called using exsr key word in AS400.
2. Generate the process flow

* Search for called sub routine (exsr Subroutine name or cas option subroutine name ).
* Compare it against sub routine definition (begsr Subroutine name ).
* if sub routine definition is found read the subroutine definition for any other subroutine call if found go read that definition also continue the process until no other function calls are found
* After reaching the last subroutine call mark the starting sub routine definition as parent and subroutines calls found inside as children.
* All the subroutines captured in the above step is considered as Active subroutines.

1. Find the dead sub routines, if a subroutine is defined and not called any anywhere in the programme then that sub routine is considered as dead.

* If a sub routine is defined using begsr and not called using exsr or cas then the subroutine is dead.
* In order to find the dead sub routine we need to find the subroutines which are alive
* Difference between All subroutines captured in step 6 and Active subroutines in step 7 is Dead subroutines

1. Find the dead code related details like LoC of code of each dead sub-routine is identified and summed up with all dead sub-routine lines to get total deadlines. Total number of dead subroutines, list of dead para names , count of total number of subroutines.
2. Update the dead code related details found in step 9 into master inventory using below mongo db query.

* db.master\_inventory\_report.update\_many()

1. Create the process flow based on the 7th step pass it to the web service in json format using
   * requests.post('http://localhost:5001/api/v1/update/procedureFlow'
2. Delete the expanded copy file
3. Repeat step 4 to step 11 for all the files in RPG folder.

**Missing Report**

1. If a component is **called** (referenced) by one or more components in the cross reference report, but not present in the master inventory, then it is called as Missing component.
2. Take the called component list and called type and compare it against “component name and type” in Master inventory list . If called component is NOT present in Master inventory then the component is considered as Missing component.
3. Fetch data from cross ref table from the data base using

* distinctIdCode = { "$group": { "\_id": { "called\_name": "$called\_name", "called\_type": "$called\_type" } } }
* res = db. cross\_reference\_report.aggregate([distinctIdCode])

1. Fetch data from Master inventory table from database using

* distinctIdCode = { "$group": { "\_id": { "component\_name": "$component\_name", "component type": "$component type" } } }
* res = db.master\_inventory\_report.aggregate([distinctIdCode])

1. Data from Cross reference table is compared against the data from Master inventory table.

* perform subtraction between Cross reference list and Master inventory list
* Resultant list from the subtraction is required Missing List.

1. Insert headers using

* db.missing\_components\_report.update\_one({"type": "metadata"}, {"$set": {"last\_updated\_on": current\_time,"time\_zone": time\_zone, "headers": ["component\_name", "component\_type"],

"script\_version":SCRIPT\_VERSION }}, upsert=True).acknowledged:

1. Insert missing data to mongo db along with headers using

* db.missing\_components\_report.insert\_many()

**Orphan Report**

1. If a component is **present in** master inventory, but not called (referenced) by any of the components in cross reference report , then it is called as Orphan component.
2. - Take the component list and Type from Master inventory and compare it against “called name and type” in Cross reference report. If the component is NOT present in the “called” column of cross reference report, then the component is considered as Orphan component.
3. Fetch data from cross ref table from the data base using

* distinctIdCode = { "$group": { "\_id": { "called\_name": "$called\_name", "called\_type": "$called\_type" } } }
* res = db. cross\_reference\_report.aggregate([distinctIdCode])

1. Fetch data from Master inventory table from database using

* distinctIdCode = { "$group": { "\_id": { "component\_name": "$component\_name", "component type": "$component type" } } }
* res = db.master\_inventory\_report.aggregate([distinctIdCode])

1. Data from master inventory table is compared against the data from Cross reference table.

* perform subtraction between Master inventory list and cross reference list
* Resultant list from the subtraction is required orphan list.

1. Insert headers using

* db.orphan\_report.update\_one({"type": "metadata"}, {"$set": {"last\_updated\_on": current\_time,"time\_zone": time\_zone,"headers": ["component\_name", "component\_type"],"script\_version":SCRIPT\_VERSION }}, upsert=True).acknowledged:

1. Insert Orphan component to data base using.

* db.orphan\_report.insert\_many().

**Control Flow:**

1. Capture the request from Front-End using Get Method and Flask.
2. Request from Front end contains **Component Name** and **Component Type.**
3. Store the Component Name and Component Type in a Dictionary called **response.**
4. Insert the flowing keys to Dictionary **Component Name** ,**Component Type,links,nodes.**
5. Define the function called **db\_link(input\_name,input\_type)**  to fetch data from Cross-ref**.**
6. Pass the component name and component type to the db\_link function.
7. Using the below mongo Query we can fetch the Cross reference repot.

**connections = db.cross\_reference\_report.find({"$or": [**

**{"$and": [{"component\_name": input\_name}, {"component\_type": input\_type}]},**

**{"$and": [{"called\_name": input\_name}, {"called\_type": input\_type}]}]}, {'\_id': 0})**

1. Above Query will fetch all json’s in cross-ref collection.
2. Use for loop to iterate over the record fetched from cross-ref.
3. In response[nodes] list add all the nodes.
4. Using if else condition add the links to response[links].
5. Define the function called **db\_link2(input\_name,input\_type)**  to fetch data from Crud**.**

**connections = db.crud\_report.find({"component\_name" : input\_name})**

1. Use for loop to iterate over the record fetched from Crud.
2. In response[nodes] list add all the nodes.
3. Using if else condition add the links to response[links].
4. Return the Response dictionary.

**Glossary**

1. Read the Folders the **filepath** given as input.
2. Read folders of different component types one by one.
3. Open RPG folder first and start reading the RPG files one by one and capture the Glossary related details for each file.
4. Skip the comment lines in the each file.
5. Capture line starting with “**D**” or “**d**”
6. Split the line using Split() function and capture the variable in first index or second index.
7. Append the variables to the list.
8. Using **for** loop form the json for each and every variable.
9. Json should in formed in the below formate.
10. Glossary\_json = {"component\_name": component\_name, "Variable":var, "Business\_Meaning": BM}
11. Append all the json’s to a list and insert it into mongo db along with headers.

**Rules Report**

1. Fetch the BRE data using bellow DB Query.
2. details = collection.find({"$and":[{"parent\_rule\_id" :{"$ne": ""}},

{"source\_statements": {"$ne": "endif"}},

{"source\_statements": {"$ne": "select"}},

{'type': {"$ne": "metadata"}},

]}).sort("\_id",1)

1. Write the BRE data to Excel.
2. After writing the data to excel perform below operations on excel using xlrd framework.
3. Open excel and read the cell values in column 4 i.e parent rule id column.
4. Merge the cells having similar parent rule id. Along with the other columns data.
5. Assign the new rule id to merged data.
6. From the json in the below format
7. {

"fragment\_id" : "1",

"pgm\_name" : "IAC001",

"para\_name" : "œinit",

"source\_statements" : "if \*inu8 = \*on <br> eval \*in68 = \*on <br> else <br> eval \*in68 = \*off <br> ",

"rule\_category" : "",

"rule\_description" : "",

"Rule" : "Rule-0001",

"rule\_relation" : ""

}

1. Append all the json’s to a list .
2. Insert that list to the mongodb.

**Commented lines Report**

1. Read the Folders the **filepath** given as input.
2. Read folders of different component types one by one.
3. Open RPG folder first and start reading the RPG files one by one and capture the Glossary related details for each file.
4. Skip the Empty lines in the each file.
5. Capture the line which is having **“\*”** at the 6 the position or lines starts with “**//**”
6. Replace /n with <br> in captured line.
7. Form the json for each line with below keys.
8. {'component\_name': eachFile.strip(file\_location).replace(" ","\_").strip('.\*'), 'component\_type': file\_type.strip("\*."), 'codeString': god\_string}
9. Append all the json’s to a list .
10. Insert above list to the mongodb.