

# **File System Manager**

**Un mod mult mai elegant de a copia sau muta fișiere,  
astfel reușind să le manipulăm într-o manieră  
mult mai complexă și detaliată.**

**by**

**Simofi Robert**  
**Informatică Romana**

## Cuprins:

1. Descrierea problemei.....	
2. Separarea codului.....	
3. Funcționalitatea.....	
3.1. Descriere.....	
3.2. Cazuri de utilizare.....	
4. Interfața grafică.....	
5. Integrarea scriptului cu interfața grafică.....	
6. Exemple de utilizare.....	
7. Planuri de îmbunătățire.....	

## 1. Descrierea problemei

Deseori, cei mai mulți utilizatori ai calculatorului, în special cei cu sistemul de operare Windows, se lovesc de problema mutării sau copierii fișierelor. De multe ori se ajunge la parcurgerea a multor directoare, totul făcându-se manual, astfel pierzând mult timp și energie pe tot parcursul. Totodată se poate ivi ocazia de a filtra anumite fișiere dintr-un loc, de a copia o structură avansată de directoare, de a sustrage din mai multe directoare fișierele într-un singur director destinație, de a dori să se extragă doar fișiere cu o anumită extensie (sau mai multe extensii), etc...

Ei bine, File System Manager (FSM) poate realiza aceste lucruri într-o manieră minimalistă, fiind ușor de folosit și înțeles.

## 2. Separarea codului

Aplicația este alcătuită din două module principale, acelea fiind partea interfeței grafice și partea funcționalității.

Dezvoltarea a început cu partea funcțională, ea fiind defapt un simplu fișier de tip python script, numit "FSM.py". Acest script este construit astfel încât să poată fi folosit și de sine stătător, are propria ei secție de validare și folosește argumentele primite ca și date de intrare.

A doua componentă principală a aplicației este partea interfeței grafice, ea fiind realizată în Visual C++, cu ajutorul claselor MFC. Interfața grafică are rolul de a-i ascunde utilizatorului toată partea funcțională a aplicației și de a-i crea ușurință în utilizare. Nu cuprinde deloc părți funcționale, ci doar evenimentele de tratare a datelor de intrare pe care utilizatorul vrea să le selecteze. Astfel, cele două componente sunt total independente și oricând apare anumită schimbare într-o parte, cealaltă nu va fi afectată.

### 3. Funcționalitatea

#### 3.1. Descriere

Funcționalitatea aplicației este integrată în fișierul “FSM.py”. Este un script python care a fost conceput cu scopul de a gestiona directoare și fișiere. Scriptului i se predau următoarele argumente pentru procesare:

1. **Directorul sursă** – specifică calea către directorul de la care se va începe copierea sau mutarea.
2. **Directorul destinație** – specifică calea către directorul la care se va face copierea sau mutarea.
3. **Caută în subdirectoare** – opțiune de a alege dacă se scanează fișierele recursiv în toate subdirectoarele, cu rădăcina la sursă, pentru a fi copiate sau mutate.
4. **Reconstruiește structura directoarelor** – utilizatorul poate alege între a avea toate fișierele într-un singur director sau dacă dorește să i se reconstruiască toată structura directoarelor, după mutare sau copiere (dacă se alege această opțiune, de fiecare dată se vor scana fișierele din toate subdirectoarele).
5. **Mutare sau copiere** – utilizatorul poate decide dacă vrea să copieze sau să mute fișierele de la sursă la destinație
6. **Numele noului director destinație** – de fiecare dată când se utilizează scriptul, se crează un nou director în care vor fi puse fișierele
7. **Extensii** – toate extensiile fișierelor pe care utilizatorul vrea să le manipuleze vor fi specificate la sfârșit, astfel scriptul fiind capabil să ruleze cu foarte multe extensii / argumente (în funcție de sistemul de operare). Totodată, scriptul poate rula cu toate tipurile de fișiere deodată, astfel nespecificând nici o extensie.

La începutul rulării scriptul face o validare proprie pentru a testa numărul corect de argumente, existența directoarelor sursă și destinație și faptul că nu există alt director în destinație care să aibă numele noului director în care vor fi puse fișierele după operație. Aceste validări sunt necesare pentru a putea rula scriptul și din linia de comandă, astfel făcându-l independent de interfața grafică și oferind ușurință la dezvoltare, testare și mentinere.

Se folosesc modulele python **sys** pentru comunicarea cu sistemul de operare pe care rulează și anumite comenzi, cum ar fi “sys.argv” pentru argumente sau “sys.exit()” pentru încetarea rulării la comandă; **os** pentru manipularea directoarelor și a fișierelor, cu ajutorul comenzilor de tip “os.path.isdir(path)” pentru a verifica dacă o cale este director sau “os.walk” pentru parcurgere recursive și în final **shutil** pentru copierea fișierelor, cu ajutorul comenzilor “shutil.copy2(src, dst)” pentru a copia fișierul de la calea sursă în directorul de la calea destinație sau “shutil.copytree(src, dst)” pentru copierea recursivă a tuturor directoarelor și fișierelor de la src la dst.

### 3.2. Cazuri de utilizare

Scriptul a fost conceput pentru a rezolva multe probleme întâmpinate la mutarea sau copierea fișierelor. Cele mai comune cazuri de utilizare, împreună cu setările respective, sunt, de exemplu:

- dacă se dorește mutarea unei structuri întregi de directoare, se specifică directorul rădăcină al sursei, faptul că se dorește reconstruirea structurii de fișiere și nu se specifică nici un argument, pentru a-i permite scriptului să scaneze toate fișierele.
- dacă se dorește filtrarea unor anumite fișiere (de exemplu muzică) dintr-un loc în altul, se specifică extensiile respective (în cazul de față .mp3, .wav, .flac, etc).
- dacă se dorește copierea unor fișiere într-un singur director, de exemplu când vrem să obținem toate fotografiile din celebrul director “poze”, alegem ca scriptul să parcurgă și în subdirectoare, dar să nu reconstruiască structura directoarelor, ci doar să le pună în folderul nou destinație, opțiunea de copiere sau mutare fiind la alegerea utilizatorului.

Scriptul este capabil să detecteze și fișierele duplicate (de exemplu fișierul “test.exe” se află și în directorul “A”, și în “B”. De aceea se va copia doar prima variantă, a doua nu se va mai suprascrie și se va genera un fișier numit “duplicates.txt” în care vor fi specificate fișierele duplicate.

## 4. Interfața grafică

Interfața grafică a acestei aplicații este realizată în Visual C++, cu ajutorul claselor MFC. Interfața a fost realizată în IDE-ul Visual Studio. S-a început cu creerea unui dialog, deoarece aplicația este dialog-based. S-a creat un șablon simplu, generat de Visual Studio, pe care, ulterior, s-a continuat dezvoltarea. S-a început cu aranjarea componentelor pe dialog, cum ar fi butoane, căsuțe de text editabile, butoane radio, etc.

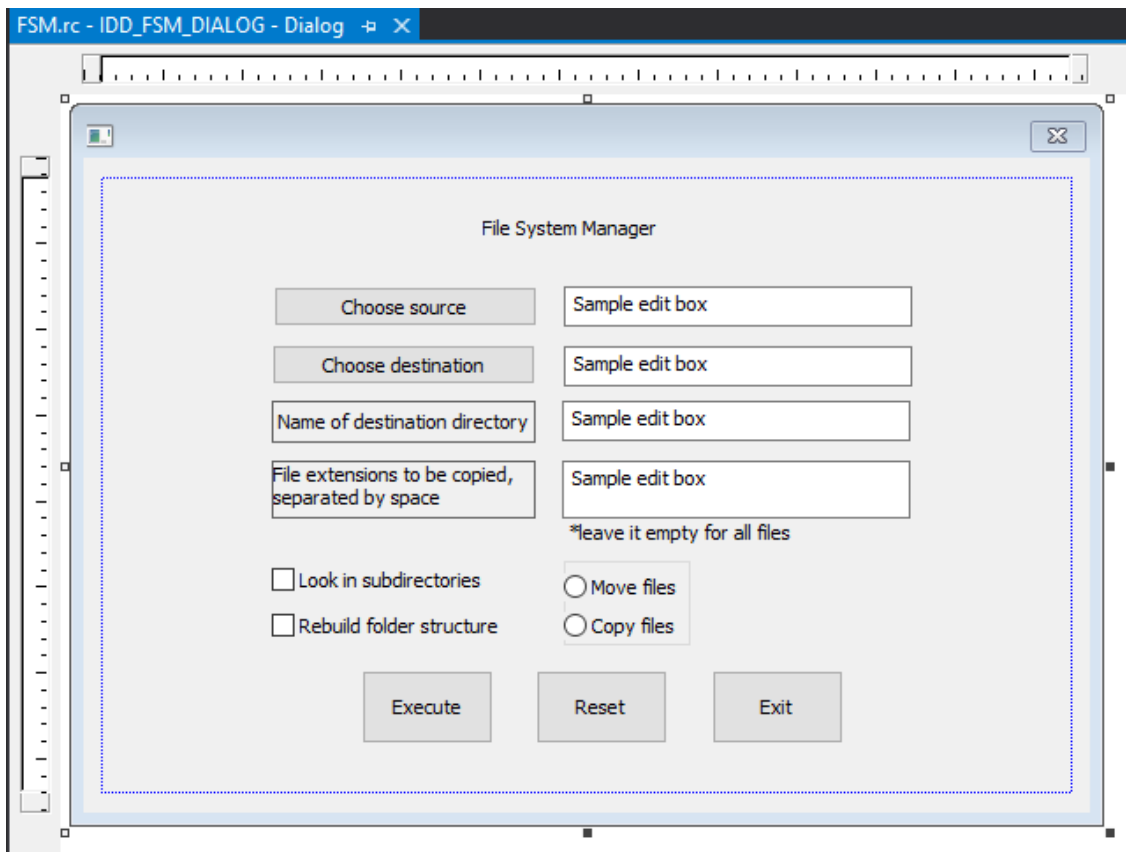


Figura 1: aranjarea elementelor pe dialog

Ulterior s-au asigat variabile de control pentru fiecare secțiune în parte, pentru a putea manipula în runtime conținutul dialogului și a-l putea face capabil să ruleze ciclic, atâta timp cât utilizatorul are nevoie să facă operații.

S-a ales controlul bazat pe evenimente, care este standardul aplicațiilor de acest tip. Acest control are o funcție specială, numită „DoDataExchange()”, care permite schimbul de date dintre utilizator și aplicație.

S-a asignat ulterior acțiunea fiecărui buton în parte. Pentru butoanele “Choose source” și “Choose destination” se afișează selectorul de directoare, care este implementat în clasa standard “CFileDialog”.

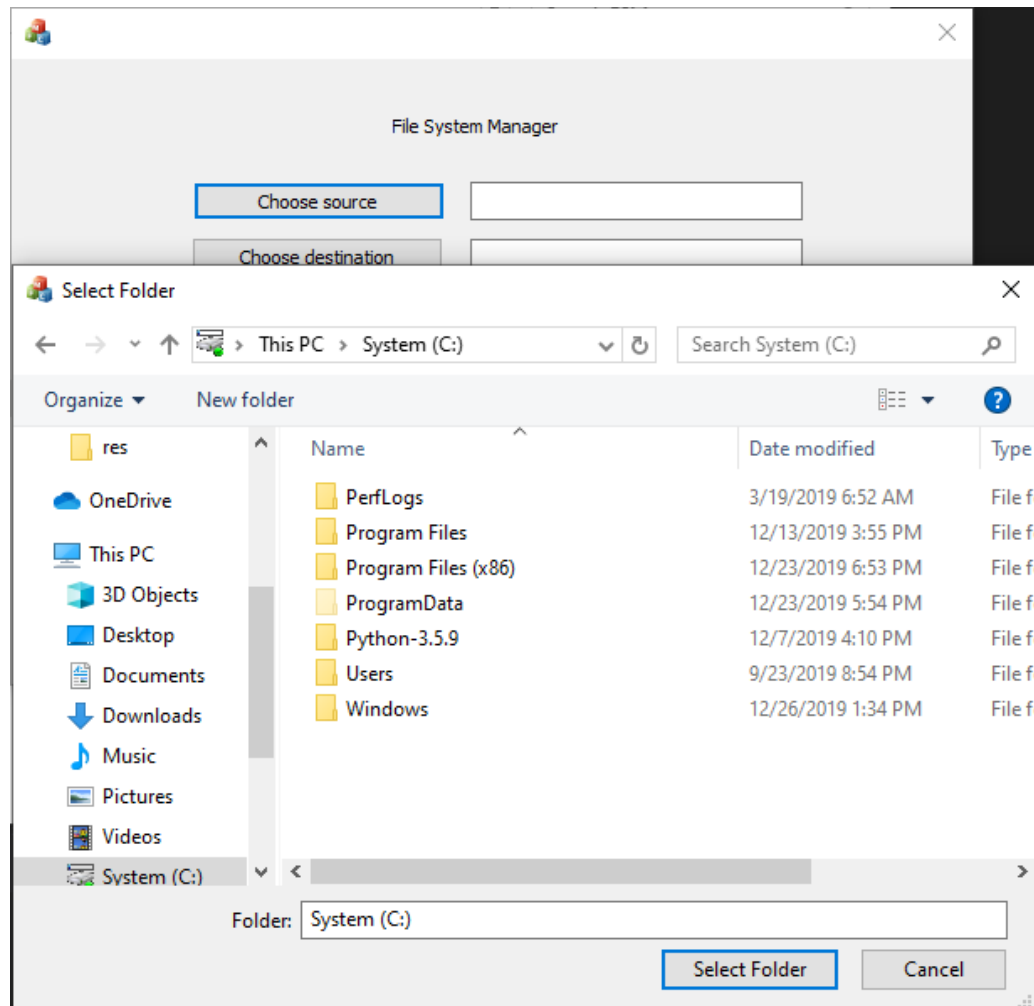


Figura 2: afișarea selectorului de folder cu butoanele respective

Pe urmă s-a trecut la setarea butoanelor check. Valorile butoanelor “Look in subdirectories” și “Rebuild folder structure” sunt obținute cu ajutorul metodei “GetCheck()”. La fel și butoanele radio “Move files” și “Copy files” au aceeași abordare, dar ele aparțin unui “Group box control” pentru a asigura selecția doar unei variante dintre cele două.

Butonul “Execute” este cel mai important din toată funcționalitatea interfeței grafice. El face validările datelor de intrare, verifică dacă sunt completate bine, așa cum cere și specificația. Aici se formatează argumentele ce sunt pasate către apelul sistem, și totodată butonul

lansează execuția scriptului cu argumentele primite din dialog. După aceea se returnează un mesaj de informare despre starea succesului execuției.

Butonul reset are rolul de a reseta tot conținutul dialogului și de a-l aduce în starea inițială, pentru a-i oferi utilizatorului o cale mai rapidă spre o nouă operație.

Butonul exit are rolul de a opri execuția aplicației, dar asta se poate realiza și din butonul “X”.

## 5. Integrarea scriptului cu interfața grafică

În cadrul acțiunilor executate de butonul “Execute” se află și integrarea funcționalității python în cadrul unui executabil C++. În limbajul C se află o instrucțiune foarte puternică numită system call. Se folosește instrucțiunea “\_popen(char \*command, char \*mode)” care, în contextul de față, “command” este alcătuit din path-ul către environmentul în care se va executa apelul, adică path-ul către python.exe, următorul argument fiind path-ul către fișierul scriptului python “FSM.py”, urmat de linia de argumente pentru script. Astfel, se realizează un apel PowerShell de acest tip, toată comunicarea dintre module realizându-se în fundal, prin std-ul bash-ului. Scriptul python are anumite secvențe de “print()”-uri, lucruri care sunt puse pe bufferul bash, astfel modulul C++ fiind capabil să extragă acele mesaje și de a le salva într-un buffer intern, numit “psBuffer”. Toată această comunicare are forma unui pipe. Atăta timp cât scriptul rulează, firul de execuție al componentei grafice este suspendat, iar la finalul rulării acesta-și dă “unlock”, având un sistem de mutex în spate. Astfel, nu este nevoie de a se crea obiecte de tip python, utilizând librăria python a limbajului C++, și nici nu este nevoie de integrare de tip embedded.

```
CString stringToCall = pythonLocation + " \"\" + scriptName + "\" \"\" + srcDir + "\" \"\" + dstDir + "\" \"\" +
    subDirString + " \"\" + rebuildString + " \"\" + deleteString + "\" \"\" + dstDirName + "\" \"\" + extensionList;

char sysCallString[1000];
sprintf(sysCallString, "%ls", stringToCall.GetBuffer());

//python integration
char psBuffer[128];
FILE* pPipe;

if ((pPipe = _popen(sysCallString, "rt")) == NULL)
    exit(1);

int returnValue = -1;
while (fgets(psBuffer, 128, pPipe)) {
    returnValue = psBuffer[0] - 48;
}

if (feof(pPipe))
    int closePython = _pclose(pPipe);
```

Figura 3: integrarea scriptului python



## 6. Exemple de utilizare

Aplicația se pornește din executabilul “FSM.exe”, la nivelul cu soluția “FSM.sln”, pentru a identifica toate obiectele codului, păstrând setările soluției pentru funcționare. Apoi va porni dialogul principal, în care utilizatorul va putea oferi date de intrare pentru execuție.

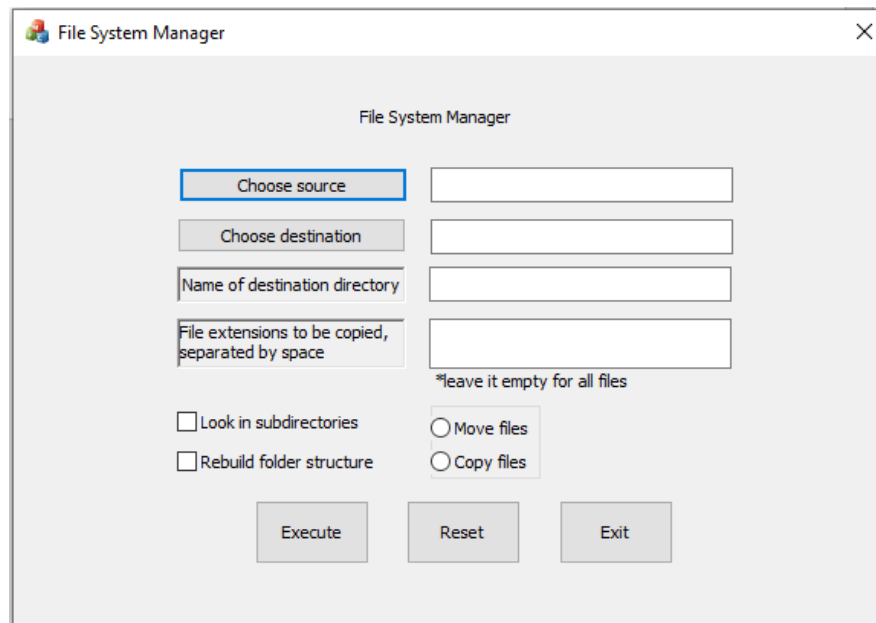


Figura 4: fereastra principală a aplicației

Ulterior, se pot alege directoarele sursă și destinație, cum s-a specificat la Figura 2. Dacă nu se completează bine toate datele de intrare, sau anumite câmpuri sunt lăsate goale, aplicația îi va transmite utilizatorului aceste lucruri, ca în exemplul de jos.

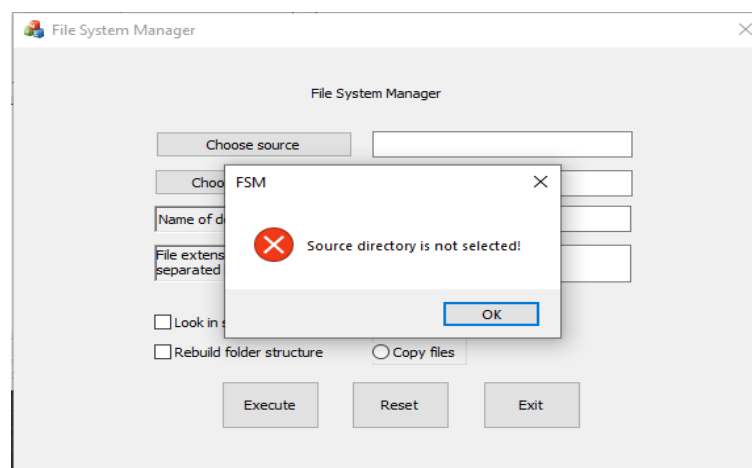


Figura 5: exemplu de validare a datelor de intrare.

Dacă datele sunt completate bine, se lansează apelul sistem către scriptul python, care la rândul ei va verifica și corectitudinea intrărilor.

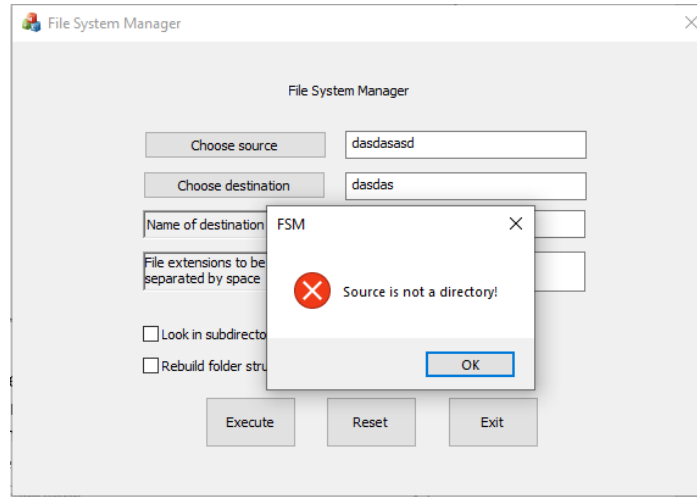


Figura 6: validare făcută de scriptul python

În exemplul anterior s-a introdus un director aleator la sursă și scriptul nu l-a putut găsi, astfel el fiind invalid.

În cazul în care toate datele sunt completate și validate, începe execuția. Pot fi combinate o grămadă de cazuri, cum ar fi extragerea unor fișiere .txt dintr-un set de directoare, de exemplu.

Se presupune că avem un fișier **"file1.txt"** în următoarele locații: **"D:\testFSM"** și **"D:\testFSM\dir1"**, iar fișierul **"file2.txt"** se află în directorul **"D:\testFSM\dir1\dir2"**. Dacă alegem să completăm datele de intrare în felul următor,

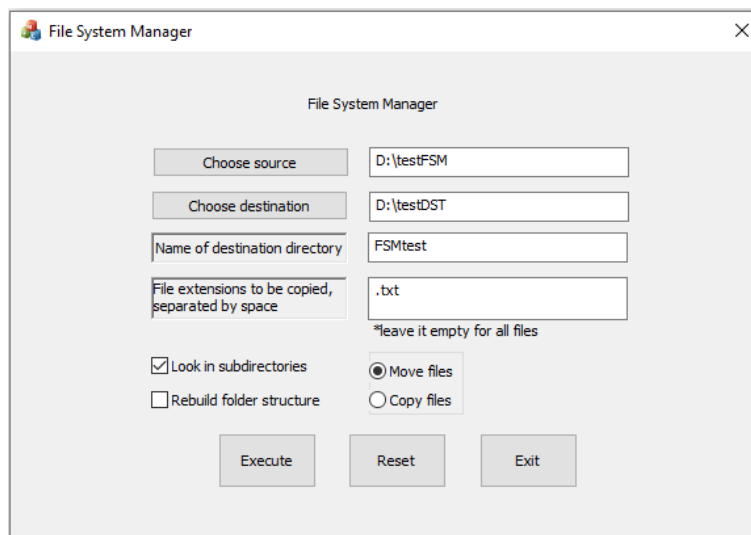
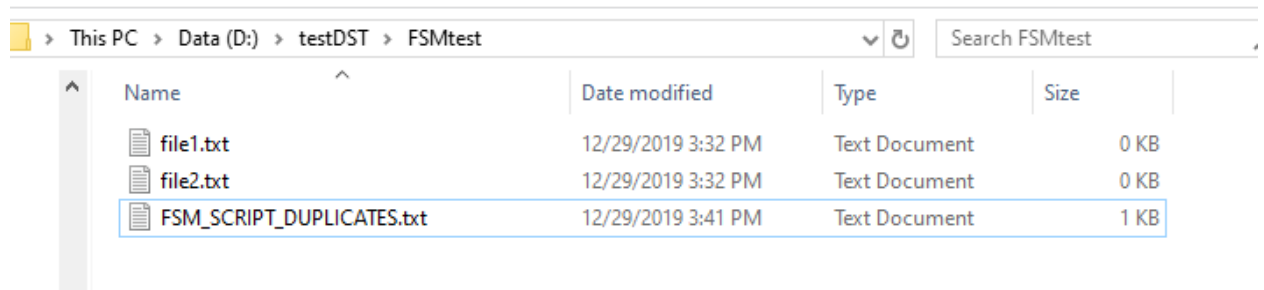


Figura 7: date de intrare pentru exemplul de sus

pentru a extrage toate fişierele cu extensia .txt din toate subdirectoarele cu rădăcina la directorul sursă, atunci în directorul “FSMtest” vom avea următoarele fişiere:



Name	Date modified	Type	Size
file1.txt	12/29/2019 3:32 PM	Text Document	0 KB
file2.txt	12/29/2019 3:32 PM	Text Document	0 KB
FSM_SCRIPT_DUPLICATES.txt	12/29/2019 3:41 PM	Text Document	1 KB

Figura 8: fişierele mutate în urma operaţiei

Intră în context fişierul “FSM\_SCRIPT\_DUPLICATES.txt”, generat de script, deoarece s-au găsit fişierele “file1.txt” în două directoare diferite, la scanare, așa că ele n-au fost mutate, ca să nu existe pierderi de date. În schimb, fişierul “file2.txt”, care a fost găsit doar o singură dată, a fost mutat și directorul din care a făcut parte, “dir2”, a fost șters, deoarece a rămas gol. Funcția de ștergere a directoarelor goale face parte din scriptul python, și se numește “clean(src)”.

## 7. Planuri de îmbunătățire

Principalele îmbunătățiri ce pot fi aduse acestei aplicații constă în:

- installer, care să pregătească toate componentele necesare rulării (Visual C++)
- build sistem
- interfață grafică mai complexă, care să fie mai atrăgătoare