

PROIECT
★Metode de dezvoltare
software ★
-Copy service-

Grupa 232
Membri:

- 1.Stoian Mihai**
- 2.Tesileanu Alexandru**
- 3.Pietroreanu George**
- 4.Schipor Teodora**

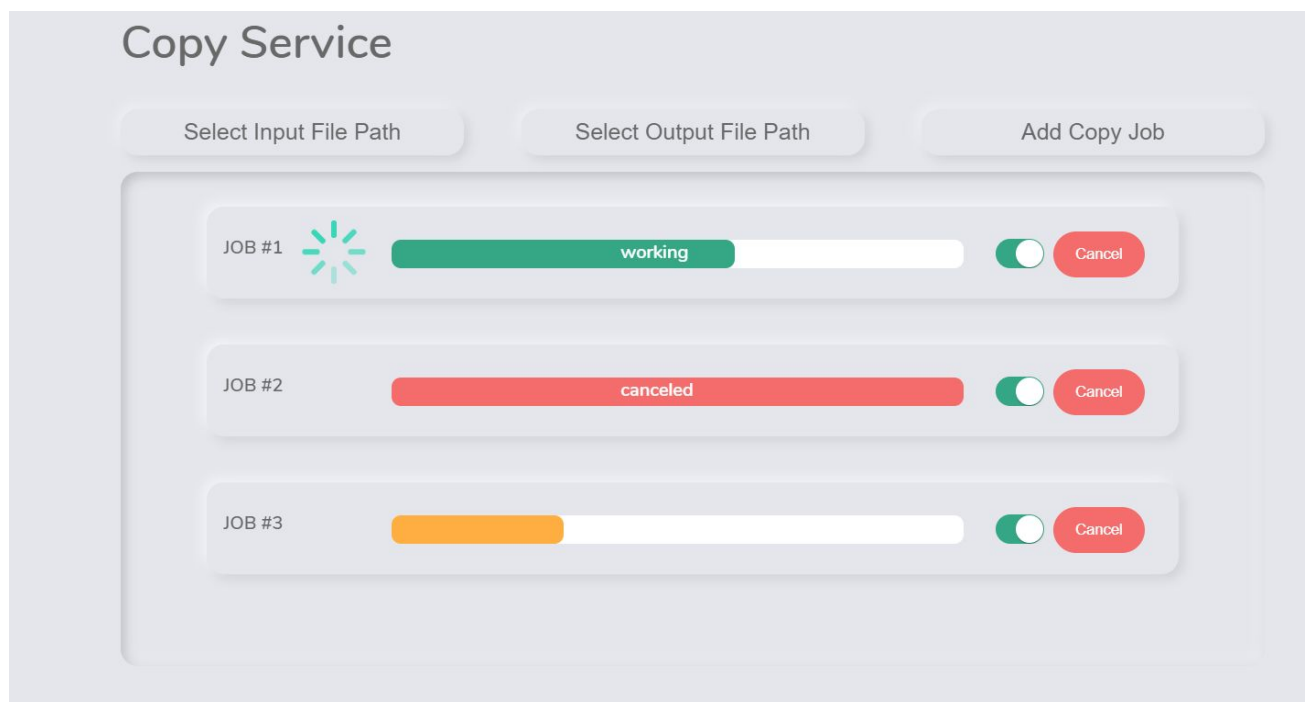
Cuprins

1. Introducere
2. Prezentarea aplicatiei
3. Impartirea task-urilor
4. Codul sursa
5. Design, arhitectura
6. Probleme intalnite
7. Source Control
8. Testare
9. Viitoare dezvoltari
10. Incheiere

1.Introducere

Primul target al proiectului a fost acela al alegerii temei, iar dupa o lunga sesiune de expus idei am avut de ales dintre a crea un site web pentru a ajuta locatarii cu masina dintr-un anumit cartier, putand sa puna locul lor de parcare la dispozitia altor persoane cand ramane liber si copy service. In cele din urma, punand cap la cap ce ne-ar ajuta mai mult pe viitor am ales sa implementam Copy Service-ul.

Proiectul Copy Service a fost creat folosind limbajele de programare C, JavaScript, Python si HTML + CSS oferind o interfata simplista cu care putem copia in paralel multiple fisiere pe medii Windows si PSOX.

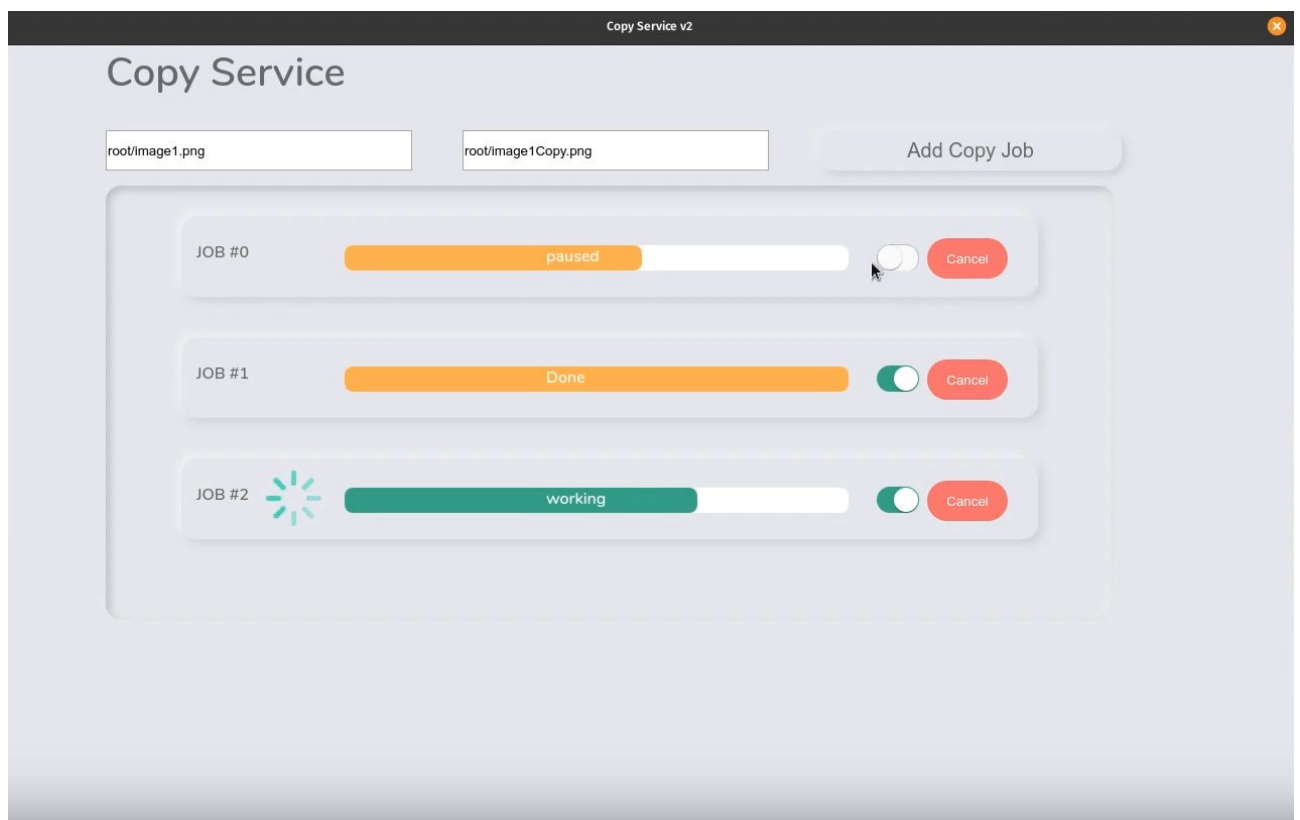


2. Prezentarea aplicatiei

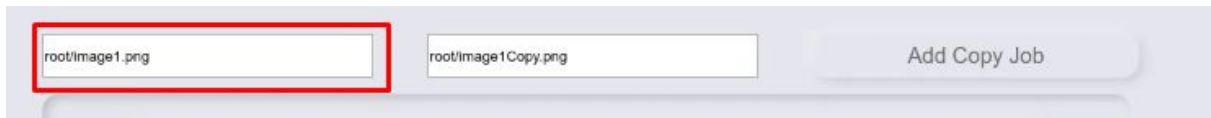
Aplicatia Copy Service realizeaza copiere de fisiere de la sursa la destinatie. Aceasta poate sa lucreze cu mai multe fisiere in acelasi timp. Este configurabila, deci user-ul isi poate alege cate sa copieze deodata.

Caracteristici:

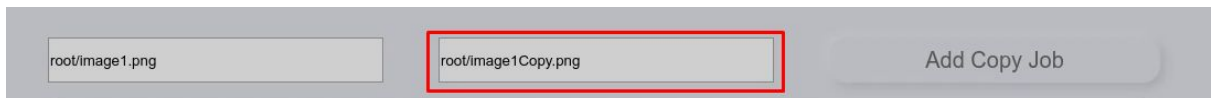
- **eficienta** deoarece lucreaza cu apeluri de sistem;
- **rapida** datorita faptului ca fiecare copiere se intampla pe un thread separat, deci un proces nu asteapta dupa altul decat daca sunt prea multe (atunci se efectueaza un pull de thread-uri);
- i-am adaugat si o **interfata grafica** simplista.



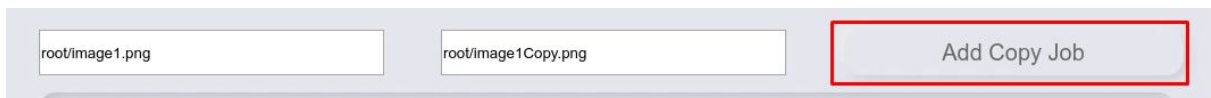
Ecranul principal al aplicatiei este unul gandit sa fie intuitiv, cu o incadrare si o tema de culori familiara utilizatorului.



În partea de sus a ecranului se afla zona destinată adăugării unui nou proces de copiere. Câmpul încercuit reprezintă calea fișierului care urmează a fi copiat.

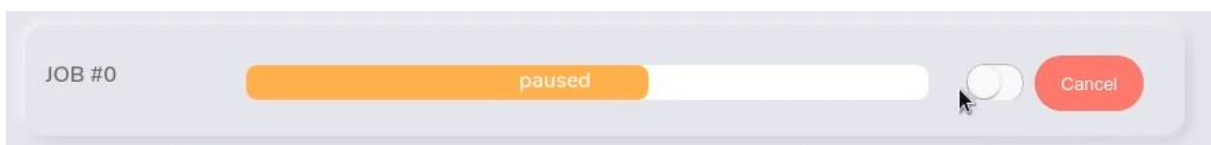


Următorul câmp este destinat introducerii căii unde fișierul selectat anterior va fi copiat.

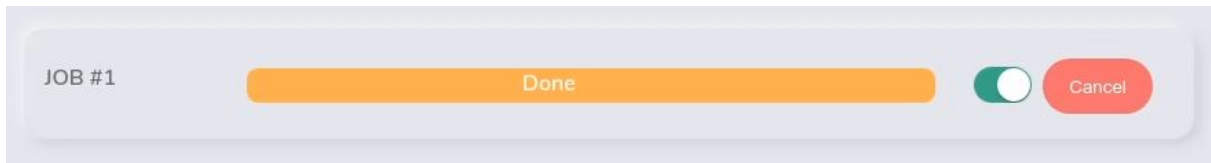


Butonul “Add Copy Job” are o funcționalitate destul de intuitivă, odată introduse căile pentru sursă și destinație, acesta porneste procesul de copiere.

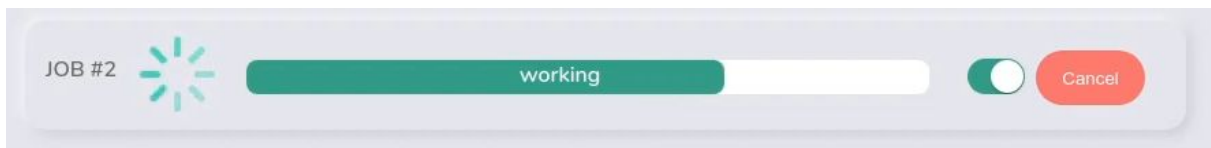
Urmează apoi secțiunea din ecran ce afișează job-urile în desfășurare dar și cele recent încheiate. Aici se disting trei tipuri de afișare, în funcție de tipul de job: în desfășurare, pus pe pauză și terminat.



- Exemplu de ecran pentru un job în stadiul de pauză

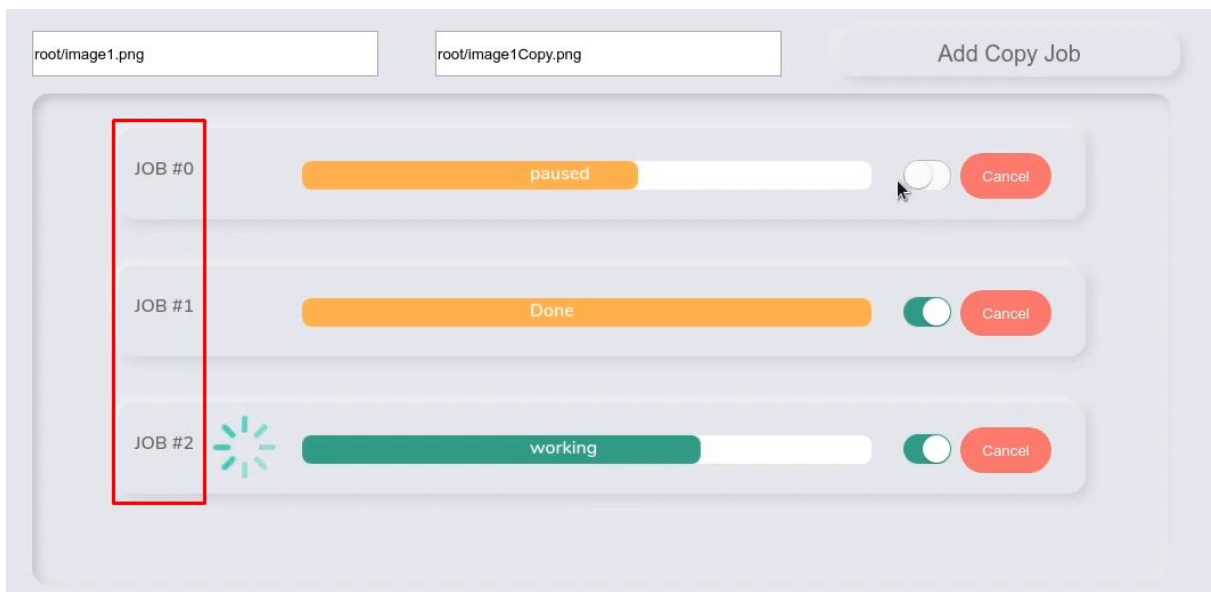


- Exemplu de ecran pentru un job ce a terminat de copiat



- Exemplu de ecran pentru un job ce inca se afla in progress.

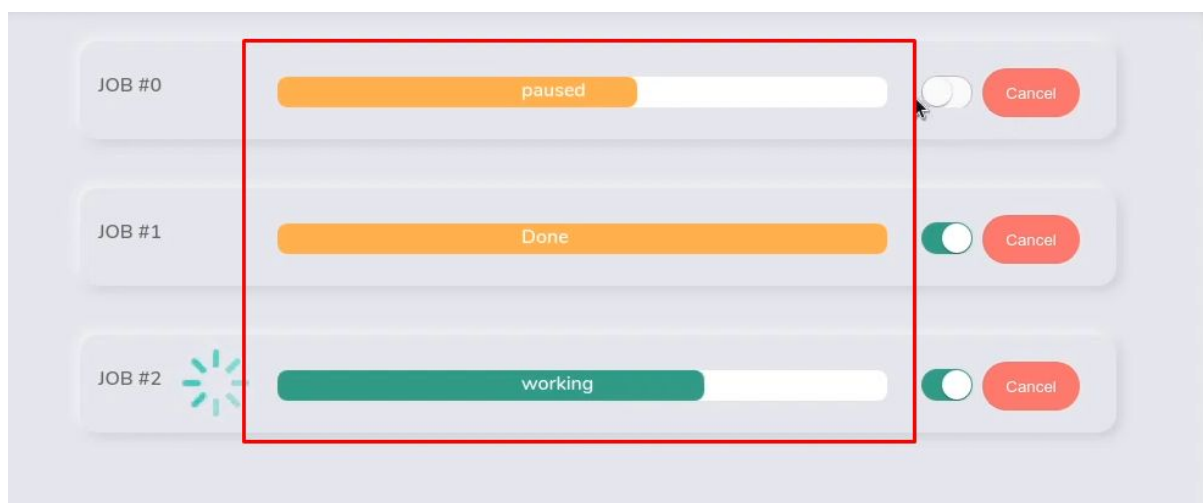
Se pot observa caracteristici comune ale acestora, asa ca le vom discuta impreuna in cele ce urmeaza.



Pe aceasta coloana se afla identificatorul unic al jobului. Acesta este generat in partea de backend a aplicatiei si este introdus pentru a aduce o informatie utila atat pentru utilizator, dar si pentru partea de logging sau eventual debugging.



Pe aceasta coloana, pentru joburile in desfasurare (i.e. cele care nu au terminat sau sunt in stadiul de pauza) gasim un indicator de functionare. Util pentru fiserele mari unde copierea se face mai greu, spune utilizatorului ca programul lucreaza si nu este in stadiu de “freeze”.



In centrul ecranului, in centrul “atentiei” se afla progress bar-ul. In tema simplitatii si intuitivitatii, acesta nu prezinta detalii sau log-uri complicate, ci doar da utilizatorului o idee despre progresul facut de aplicatie.



Butoanele de tip toggle acopera feature-ul ce permite utilizatorului sa pauzeze sau sa reporneasca un job.



In cele din urma, butonul “Cancel” anuleaza un job de copiere, fie ca este in progres sau oprit pentru moment.

3. Impartirea Task-urilor

Crearea proiectului a avut la baza un plan bine stabilit in care fiecare membru al echipei a avut definit task-ul sau pe platforma GitHub. Principalele task-uri au fost :

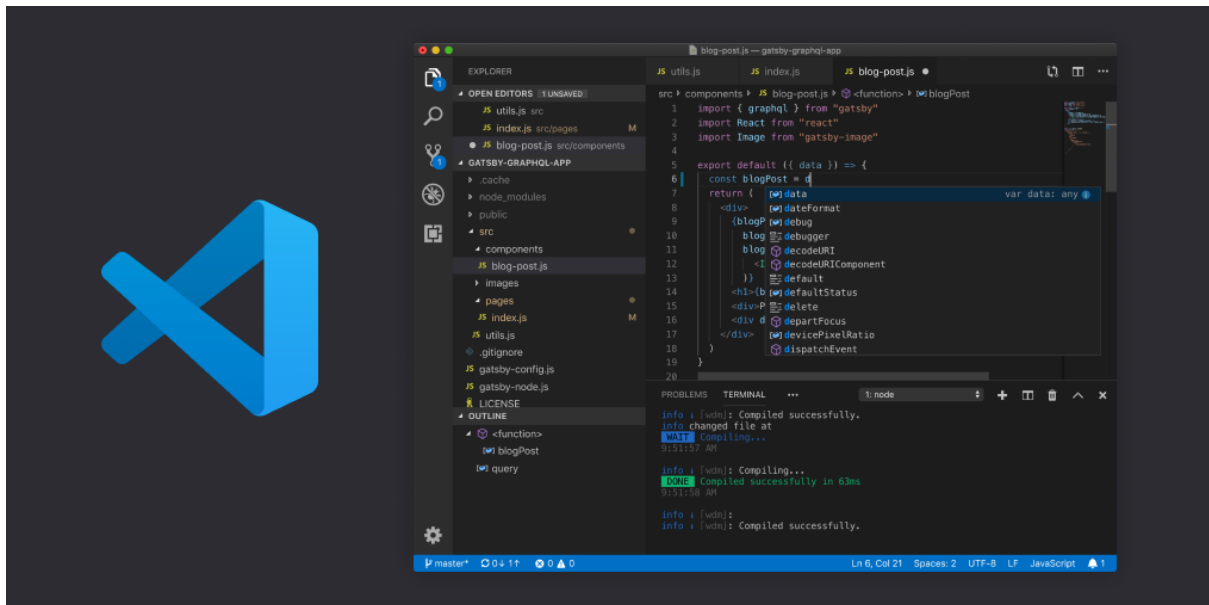
1. Backend in C (Daemon and Client)
2. Frontend (Interface)
3. Avem nevoie de o conexiune intre frontend-ul care este in Javascript si backend-ul (daemon) care este în C. Am scris functii pentru comunicarea cu C backend. Am utilizat nodej-uri cu Express sau Python cu flask si am configurat un api REST folosind cereri POST.
4. Functionality testing
5. Fix Bugs

Plus multe alte task-uri care pot fi vizualizate pe link-ul urmator de github : <https://github.com/mihainsto/CopyService-Application/issues>.

Tot pe github se poate observa fiecare commit in parte cu detaliile respective,dar si alte versiuni ale proiectului fiind pus pe GitHub de la prima versiune plus multe alte teste facute proiectului in timp.

4. Codul sursa

Pentru partea de back-end am ales sa folosim ca editor de cod Microsoft Visual Code fiind un editor de cod uimitor si avansat cu unele caracteristici ale unui IDE cum ar fi debugging-ul si putand aplica alte functii prin intermediul pluginurilor , acest editor fiind ideal pentru proiectul nostru .



In urmatoarea poza de mai jos se poate observa o parte din codul nostru sursa,mai exact una dintre cele mai importante parti de cod care trebuie rulata pentru a functiona copierea sau oricare alt serviciu pe care clientul il alege. Serviciile disponibile sunt urmatoarele : copiere ,suspendare (anuleaza copierea) , status (pentru a verifica in ce stadiu se afla procesul de copiere), stop (pentru a pune pauza procesului) si resume (pentru a reporni copierea respectiva).

```

1  #include "IPC.h"
2  #include "threadService.h"
3
4
5
6  int main() {
7      maxThreads = 2;
8      IPCmessageToDaemon client_message;
9      IPCmessageToClient sentmessage;
10
11     while (1) {
12         client_message = IPC_DaemonRecvMessage();
13         printf("%s\n", client_message.task);
14         if (strcmp(client_message.task, "copy") == 0)
15         {
16             int id = copyThread(&client_message);
17             sentmessage.jobId = id;
18             IPC_DaemonSendMessage(sentmessage);
19         }
20         else if (strcmp(client_message.task, "status") == 0)
21         {
22             printf("STATUS");
23             float status = statusThread(client_message.jobID);
24             sentmessage.status = status;
25             IPC_DaemonSendMessage(sentmessage);
26         }
27         else if (strcmp(client_message.task, "stop") == 0)
28         {
29             stopThread(client_message.jobID);
30         }
31         else if (strcmp(client_message.task, "suspend") == 0)
32         {
33             pauseThread(client_message.jobID);
34         }
35         else if (strcmp(client_message.task, "resume") == 0)
36         {
37             resumeThread(client_message.jobID);
38         }
39         /*printf("%s\n", client_message.task);
40         printf("%s\n", client_message.source);
41         printf("%s\n", client_message.destination);
42         printf("%d\n", client_message.jobID);*/
43
44     }
45     return 0;
46 }

```

Pentru partea de Front-End am ales sa folosim libraria React datorita faptului ca este o librarie JavaScript eficienta si flexibila pentru o interfata grafica specifica pentru aplicatii cu o singura pagina fiind fix de ce am avut nevoie pentru aplicatia noastra.

```

31 |   })
32 |   const [switchStatus, setSwitchStatus] = useState("true");
33 |   const [pathFrom, setPathFrom] = useState("");
34 |   const [pathTo, setPathTo] = useState("");
35 |   const cancelClicked = (value, id, index) => {
36 |     cancelCopyJob(id);
37 |   };
38 |   const swichClicked = (value, id, index) => {
39 |     setSwitchStatus(false);
40 |     console.log(index);
41 |     const oldList = jobs.list;
42 |     const curent_status = oldList[index].switchStatus;
43 |     if (curent_status === false) {
44 |       oldList[index].switchStatus = true;
45 |       oldList[index].status = "working";
46 |       resumeCopyJob(oldList[index].id);
47 |     } else {
48 |       oldList[index].switchStatus = false;
49 |       oldList[index].status = "paused";
50 |       pauseCopyJob(oldList[index].id);
51 |     }
52 |
53 |     setJobs({ list: oldList });
54 |   };
55 |   const addJobClicked = () => {
56 |     console.log(pathTo)
57 |     createCopyJob(pathFrom, pathTo, jobs, setJobs);
58 |     //alterCopyJob(1, jobs, setJobs, 80, "canceled")
59 |     //updateCopyJob(1, jobs, setJobs);
60 |   };
61 |   const inputChangedValue = (value, stateSet) => {
62 |     stateSet(value.text);
63 |     console.log(value.text)
64 |   };
65 |   return (
66 |     <div className="layout">
67 |       <div className="title">Copy Service</div>

```

Aplicatia se foloseste de 3 componente:

- Componenta layout:

Aceasta componenta defineste pagina principala si ea randeaza mai multe componente de tip Job.

Pentru conectarea la backend este definită o librărie care se folosește de requesturi post, după ce se fac aceste requesturiaceasta componenta se folosește de state-management din libraria React, lucru care asigura eficient re-randarea componentelor actualizare la fiecare request dat.

- Componenta Job:

Componenta job primește toate detaliile despre jobul pe care îl definește cum ar fi id, progress, index, status și funcții de tip callback. Această componentă se folosește de componenta progress bar.

- Componenta progress bar:

Componenta progress bar este o componentă simplă care definește o linie de progress și îi atribuie o culoare, ea primește progresul și statusul jobului.

Pentru styling se folosește SASS, o extensie pentru CSS care adaugă multe îmbunătățiri la CSS-ul clasic și face dezvoltarea pe componente mai ușoară.

Deși aplicația este scrisă în tehnologie web noi ne folosim de biblioteca electron pentru a transforma pagina web în aplicație desktop.

- REST API

Pentru a ne conecta la daemonul de copiere care este scris în C am făcut un API REST în python folosind flask. Acest api expune endpoint-uri pentru toate feature-urile de care dispune daemon-ul de copiere.

Conexiunea la acest API se face prin requesturi de tip POST, și el la fiecare request primit apelează funcția corespunzătoare din daemon-ul de copiere și trimite confirmare ca răspuns.

Endpointurile expuse de acest api sunt:

```

44 @app.route('/copy', methods=['POST'])
45 def copy():
46 >     if request.method == 'POST': ...
63
64
65 @app.route('/status', methods=['POST'])
66 def status():
67 >     if request.method == 'POST': ...
74
75
76 @app.route('/stop', methods=['POST'])
77 def stop():
78 >     if request.method == 'POST': ...
86
87
88 @app.route('/suspend', methods=['POST'])
89 def suspend():
90 >     if request.method == 'POST': ...
99
00 @app.route('/resume', methods=['POST'])
01 def resume():
02 >     if request.method == 'POST': ...
09
10 @app.route('/allJobs', methods=['POST'])
11 def allJobs():
12 >     if request.method == 'POST': ...
19

```

Detalii despre daemon:

Pentru explicarea codului am folosit Doxygen (un instrument pentru scrierea documentatiilor de referinta software). Am redactat documentatia in cod si, prin urmare, este relativ usor de pastrat la zi. Doxygen poate traversa documentatia si codul de referinta, astfel incat cititorul unui document se poate referi cu usurinta la codul real.

Acesta se afla in:

back-end/Documentation_doxygen/Doxygen documentation.



Deci Doxygen ne ofera posibilitatea de a gasi codul sursa, dar si de a vedea organizat ce ne-ar putea interesa din cod, precum:

- ☐ lista claselor;
- ☐ lista fisierelor;
- ☐ membrii claselor (toti membrii, variabile, etc.);
- ☐ membrii fisierelor (toti membrii, functii, variabile, typedefs, etc.).

LISTA CLASELOR:

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

 IPCmessageToClient	Struct contains the id of the current job, the status and a list of other j
 IPCmessageToDaemon	Struct contains the source, destination, the task and the type of job

La click pe clasa, vedem detalii despre aceasta: scopul ei, parametrii, descrierea parametrilor.

Exemplu click pe clasa IPCmessageToDaemon:

IPCmessageToDaemon Struct Reference

Struct contains the source, destination, the task and the type of job. [More...](#)

```
#include <IPC.h>
```

Public Attributes

char	source	[1024]
char	destination	[1024]
char	task	[1024]
int	jobID	

Detailed Description

Struct contains the source, destination, the task and the type of job.

Parameters

source Char array with len of 1024, contains the source from where it should copy.

destination Char array with len of 1024, contains the destination to where it should copy.

task Char array with len of 1024, tells the daemon what job it should accomplish: -"copy" = tells the daemon to start a copy tipe job -"status" = gets the status of a current job -"stop" = stops a job -"suspent" = pauses / suspends a job -"resume" = resumes a job

•

Warning

A wrong typed task will result in the daemon doing nothing.

Parameters








jobID Integer, tells us the thread id.

LISTA FISIERELOR:

Copy Service

Main Page	Classes ▾	Files ▾	
File List			

Here is a list of all files with brief descriptions:

 client.c	
 client.h	Makes connection between user and back-end (Daemon
 daemon.c	
 daemon.h	
 IPC.c	
 IPC.h	Inter Process Communication Library
 threadService.c	
 threadService.h	Thread Service Library

FISIERE:

In fisierele cu extensia ".h" se gasesc detalii despre codul scris in fisierul corespunzator ".c".

Aici aflam ce face fisierul .c, variabilele folosite, functiile ce il alcatuiesc, descriere pe scurt a fiecărei functii (ce face, parametri, ce returneaza etc.). La click pe functie, apare o descriere mai detaliata a acesteia.

Fisierele noastre:

IPC.h:

De aici aflam ca IPC permite daemon-ului si clientului sa trimita si sa primeasca mesaje intr-un mod definit.

IPC.h File Reference

Inter Process Communication Library. [More...](#)

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
```

[Go to the source code of this file.](#)

Classes

struct **IPCmessageToDaemon**

Struct contains the source, destination, the task and the type of job. [More...](#)

struct **IPCmessageToClient**

Struct contains the id of the current job, the status and a list of other jobs. [More...](#)

Typedefs

typedef struct **IPCmessageToDaemon** **IPCmessageToDaemon**

typedef struct **IPCmessageToClient** **IPCmessageToClient**

Functions

	void	IPC_ClientSendMessage	(IPCmessageToDaemon msg)
IPCmessageToDaemon		IPC_DaemonReceiveMessage	() Receive the IPCmessageToDaemon struct. More...
	void	IPC_DaemonSendMessage	(IPCmessageToClient msg) Send the IPCmessageToClient struct to the Client. More...
IPCmessageToClient		IPC_ClientReceiveMessage	() Receive the IPCmessageToClient struct. More...

Detailed Description

Inter Process Communication Library.

Inter-Process communication library that allows the deamon and the client to send and receive messages in a defined way.

Typedef Documentation

◆ IPCmessageToClient

typedef struct **IPCmessageToClient** **IPCmessageToClient**

◆ IPCmessageToDaemon

typedef struct **IPCmessageToDaemon** **IPCmessageToDaemon**

Function Documentation

◆ IPC_ClientReceiveMessage()

IPCmessageToClient IPC_ClientReceiveMessage ()

Receive the **IPCmessageToClient** struct.

Parameters

None

Returns

The message from Client \Warning Use only in Clients

◆ IPC_ClientSendMessage()

void IPC_ClientSendMessage (**IPCmessageToDaemon** msg)

◆ IPC_DaemonReceiveMessage()

IPC_DaemonReceiveMessage ()

Receive the **IPCmessageToDaemon** struct.

Parameters

None

Returns

The message from Deamon

Warning

Use only in Daemon

◆ IPC_DaemonSendMessage()

void IPC_DaemonSendMessage (**IPCmessageToClient** msg)

Send the **IPCmessageToClient** struct to the Client.

Parameters

msg The message to be sent to the Client

Returns

void

Warning

Use only in Daemon

threadservice.h:

Observam ca threadservice permite pornirea, intreruperea , reluarea si anulara unui copy job.

threadService.h File Reference

Thread Service Library. [More...](#)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <string.h>
#include <signal.h>
#include "IPC.h"
```

[Go to the source code of this file.](#)

Functions

void	stopThread (int index)	Stops a thread found at position index in threads array. More...
int	copyThread (IPCmessageToDaemon *client_message)	Starts a new copy job on a new thread;. More...
void	pauseThread (int jobID)	Pauses a thread with the idex / jobID. More...
float	statusThread (int jobID)	Gets the progress of a job. More...
void	resumeThread (int jobID)	Resumes a job that has been paused. More...

Variables

pthread_t	threads [1024]	Array of threads, used to store thee jobs on independent threads. More...
int	maxThreads	The maximum number of threads that can run in the same time. Configurable. More...
int	pause_status [1024]	Array of ints that store the status of the thread (running/paused/stopped). More...
float	progress [1024]	Array of floats that store the progress of the thread (from 0=no progress to 1=done) More...
pthread_mutex_t	mtx	General mutex, for all type of jobs. More...
pthread_mutex_t	copyThreadMutex	Mutex for the copy jobs. More...
pthread_mutex_t	pauseMutex	Mutex got the pause jobs. More...
pthread_mutex_t	progressMutex	Mutex ffor the progress jobs. More...

Detailed Description

Thread Service Library.

Thread Service is a Library that allows the start, pause, resume and cancel of a copy job.

Function Documentation

◆ copyThread()

```
int copyThread ( IPCmessageToDaemon * client_message )
```

Starts a new copy job on a new thread;.

Parameters

client_message Informations got form the client like from where to where to copy.

Returns

Integer, is the index of the new thread. Needs to be stored in order to know what thread it is.

◆ pauseThread()

```
void pauseThread ( int jobId )
```

Pauses a thread with the idex / jobId.

Parameters

jobID Integer, the job that needs to be paused

Returns

void

◆ resumeThread()

```
void resumeThread ( int jobId )
```

Resumes a job that has been paused.

Parameters

jobID Integer, the job that needs to be resumed

Returns

void

◆ statusThread()

```
float statusThread ( int jobId )
```

Gets the progress of a job.

Parameters

jobID Integer, the job that i want to know it's progress

Returns

Float, the progress of the job

◆ stopThread()

```
void stopThread ( int index )
```

Stops a thread found at position index in threads array.

Parameters

index Index of the thread that needs to be stopped

Returns

void

Variable Documentation

◆ copyThreadMutex

```
pthread_mutex_t copyThreadMutex
```

Mutex for the copy jobs.

◆ maxThreads

```
int maxThreads
```

The maximum number of threads that can run in the same time. Configurable.

◆ progressMutex

pthread_mutex_t progressMutex

Mutex for the jobs progress.

◆ threads

pthread_t threads[1024]

Array of threads, used to store the jobs on independent threads.

◆ mtx

pthread_mutex_t mtx

General mutex, for all type of jobs.

◆ pause_status

int pause_status[1024]

Array of ints that store the status of the thread (running/paused/stopped).

◆ pauseMutex

pthread_mutex_t pauseMutex

Mutex for the pause jobs.

◆ progress

float progress[1024]

Array of floats that store the progress of the thread (from 0=no progress to 1=done)

daemon.h:

"Daemon" ruleaza ca proces de fundal si primeste mesaje de la client. Acesta incepe thread-uri separate pentru fiecare copy job.

daemon.h File Reference

[Go to the source code of this file.](#)

Functions

int **main** ()

Daemon runs as a background process. [More...](#)

Function Documentation

◆ main()

int main ()

Daemon runs as a background process.

Parameters

None

Returns

0

Receives messages from client

Warning

Starts SEPARATED threads for each copy job.

client.h:

"Client" face conexiunea intre utilizator si back-end (daemon). Primeste mesaje de la utilizator (input) si trimite la daemon un mesaj care contine cerinta (scrierea unui copy job, etc).

client.h File Reference

Makes connection between user and back-end (Daemon). [More...](#)

[Go to the source code of this file.](#)

Functions

```
int main ()
```

Detailed Description

Makes connection between user and back-end (Daemon).

Receives messages from user (input)

Sends to daemon a message which contains the requirement (writing a copy job, etc).

Function Documentation

◆ main()

```
main ( )
```

Parameters

None

Returns

0

MEMBRII CLASELOR:

Avem o lista a membrii claselor impreuna cu clasele de care apartin.

Copy Service

Main Page	Classes ▾	Files ▾	
-----------	-----------	---------	--

Here is a list of all class members with links to the classes they belong to:

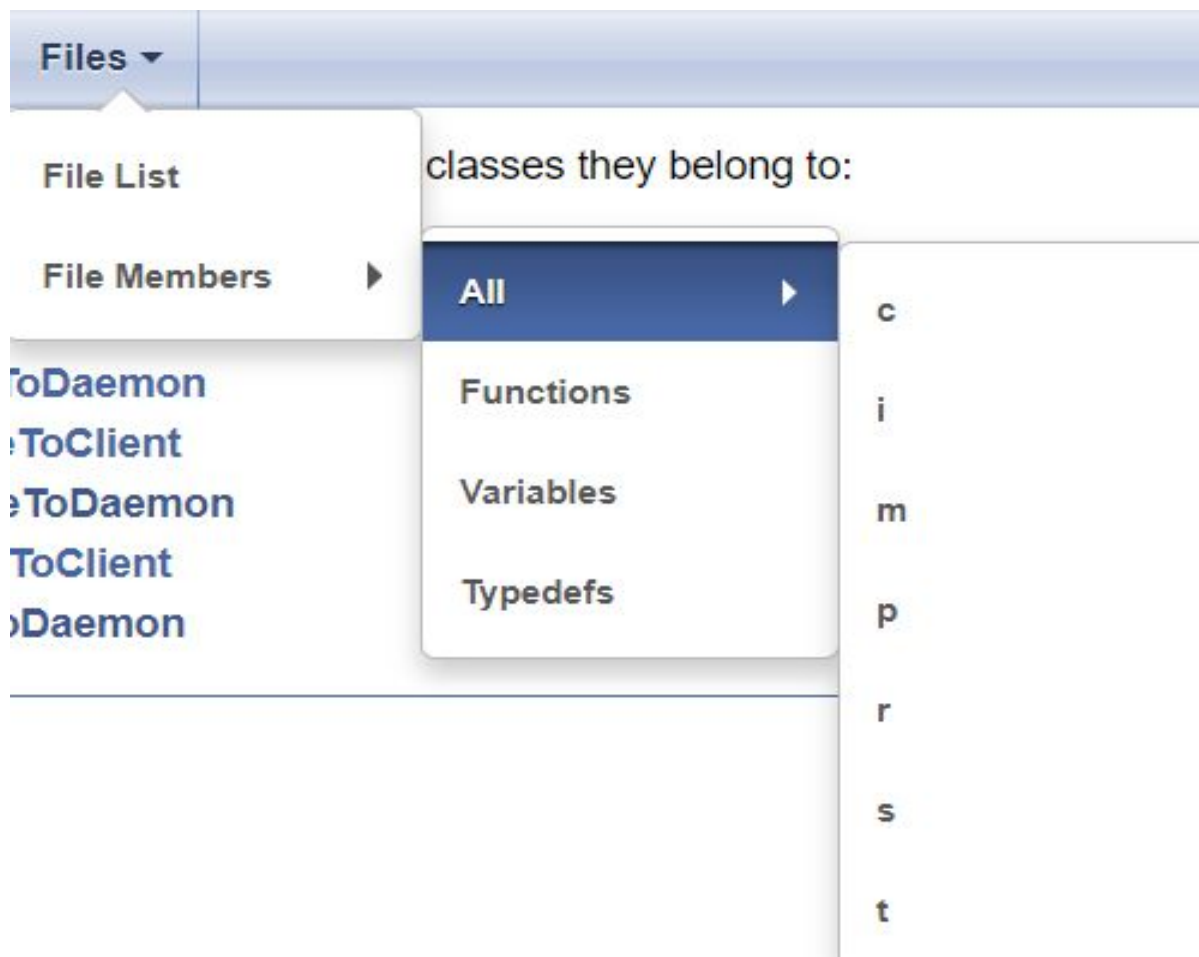
- destination : [IPCmessageToDaemon](#)
- jobId : [IPCmessageToClient](#)
- jobId : [IPCmessageToDaemon](#)
- jobList : [IPCmessageToClient](#)
- source : [IPCmessageToDaemon](#)
- status : [IPCmessageToClient](#)
- task : [IPCmessageToDaemon](#)

MEMBRII FISIERELOR:

Avem mai multe variante in care putem vedea membrii fisierelor:

- ☐ toti membrii in ordine alfabetica (nume : fisierul_din_care_provine)
- ☐ functii (numele_functiei() : fisierul_din_care_provine);
- ☐ variabile (numele_variabilei : fisierul_din_care_provine);
- ☐ typedefs (numele_functiei : fisierul_din_care_provine).

Acestea faciliteaza cautarea noastra, cat si intelegerea aplicatiei.



Here is a list of all file members with links to the files they belong to:

- c -

- config() : [threadService.c](#)
- copy() : [threadService.c](#)
- copyThread() : [threadService.h](#) , [threadService.c](#)
- copyThreadMutex : [threadService.h](#)

- i -

- IPC_ClientReceiveMessage() : [IPC.h](#)
- IPC_ClientReciveMessage() : [IPC.c](#)
- IPC_ClientSendMessage() : [IPC.h](#) , [IPC.c](#)
- IPC_DaemonReceiveMessage() : [IPC.h](#)
- IPC_DaemonReciveMessage() : [IPC.c](#)
- IPC_DaemonSendMessage() : [IPC.c](#) , [IPC.h](#)
- IPCmessageToClient : [IPC.h](#)
- IPCmessageToDaemon : [IPC.h](#)

- m -

- main() : [client.c](#) , [client.h](#) , [daemon.h](#) , [daemon.c](#)
- maxThreads : [threadService.h](#)
- mtx : [threadService.h](#)

- p -

- pause_status : [threadService.h](#)
- pauseMutex : [threadService.h](#)
- pauseThread() : [threadService.h](#) , [threadService.c](#)
- progress : [threadService.h](#)
- progressMutex : [threadService.h](#)

- r -

- resumeThread() : [threadService.c](#) , [threadService.h](#)

5. Design, arhitectura

Pentru a intelege mai bine cum functioneaza aceasta aplicatie, am creat cateva reprezentari grafice:

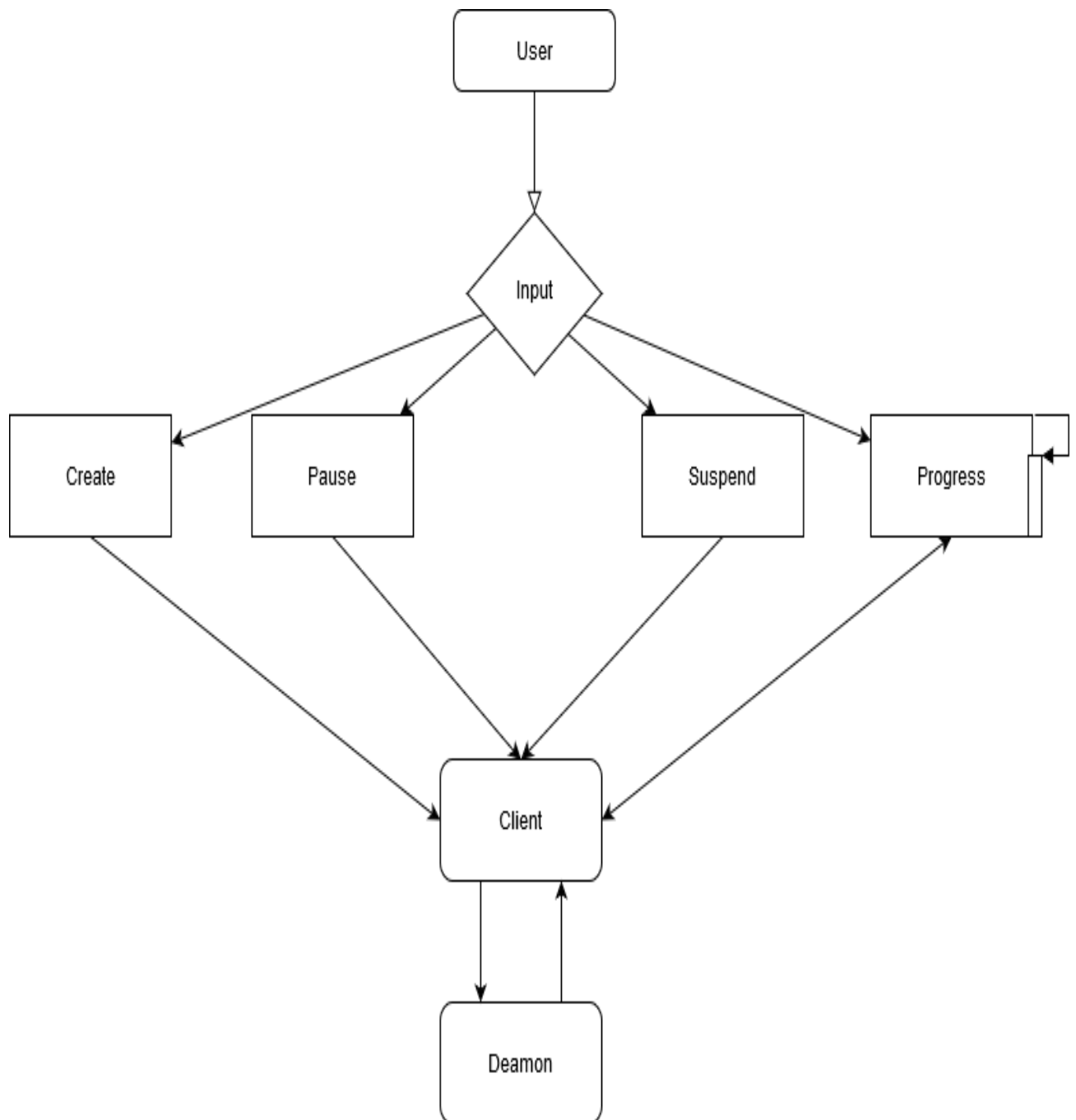
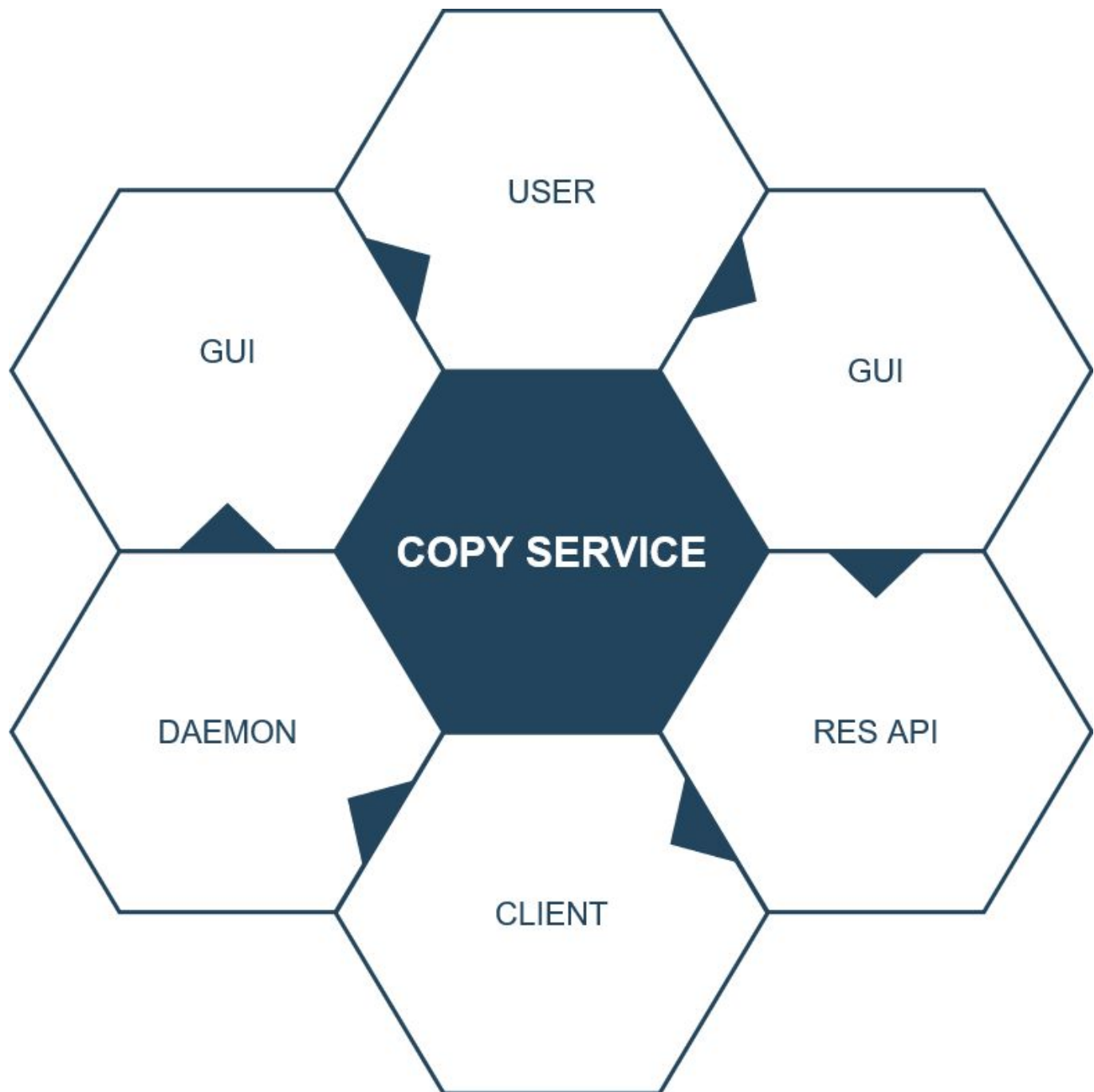


Diagrama de stari , triunghiul mic colorat reprezentand starile prin care trece procesul.



6. Probleme intalnite

La capitolul probleme, fie ca discutam despre issue-uri ,bug-uri sau alte feature-uri despre care am aflat ca sunt necesare, abordarea a fost dinamica si rezolvarea lor s-a putut face “on the go”. Un punct forte al proiectului a fost modularitatea acestuia, facilitand astfel posibilele interventii.

Cateva probleme usor de reparat au fost: repararea functiei ce returneaza statusul unui job, se pare ca aici a fost o gresala de neatenție si de sintaxa C (statusul este de tipul float dar acesta se returna ca un int, conversia de float->int cauzand problema).









O problema mai serioasa, in partea de backend C a fost nesiguranta thread-urilor. Aceasta problema a fost detectata timp indelungat de la dezvoltarea librariilor noastre (IPC si threadService) iar reparatia acestora a fost mai dificila.

Ajunsi la etapa de creare a interfetei grafice, am constatat ca serviciile din C sunt incomplete, si constatham nevoia impelentarii unei functii ce ne furnizeaza o lista cu toate job-urile active. Deoarece proiectul era in o faza terminala, solutia optima pentru rezolvarea acestei probleme necesita foarte mult timp. In acest sens am adaugat acest nou feature librariilor noastre, in un mod mai putin eficient, astfel incat sa le pastram functionalitatea si fara un cost de resurse prea mare.

Deoarece proiectul a fost dezvoltat in paralel de catre toti membrii echipei, o mare parte din probleme se pot gasi pe GitHub sub sectiunea Issue, impreuna cu rezolvarile lor.

7.Source Control

Ca orice alt proiect de dimensiuni mai mari, nu puteam avea o singura versiune a codului. Prima problema care a aparut a fost aceea de a ne sincroniza toti membrii proiectului in acelasi timp pentru a lucra ,dar in salvarea noastra a venit aplicatia GitHub.Aplicatia GitHub ne-a ajutat sa salvam fiecare versiune a proiectului dupa cum se pot vedea urmatoarele branch-uri, dar si sa lucram individual dupa propriul program, iar mai apoi ceilalti membrii sa aiba proiectul updatat.

Default branch					
	master	Updated 1 hour ago by mihainsto		Default	
Your branches					
	rest-api	Updated 2 days ago by mihainsto	<div><div></div><div>25</div><div>0</div></div>	#15	Merged 
Active branches					
	gh-pages	Updated 44 minutes ago by mihainsto	<div><div></div><div>38</div><div>12</div></div>	 New pull request	
	auto-build	Updated 1 hour ago by Teshyx	<div><div></div><div>1</div><div>0</div></div>	#25	Merged 
	front-back-sync	Updated 1 hour ago by mihainsto	<div><div></div><div>5</div><div>0</div></div>	#24	Merged 
	dest-fix	Updated 3 hours ago by Teshyx	<div><div></div><div>7</div><div>0</div></div>	#23	Merged 
	front-requests	Updated 3 hours ago by mihainsto	<div><div></div><div>10</div><div>0</div></div>	#21	Merged 
View more active branches >					

Nu putem omite numarul commit-urilor care este mare, rezultand faptul ca proiectul a fost intr-o munca continua din partea tuturor membrilor.

working fronend and backend

 mihainsto committed 1 hour ago



54e470c



Merge pull request #23 from mihainsto/dest-fix ...

 mihainsto committed 3 hours ago

Verified



5ea7bb8



Fixed Dest File issue


 Teshyx committed 3 hours ago



6f9cc3d



Merge pull request #21 from mihainsto/front-requests ...

 TesileanuAlexandru committed 3 hours ago

Verified



c70952e



small fix backend


 mihainsto committed 3 hours ago



2e7f9c7



Merge pull request #20 from mihainsto/front-requests ...

 TesileanuAlexandru committed 4 hours ago

Verified



6bdaac6



pause cancel create requests

 mihainsto committed 4 hours ago



db4733e



Merge pull request #19 from mihainsto/getter-jobs ...

 mihainsto committed 4 hours ago

Verified



f75f806



getJobs Added ...

 Teshyx committed 4 hours ago



c36096e



Merge pull request #17 from mihainsto/front-end ...

 pgeorge26 committed 6 hours ago

Verified



6f098ed



Merge pull request #16 from mihainsto/daemon-fix ...

 pgeorge26 committed 6 hours ago

Verified



6037584



8.Testare

Pentru a ne asigura ca aplicatia functioneaza optim, testarea are un rol cheie in dezvoltare. Astfel, primul pas spre asigurarea unei functionalitati dorite am recurs la testarea manuala, iterativa, a fiecarei functie creata de noi.

In acest fel ne-am asigurat trecerea de la idee la implementare a functionat asa cum ne-am dorit. Acest lucru a fost facilitat si de modularitatea cu care aplicatia a fost dezvoltata, putand astfel testa diverse module in mod individual.

O alta masura luata a fost creerea de teste automate. Acestea verifica diverse module pentru a corecta diverse erori de logica, de tiparire, de functionare. Acestea s-au dovedit a fi utile in cazul modificarilor aduse proiectului, testele indicand diverse erori in noile imbunatatiri.

```
mstoian@pop-os:~/Documents/CopyService-Application/Rest API/Test$ node unitTesting.js
Starting unit testing...
{ status: 'OK', id: 0 }
{ status: 'OK', id: 0 }
stdout:
***TEST#1 Text test Passed - Can copy text files***
stdout:
***TEST#2 Image test Passed - Can copy image files***
{ status: 'OK', data: '0.000000' }
***TEST#3 Passed - Can request status***
{ status: 'OK', id: 0 }
stdout:
***TEST#4 Passed - Can pause and resume copy jobs***
{ status: 'OK', data: [ '0', '0', '0', '0' ] }
***TEST#5 Passed - Can request all jobs***
```

Am folosit un numar de 5 teste automate care se asigura ca toate endpoint-urile backend-ului functioneaza corect.

Aceste teste au fost scrise in NodeJs, ele testeaza in acelasi timp si functionalitatea backend-ului scris in python dar si functionalitatea daemon-ului de copiere.

Testul 1 - Se asigura ca se poate porni un job de copiere si ca se poate copia un fisier text cu succes.

Testul 2 - Se asigura ca se poate copia o imagine cu succes.

Testul 3 - Acest cere statusul pentru un job de copiere si se asigura ca acesta se primeste cu succes.

Testul 4- Acest test porneste un job de copiere pentru o imagine pe care il pune pe pauza si dupa il reporneste, verificand la final ca imaginea copiata este identica cu cea initiala, astfel testul concluzioneaza ca endpoint-urile de pauza/repornire functioneaza.

Testul 5 - Solicita toata joburile de copiere din copy service pentru a vedea daca acest endpoint functioneaza

9. Viitoare dezvoltari

Pentru moment consideram aplicatia noastra un MVP (minimum viable product) deoarece acesta indeplineste toate feature-urile cheie, dar si cateva feature nice to have precum progress bar-ul, indicatorul de functionare.

Pe viitor aplicatia poate fi dezvoltata in multiple directii precum:

- portarea acesteia pe platforma Android (ambele avand la baza un mediu compatibil cu limbajul C)
- introducerea unui buton ce permite afisarea unui log mai detaliat asociat job-ului
- introducerea unei pagini de configurari, precum numarul total de job-uri active, util pentru dispozitive cu specificatii mai slabe, limitand numarul de resurse consumate
- diverse imbunatatiri la aspectul de viteza al aplicatiei

10.Incheiere

In incheiere , putem spune ca prin acest proiect am aprofundat cunostintele de programare , dar cel mai important lucru a fost ca am lucrat ca o echipa pregatindu-ne intr-o masura mica de munca si de proiectele ce vor urma in viitorul apropiat pentru noi. Pentru a vedea mai in amanunt proiectul nostru si pentru a il testa , il puteti descarca de aici:[Link](#) si asteptam un feedback pentru a imbunatati aplicatia ,iar pe viitor speranta noastra e sa ajunga de la un mic proiect la ceva mare.