Documentaţie

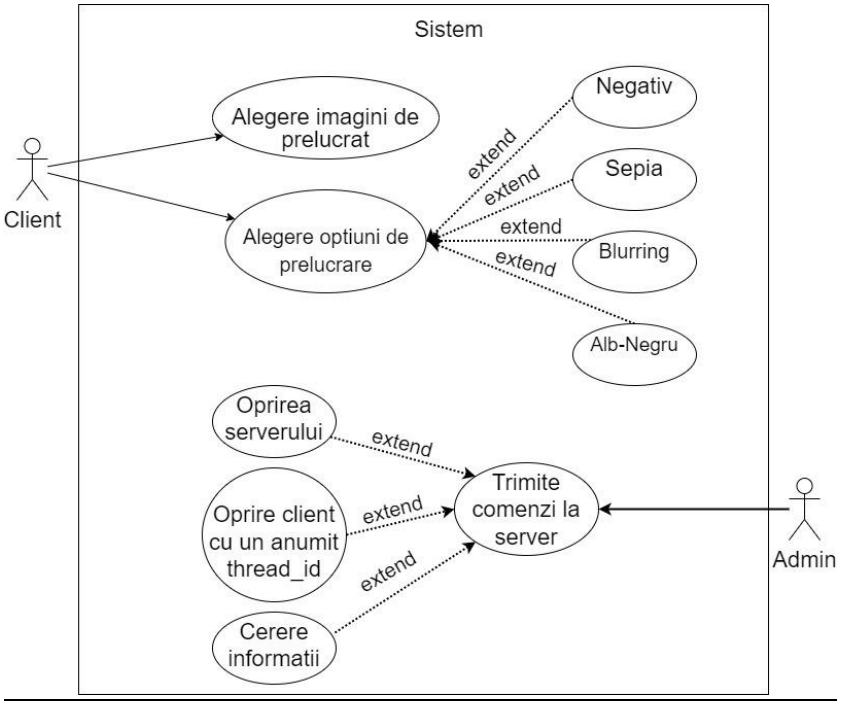
1. **Înregistrarea aplicaţiei: Procesare de imagini**
    - Link Git: https://github.com/MikloBenjamin/PCD
2. **Scheletul aplicaţiei - structura FCE**
    - Client – Server -> Multiclient -> Client_Py, REST -> Admin
    - **Serverul, clientul 1** (administratorul) şi clientul 2 sunt realizaţi în limbajul de programare C.
        i. După compilare: “make”, serverul se porneşte cu comanda: “./main”.
        ii. Adminul poate să trimită comenzi la server cu mesaje tip:
            1. [0] -> Deconectează adminul de la server
            2. [1] -> Închide serverul
            3. [2, client_id] -> Opreşte conexiunea cu clientul care are id-ul client_id
            4. [3] -> Cerere pentru informaţii despre server, numărul de clienti etc.
        iii. Serverul creează un socket, după care 3 threaduri:
            1. Thread care aşteaptă şi procesează conexiunea cu admin
            2. Thread care aşteaptă şi procesează conexiunea cu clienţi
            3. Thread care se ocupă cu comunicarea între server şi client
    - **Clientul 2** este scris folosind limbajul de programare C.
        i. Clientul crează un fir de execuţie pentru citirea pachetelor care vin de la server, unul pentru procesarea lor şi unul care se ocupă cu trimiterea imaginilor la server.
        ii. În funcţia main() se face conexiunea la server, se prelucrează argumentele date în linia de comandă referitoare la calea spre folderul cu imaginile şi modul de prelucrare şi se lansează firele de execuţie descrise mai jos. Acesta va aştepta până când s-a încheiat primirea ultimei poze, după care va opri firul de execuţie care citeşte pachetele de la server, astfel încheindu-se aplicaţia.
        iii. Firele de execuţie pentru citirea pachetelor şi procesarea lor au grijă să rezolve problema producător-consumator folosind mutex-uri şi semafoare pe coada de pachete.
        iv. Firul pentru trimiterea imaginilor va trimite fiecare imagine pe rând, iar după fiecare pachet trimis va aştepta o confirmare de la server după cum este descris protocolul în secţiunea dedicată; după fiecare pachet se va opri la un semafor până când este primit mesajul corespunzător de la server.
        v. Firul pentru procesarea pachetelor va interpreta pachetele de la server:
            1. Dacă este pachet de confirmare, se va incrementa semaforul la care asteaptă firul care trimite imaginile.

2. Dacă este pachet care anunţă câte pachete urmează să fie trimise, va crea un fişier care reprezintă poza procesată trimisă de către server.
3. Dacă este pachet cu informaţie aparţinând pozei, atunci va scrie în fişierul menţionat mai sus datele. Dacă s-a ajuns la ultimul pachet, fişierul este închis.

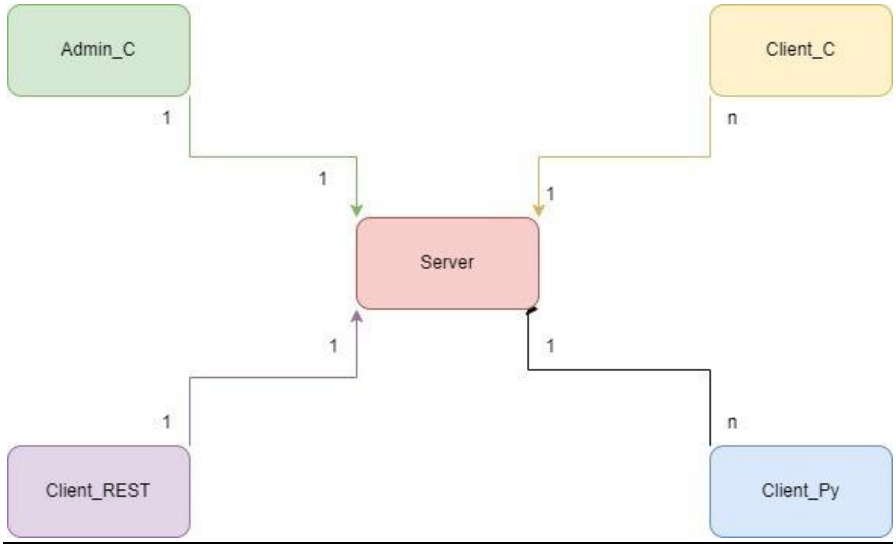
- **Clientul 3** are aceeaşi funcţionalitate ca şi clientul 2, însă este scris în limbajul Python. Procesul clientului 3 este împărţită în 2 threaduri + mutex:
  i. Primul thread:
     1. Clientul 3 după ce s-a conectat cu serverul, va citi imaginile din folderul precizat în comanda de pornire.
     2. După ce încarcă imaginile, se trimite un pachet către server cu numărul imaginilor şi opţiunea dorită.
     3. Următorul pas al clientului 3 va fi să trimită un pachet cu numărul pachetelor/părţilor din imagine după care va primi o confirmare de la server.
     4. După confirmarea serverului, clientul 3 va trimite pe rând părţile imaginii cu tot cu mărimea acesteia. După fiecare parte a imaginii trimise, clientul primeşte confirmarea serverului.
  ii. Al doilea thread:
     1. Clientul 3 după ce a trimis poza curentă, va aştepta după server să trimită pachetul cu numărul părţilor imaginii şi procesul anterior se repeta, dar acum serverul trimite părţile şi clientul 3 le va primi.
- **Clientul 4** are aceeaşi funcţionalitate ca şi clientul 2, însă este scris în limbajul Java (Spring Boot) şi are o interfaţă web unde se vor încărca şi afişa pozele.
  i. Prima acţiune realizată de Clientul 4 este cea de colectare de imagini prin intermediul unei interfeţe web. Acest lucru se realizează în cadrul unei operaţiuni de tip GET, printr-un buton upload.
  ii. Imaginile încărcate în urma upload-ului vor fi stocate într-un folder temporar.
  iii. În continuare, va avea loc o operaţiune de tip POST, prin care se încearcă crearea canalului de comunicare între Client 4 şi Serverul aplicaţiei.
  iv. Client-ul 4 va trimite fiecare poza sub forma de “byte array” Serverului ,respectând protocolul stabilit.
  v. După ce toate imaginile au fost prelucrate de server, acestea vor reveni înapoi la Client 4 sub forma de “byte array”, ulterior vor fi transformate în imagini şi stocate într-un nou fişier temporar.
  vi. Ultima etapă este reprezentată de o operaţie POST, în care imaginile prelucrate vor fi afişate pe o interfaţă web.
- Se vor trimite pozele pe rând serverului şi opţiunea de prelucrare, după care se va modifica poza făcând operaţii pe canalele de culoare RGB folosind librăria

"libpng", după care vor fi trimise înapoi clientului. Administratorul va putea trimite serverului comenzi referitoare la informaţii despre numărul de clienţi conectaţi, deconectare client şi închidere server. Interacţiunile detaliate sunt descrise în diagrama de secvenţă.

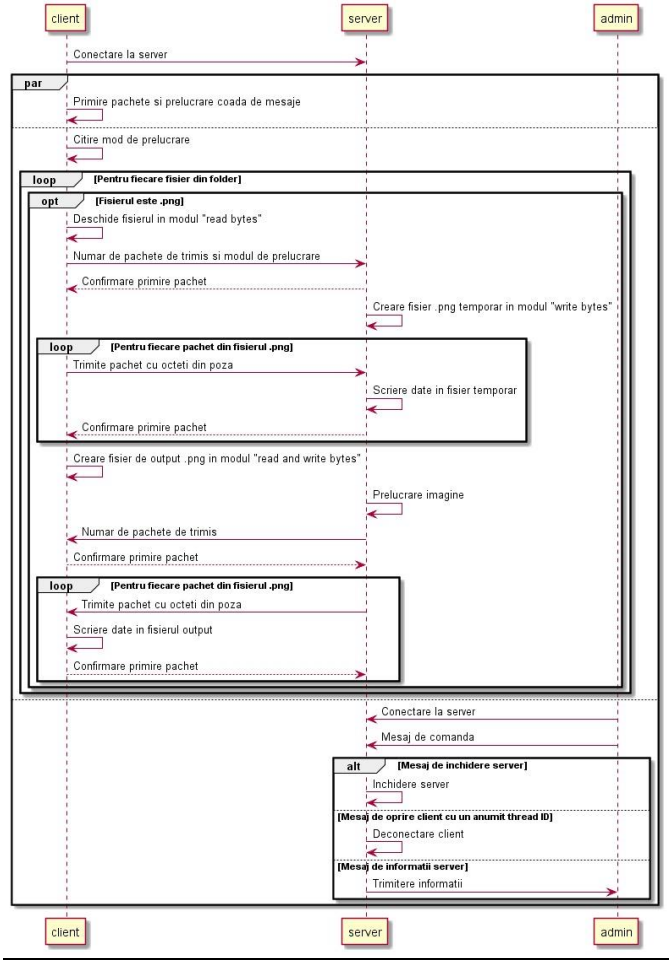
- Diagrama cazurilor de utilizare



- Structura de ansamblu a proiectului



- Diagrama de secvenţă - descrie comunicarea dintre server şi administrator şi dintre server şi un client standard, însă structura se generalizează pentru n clienţi



3. Componenţa echipei şi contribuţia membrilor:

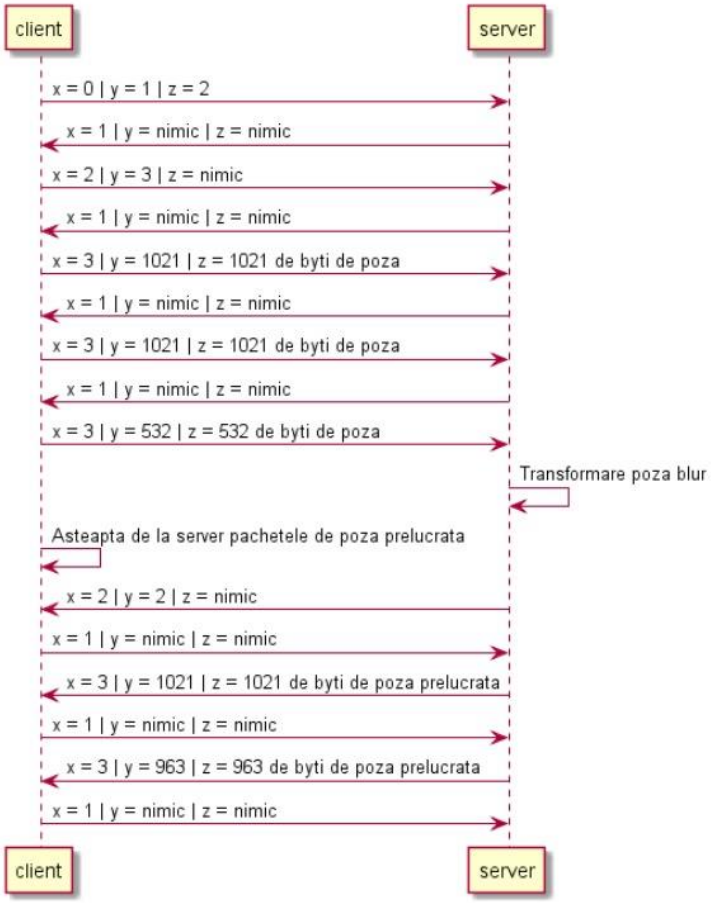
a. Németh Tímea Sarah: client Python
b. Olah Mihai Christian: client C, protocol
c. Miklo Benjámin: server, admin
d. Princz Antonio Ştefan: client Java (Spring Boot şi REST)

4. Protocol

a. protocolul este format din 3 părţi x, y, z, unde:
   i. X -> tipul mesajului (0 … 4)
   ii. Y -> lungimea mesajului(când x = 3) , numărul de pachete (când x = 2), numărul de poze care vor fi trimise (când x = 0)
   iii. Z -> parte din conţinutul pozei, cu fiecare mesaj, se trimit 1021 bytes din poza originală (când x = 3), transmite opţiunea server-ului (când x = 0)

b. fiecare mesaj al protocolului are o lungime constantă de 1024 bytes dintre care x-ul acoperă 1 byte, y-ul acoperă 2 bytes şi z acoperă 1021 bytes.
c. înţelesul fiecărui mesaj transmis este dat de combinaţia x - y:
    i. x = 0 şi y = nr. poze : se transmit numărul de poze care vor fi prelucrate de la client la server si z = nr. olpțiunii de filtru
    ii. x = 1 : reprezintă confirmarea că pachetul a fost primit de către server sau de către client
    iii. x = 2 şi y = nr. pachete poză: se transmite numărul de pachete care vor fi preluate de către server şi vice-versa către client
    iv. x = 3 şi y = lungimea mesajului: se transmite lungimea mesajului server-ului/clientului

Exemplu:



În exemplul de mai sus clientul cere prelucrarea unei singure imagini cu filtrul de blur (x = 0 - informaţii referitoare la editare, y = 1 – o singură imagine se doreşte a fi prelucrată, z = 2 – ID-ul filtrului, adică blur în acest caz). Serverul trimite o confirmare (x = 1), după care clientul trimite serverului numărul de pachete din care e compusă imaginea care urmează a fi trimis serverului (x = 2 – pachet cu informaţii referitoare la poza care urmează să fie trimisă, y = 3 – vor fi trimise 3 pachete corespunzătoare acestei poze).

Cât timp avem pachete de trimis, clientul va trimite pe rând un pachet cu octeţi din poză (x = 3 – pachet de date, y = 1021 – numărul de octeţi care urmează în pachet aparţinând pozei, z – octeţi din poza efectivă), iar serverul va trimite o confirmare (x = 1).
Când se termină de trimis datele, serverul va procesa imaginea cu filtrul cerut, iar clientul va aştepta.

După ce a fost terminată imaginea de prelucrat, serverul va trimite pachet cu numărul de pachete care urmează să fie trimise cu poza editată (x = 2, y = 2 – sunt două pachete de trimis), aşteaptă confirmare de la client, după care se face schimbul de pachete de date similar cum a fost când s-a trimis poza serverului, însă de data aceasta serverul este emiţătorul, iar clientrul receptorul.

5. Descriere aplicaţiei REST

Aplicaţia REST (client 4) a fost dezvoltată folosind framework-ul Spring Boot.

În ceea ce priveşte compoziţia acesteia, este una simplă, fiind formată din două pagini web:

a) Prima pagină web cuprinde un form un form + button prin intermediul cărora se vor alege pozele destinate ulterior prelucrării. Această acţiune este una de tip GET şi este mapată ca fiind pagina index.

b) Cea de a doua pagină reprezintă trigger-ul prin intermediul căruia pozele alese la pasul anterior vor fi distribuite pe rând către server pentru a putea fi prelucrate. Acest trigger este activat prin simpla accesare a link-ului “localhost:8080/start”. Aceasta reprezintă acţiunea de tip POST.

6. Usage

Compilare server, client 2 şi admin: make

Server: `./main`

Admin: `./admin`

Client C: `./client_2 -p path_to_the_folder_containing_the_images -n/-s/-b/-w`

`-n` = negativ

`-s` = sepia

`-b` = blur

`-w` = black and white

Client Python: `python ./client_3.py -fi path_for_a_folder_with_image -fpi path_for_a_folder_with_processed_image -n/-s/-b/-w`

`-n` = negativ

`-s` = sepia

`-b` = blur

`-w` = black and white

Client Java:

Varianta a):

1. Pornire server(client 4) din inteliJ idea
2. Accesare pagină `localhost:8080/index`, de unde utilizatorul îşi alege poza/pozele pe rând şi le dă încarcă, tot pe rând
3. După alegerea pozelor, se accesează pagina `localhost:8080/start` pentru a începe operaţiunea de trimitere şi primire a pozelor

Varianta b):

Pentru prima rulare se deschide terminalul în folderul client 4 : `./mvnw install`, urmat de `./mvnw spring-boot:run`, după care accesează browser-ul cu `localhost:8080/index` - pentru alegerea pozelor, urmat de `localhost:8080/start`.