# **Proiecte Modern C++ 2021-2022**

# **Reguli generale**

1. Cerințele proiectelor sunt distribuite în funcție de profesorul de laborator astfel:
   1. Grupele 10LF301 && 10LF303: Mentor Vlad, email: vladvra@yahoo.com, proiect [Onedrive](#_cnzqovlivtfn)
   2. Grupele 10LF202 && 10LF302: Mentor Andreea, email: matheandreean@gmail.com, proiect [Bibliotecă](#_7qep3k3doi7)
   3. Grupele 10LF201 && 10LF203: Mentor Iulian, email: ipopa.bv@gmail.com**,** proiect [Twitter](#_kdcbpfh4qbph)
2. **Durata proiectului**: **1 noiembrie -> 16 ianuarie** (prima săptămână este considerată 1 noiembrie - 07 noiembrie, perioada 20 decembrie - 2 ianuarie formează o săptămână de commituri, astfel că aveți timp 2 săptămâni pentru realizarea numărului de commituri).
3. **Durata totală a proiectului: 10 săptămâni.**
4. **Commituri:**

* **minim 12 commituri cu SENS/săptămână/echipă de 4 persoane SAU minim 9 commituri cu SENS/săptămână/echipă de mai puțin de 4 persoane SAU minim 15 commituri cu SENS/săptămână/echipă de mai mult de 5 persoane**
* **Minim 30 commituri cu SENS/persoană la deadline la prezentare proiect**

1. **Sunt necesare 2 "prezentări" intermediare (întrebări punctuale, erori, prezentare componentă, componenta preferata etc.)**
   1. **Deadline prima prezentare: 10 decembrie**
   2. **Deadline a doua prezentare: 13 ianuarie**
2. **A se ține cont de fișierul de CLEAN CODE TIPS atașat pe platformă**
3. **Proiect parte basic**: pentru rezolvarea cerințelor din baremul minimal se poate obține maxim nota 7 dacă se utilizează fișiere pentru comunicarea client-server sau maxim nota 8 dacă se implementează comunicare folosind Sockets
4. **Proiect parte advanced**: basic + **1 parte advanced**(care valorează 2 puncte)
5. Fiecare echipă o să pregătească pentru prezentarea finală proiectului un **video**. Aceasta conține explicații pe cod și funcționalitatea aplicației, punand accent pe elementele de barem – vezi demo pe platformă, **Video Demo Proiect. Acest video va fi commis pe bitbucket pana la data de 16 ianuarie, ora 23:59.**
6. **Biblioteci acceptate**: SDL, SFML, GLEW, GLM, Qt Creator (pentru oricare altă bibliotecă trebuie să se obțină acceptul coordonatorului de proiect)
7. La predarea proiectului soluția VS completă va trebui să fie **clonabilă** de pe repo și **compilabilă** pe toate configurațiile existente (se vor șterge configurațiile care nu sunt suportate). A se încerca pe un calculator pe care nu s-a făcut dezvoltare.

**Barem corectare**

Prezentarea elementelor de Modern C++ se va realiza la evaluarea proiectului. Trebuie justificată prezența (sau absența) următoarelor elemente:

* const ref
* move semantics
* structuri de date moderne - [Containers library](https://en.cppreference.com/w/cpp/container)
* algoritmi moderni - [Algorithms library](https://en.cppreference.com/w/cpp/algorithm)
* smart pointers - [Dynamic memory management](https://en.cppreference.com/w/cpp/memory)
* template
* Regex
* DLL
* Logging (în special pentru aplicațiile server)

[**Proiecte Modern C++ 2021-2022**](#_dxqahlpkryqa) **1**

[Bibliotecă](#_7qep3k3doi7) 3

[Onedrive](#_cnzqovlivtfn) 5

[Twitter](#_kdcbpfh4qbph) 7

[Anexe de interes general:](#_9wf75l6gdprk) 10

[Arhitectura client server - concepte de bază](#_ke0jhyn7ae8c) 10

[Comunicarea client-server](#_apld8innyeeg) 10

[Stocarea datelor de către client/server](#_806gff6ng4tk) 10

[Session cookie](#_1w694l8dww4q) 11

[Funcționalitate de search](#_qryu6wp874nf) 11

[Hashing](#_onwz4e3nilsg) 11

## Bibliotecă

Dorim să implementăm o aplicație care să simuleze funcționarea unei biblioteci. Aceasta va fi structurată în două programe: aplicația server și aplicația client. Aplicația trebuie să permită conectarea mai multor utilizatori care să poată citi și împrumuta cărțile din bibliotecă.

Colecția de cărți care va forma biblioteca o găsiți [aici](https://www.kaggle.com/zygmunt/goodbooks-10k).

**Barem minimal**

Funcționalitățile aplicației client:

1. La pornire, unui utilizator i se oferă posibilitatea de a se loga în contul său sau își poate crea un cont. Logarea/înregistrarea presupune introducerea numelui de utilizator.

Atenție: numele de utilizator trebuie să fie unic!

1. Meniul principal afișează următoarele opțiuni (funcționalități):
   1. Search: Utilizatorul introduce un query (format din unul sau mai multe cuvinte cheie) și i se afișează o pagină de rezultate. Criterii de căutare posibile: titlul, autorul, ISBN. Căutarea după titlu/autor se va implementa folosind un Inverted Index. În cazul în care nu se găsesc rezultate, se poate folosi drept backup distanța Levenshtein.
   2. Cărți împrumutate: Se afișează lista de cărți împrumutate de către utilizator și timpul rămas până la termenul de returnare. Pentru fiecare carte se poate afișa opțiunea de prelungire a termenului de returnare.
   3. Împrumută o carte: Un utilizator va căuta o carte, o va selecta și apoi o va adăuga în lista sa de cărți împrumutate. Împrumutul este pe o perioadă de 2 săptămâni.
   4. Returnare cărți: După afișarea listei de cărți împrumutate, utilizatorul va selecta una și apoi o va returna
   5. Citește o carte: După afișarea listei de cărți împrumutate, utilizatorul va selecta una și i se va afișa conținutul
   6. Log out
   7. Ștergere cont

Colecția de cărți aflată în fișiere trebuie procesată corespunzător funcționalităților aplicației server. Pentru fiecare carte trebuie să mai adăugați un număr de exemplare aflate la dispoziția bibliotecii. Biblioteca poate deține între 0 și 3 exemplare ale unei cărți. De asemenea, trebuie să adăugați conținutul fiecărei cărți. Puteți să folosiți o valoare default care să reprezinte textul unei cărți.

Funcționalitățile aplicației server sunt corelate cu funcționalitățile aplicației client:

1. La pornire, aplicația va citi și reține în memoria RAM colecția de cărți și lista de utilizatori înregistrați și alte detalii.

Obs! Pentru a obține rezultate de performanță este necesară reținerea și procesarea în mod eficient a datelor citite!

1. Funcția de search: primește de la aplicația client query-ul, realizează căutarea și transmite către aceasta lista de rezultate
2. Cărțile împrumutate: primește cererea, caută și returnează lista de cărți împrumutate
3. Împrumutul: primește cererea de împrumut a unei cărți, se verifică dacă utilizatorul îndeplinește condițiile și se returnează un răspuns. Termenul de returnare este de 2 săptămâni și poate fi prelungit de un utilizator de maxim 2 ori. Pentru ca un împrumut să fie valid trebuie să se îndeplinească următoarele condiții:
   1. Să nu dețină mai mult de 5 cărți împrumutate
   2. Să nu aibă cărți împrumutate și nereturnate la termen.
   3. În bibliotecă se află cel puțin un exemplar al cărții ce urmează a fi împrumutată.
   4. Să nu împrumute un al doilea exemplar al unei cărți deja împrumutate și nereturnate.
4. Returnarea cărților: primește de la utilizator informații despre cartea pe care acesta dorește să o returneze și actualizează lista de cărți împrumutate a acestuia.

**Componente avansate ale proiectului**

1. GUI - Interfață grafică pentru aplicația client
2. **Networking** - comunicarea dintre aplicațiile de client și server să se realizeze folosind Sockets
3. Baze de date - datele utilizate de către server vor fi manipulate folosind o bază de date
4. Sistem de recomandare de cărți - aplicația client va putea afișa în plus o listă de cărți recomandate. Aplicația server primește cererea și va genera listă de cărți recomandate în baza istoricului de cărți împrumutate, ale ratingurilor și a tagurilor lor.

## Onedrive

Vrem să implementăm o aplicație capabilă să sincronizeze pe mai multe calculatoare un folder și conținutul său. De multe ori descărcăm sau creem fișiere pe calculator și dorim o cale ușoară de a le transmite pe alt device. Folosind această aplicație, acest lucru se va petrece automat și facil, fără să mai folosim metode precum transferul prin stick usb sau pe mail.

**Barem minimal**

Pornim de la ipoteza că utilizatorul are un folder pe care vrea să îl sincronizeze.

Funcționalitățile aplicației client sunt:

1. La pornire, unui utilizator i se oferă posibilitatea de a se loga în contul său sau își poate crea un cont. Logarea/înregistrarea presupune introducerea numelui de utilizator.

Atenție: numele de utilizator trebuie să fie unic!

1. Se verifică pe server dacă există deja sincronizat și uploadat un folder asociat utilizatorului logat și dacă da, utilizatorul este întrebat dacă dorește să îl downloadeze. Dacă utilizatorul răspunde negativ, se poate selecta un nou folder de pe disk pentru sincronizare și datele anterioare de pe server sunt șterse.
2. Sincronizarea unui nou folder presupune:
   1. se face o listă cu numele tuturor fișierelor/subfolderelor și data ultimei modificări (și alte detalii), care se salvează local
   2. aceste metadate inițiale sunt transmise serverului pentru backup
   3. folderul cu tot conținutul său sunt transmise serverului pentru a avea un backup și pentru a putea configura noi aplicații client pe viitor
3. În background (folosind polling sau alte mecanisme) se verifică dacă apar modificări în folder:
   1. data ultimei modificări este diferită față de data ultimei salvări
   2. apar noi fișiere/subfoldere
   3. fișiere/subfoldere sunt redenumite
   4. fișiere/subfoldere sunt șterse
4. Transmite serverului informații legate de modificări și fișierele implicate
5. Primește de la server date legate de modificări apărute în folder și actualizează folderul

**Obs!** Puteți să rețineți/generați detalii relevante legate de fișiere. Pentru implementarea eficientă a cazului de redenumire puteți face așa în aplicația client: citiți în format binar întregul fișier, aplicați algoritmul de hashing [Fowler–Noll–Vo](https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function) pe conținut și salvați hash-ul obținut. În momentul în care un fișier “dispare” și “apare” unul nou, puteți să îi comparați hash-ul cu hash-ul salvat anterior pentru a detecta cazul de redenumire.

**Obs!** În cazul în care în folderul sincronizat există scurtături (fișiere care pointează la alte fișiere), ele vor fi sincronizate ca atare, drept scurtături. **NU** se va sincroniza fișierul la care pointează scurtătura!

Funcționalitățile aplicației server sunt corelate cu funcționalitățile aplicației client:

1. La pornire, aplicația va citi și reține în memoria RAM informații despre folderul sincronizat și lista de utilizatori înregistrați și alte detalii.

Obs! Pentru a obține rezultate de performanță este necesară reținerea și procesarea în mod eficient a datelor citite!

1. Pentru fiecare utilizator, stochează un folder pe care utilizatorul îl sincronizează între mai multe device-uri și detalii legate de acele fișiere  
   Obs - probabil va fi necesar sa retineti detalii despre device-urile unui utilizator
2. Primește mesaje de actualizare a datelor stocate (data ultimei modificări, numele etc.)
3. Primește cereri de interogare a datelor stocate (de obicei clientul are nevoie să facă verificări sau utilizatorul adaugă un device pentru sincronizare etc.)
4. Primește cereri de download de date

**Componente avansate ale proiectului**

1. GUI - Interfață grafică pentru aplicația client; Se vor implementa subpunctul a și cel puțin încă 3 subpuncte la alegere din următoarele:
   1. Funcție de browsing: se vor afișa iconițe pentru fișiere/subfolderele din folderul nostru sincronizat. Se poate da click pe subfoldere pentru a le vedea conținutul.
   2. Funcții de redenumire / ștergere a fișierelor.
   3. Iconițe pentru statusul fișierelor: sincronizat cu serverul = bifă verde; în curs de sincronizare = loading icon; nesincronizat = cruce roșie etc.
   4. Funcție de search în folder - a se folosi Inverted Index
   5. Funcție de a opri sincronizarea anumitor fișiere și de a o reporni mai târziu
   6. Funcție de a șterge copia locală a unui fișier și de a o downloada din nou
   7. Funcție de trash - stocare intermediară înainte de ștergerea reală a fișierului
2. **Networking** - comunicarea dintre aplicațiile de client și server să se realizeze folosind Sockets
3. Baze de date - datele utilizate de către server vor fi manipulate folosind o bază de date
4. Sistem de permisiuni - Pentru un anumit utilizator pot exista aplicații client conectate cu diferite grade de permisiuni: read only (server face undo la modificări / acces numai prin GUI) sau read&write

## Twitter

Ne propunem să recreem populara rețea de socializare Twitter. Un ochi atent va observa că orice rețea de socializare bazată pe postări text reprezintă fundația pentru alte rețele de socializare cu feature-uri mult mai complexe.

**Barem minimal**

Pornim de la intenția de a conecta utilizatori și de a implementa postări de tip text.

Funcționalitățile aplicației client sunt:

1. La pornire, unui utilizator i se oferă posibilitatea de a se loga în contul său sau își poate crea un cont. Logarea/înregistrarea presupune introducerea numelui de utilizator.

Atenție: numele de utilizator trebuie să fie unic!

1. Meniul principal afișează următoarele opțiuni (funcționalități):
   1. Post: utilizatorul poate face o postare text de maxim 140 de caractere;
   2. Personal Profile: utilizatorul își poate vedea istoricul postărilor sale
   3. Funcție de follow: utilizator poate să se “împrietenească” cu alți utilizatori. În acest fel puteți construi graful social al platformei.
   4. Feed - vizualizarea de postări. Se va afișa câte o postare o dată - pentru detalii vedeți mai jos. Sub postare se vor afla următoarele opțiuni:
      1. Like sau Dislike - utilizatorul poate da like/dislike și își poate retrage reacția
      2. Comment - utilizatorul poate scrie un comentariu, iar lista de comentarii va fi updatată
      3. See all comments
      4. Retweet - utilizatorul va putea introduce o prefață postării sale
      5. Go to referenced tweet - dacă postarea este un retweet, se va putea accesa postarea inițială
      6. Next/previous post
      7. Back to main menu
   5. Keyword Search: utilizatorul introduce un query format din mai multe cuvinte cheie și se afișează postări care conțin cel puțin un keyword căutat - vedeți Inverted Index.  
      **Obs!** Încercați să prioritizați rezultatele mai recente.
   6. User Search: utilizatorul introduce un nume de utilizator drept query și i se afișează lista de rezultate. Un rezultat nu trebuie să fie neapărat identic cu query-ul, ci se pot afișa și valori apropiate. Pentru a compara stringuri implementați [distanța Levenshtein](https://en.wikipedia.org/wiki/Levenshtein_distance).

**Obs!** În cazul în care vrem să facem refresh la feed, un client trebuie să se întoarcă în meniul principal și să selecteze din nou opțiunea de Feed. Vedeți detaliile de implementare ale aplicației server.

Afișările postărilor:

* Obișnuite: textul, (dedesubt) număr de likes, de dislikes, de comments, de retweets, (dedesubt) ultimele 5 comentarii și autorul lor
* Retweeted: textul, (dedesubt) autorul și textul postării retweeted, (dedesubt) număr de likes, de dislikes, de comments, de retweets, (dedesubt) ultimele 5 comentarii și autorul lor

**Obs!** Textul postărilor/comentariilor vor fi UNICODE! Dorim să ne exprimăm modern 😁😎

Aplicația server va face managementul de date și va răspunde cerințelor clienților. Funcționalitățile aplicației server sunt corelate cu funcționalitățile aplicației client:

1. La pornire, aplicația va citi și reține în memoria RAM lista de postări și lista de utilizatori

înregistrați și alte detalii.

Obs! Pentru a obține rezultate de performanță este necesară reținerea și procesarea în mod eficient a datelor citite!

1. Post: stochează detaliile primite de la aplicația client
2. Personal profile: transmite postările unui utilizator
3. Follow: updatează graful social al rețelei
4. Feed: Creează un **nou** Priority Queue conform detaliilor de mai jos. Va transmite aplicației client câte o postare o dată. Priority Queue-ul nu va fi stocat pe termen lung.
5. Interacțiunea cu un post: va reține detaliile interacțiunii (like, comment etc.); după caz, se retransmite postarea (cu detaliile aferente) pentru reafișare
6. Funcția de search: primește de la aplicația client query-ul, realizează căutarea și transmite către aceasta lista de rezultate

**Obs!** Algoritmul din spatele paginii de Feed va funcționa în baza unui Priority Queue. În coadă se vor afla postările ce vor fi afișate pe rând utilizatorului, iar prioritățile vor fi calculate în funcție de următoarele criterii: distanța în graful social față de autorul postării, data postării, comentarii nou apărute (după un comentariu propriu, la o postare căreia i-am dat retweet etc.), numărul de reacții ale postării etc. Încercați să calculați prioritatea folosind formule matematice în care dați ponderi pentru fiecare criteriu. Experimentați cu diferite configurații de ponderi ale criteriilor pentru a vedea cum se comportă algoritmul.

OBS! Datele pot fi retinute in fișiere de tipul json/text.

**Componente avansate ale proiectului:**

1. GUI - interfață grafică pentru toate funcționalitățile din baremul minimal ale aplicației client. Implementați cel puțin **una** dintre următoarele funcționalități adiționale:
   1. Suport pt poze drept atașament al unei postări - o poză per postare
   2. Suport pt user tags - un utilizator poate fi menționat într-o postare su forma “#utilizator” care devine un link către profilul acelui utilizator
   3. Notificări - comentarii/like/dislike la postările proprii (cu autorul interacțiunii), comentarii noi la postările la care a comentat utilizatorul etc.
2. **Networking** - comunicarea dintre aplicațiile de client și server să se realizeze folosind Sockets
3. Baze de date - datele utilizate de către server vor fi manipulate folosind o bază de date
4. Trends: Un hashtag care nu este urmat de numele unui utilizator poate reprezenta un termen popular/de actualitate. Acești termeni vor fi stocați împreună cu frecvența lor și cu postărilor conținătoare. Acești termeni pot fi integrați în diferite părți ale aplicației:
   1. În GUI: vor fi afișate drept linkuri; la click pe link se va afișa o pagină cu postări care conțin termenul ordonate descrescător cronologic;
   2. În consolă/GUI: în meniul principal se va adăuga o nouă opțiune de Trends care va afișa cele mai recente și populare hashtaguri.
   3. Funcția de search: se poate folosi funcția de keyword search pentru a găsi postări care conțin hashtaguri.

## Anexe de interes general:

### Arhitectura client server - concepte de bază

Aceasta se bazează pe comunicarea dintre mai multe aplicații client și o aplicație server. În general, aplicația client are rol de afișare a datelor primite de la server și este ușor de rulat. Aplicația server stă și așteaptă cereri de la clienți. Aceasta primește conexiuni de la clienți, procesează cererea (care de obicei implică mai multă putere de procesare decât are clientul) și returnează rezultate sau mesaje de confirmare.

**Obs!** Este important ca serverul să fie capabil să poată servi mai mulți clienți simultan!

Funcționalități comune ale aplicației client:

* La deschidere, un user se poate loga cu username-ul (parola nu este necesară)
* La deschidere, un user își poate face cont introducând un username unic; serverul face verificările necesare și dă mesaj corespunzător
* După logare, un user are posibilitatea să-și șteargă contul (dacă îndeplinește anumite condiții în funcție de aplicație)

### Comunicarea client-server

În general, clientul și serverul rulează pe calculatoare diferite și se conectează prin internet.

**Varianta basic**: Clientul și serverul sunt 2 aplicații independente care vor rula simultan pe același calculator. Se poate folosi un fișier text în care scriem date/cereri/comenzi pentru a comunicare între cele 2 procese. Ambele aplicații pot urmări ultima dată când a fost modificat fișierul și pot reacționa:

* Clientul deschide fișierul și își scrie (formatat) cererea către server, salvează și eliberează fișierul
* Serverul observă că s-a modificat data ultimei modificări, citește cererea, o procesează, și scrie în același/alt fișier rezultatul (formatat al) procesării
* Clientul observă că s-a modificat fișierul, îl citește și afișează rezultatul

**Varianta avansată:** Comunicarea se poate face folosind Sockets. Pentru aceasta trebuie să se cunoască IP-urile calculatoarelor care comunică. Se va ține mai târziu un curs legat de această componentă.

**Obs!** Este foarte recomandat să folosiți un model structurat de reprezentare a datelor pentru comunicare între aplicații. În funcție de aplicație, puteți să îl definiți voi sau puteți să folosiți un model deja existent, precum modelul JSON.

### Stocarea datelor de către client/server

În general, serverul este aplicația care reține cea mai mare cantitate de date necesară rulării aplicației, iar clientul doar strictul necesar.

**Varianta basic**: Aplicațiile client/server își stochează datele în fișiere text. Anumite fișiere text vor fi citite la pornirea aplicației pentru inițializare. Deoarece nu ne așteptăm să lucrați cu o cantitate mare de date, ceea ce citiți din fișier puteți păstra în memoria RAM pe durata rulării programului. De asemenea, va trebui să salvați în fișiere date pe care le primiți de la clienți sau date pe care le generează serverul pentru o rulare mai performantă.

**Obs!** Este foarte recomandat să folosiți un model structurat de reprezentare a datelor pentru salvarea lor. În funcție de aplicație, puteți să îl definiți voi sau puteți să folosiți un model deja existent, precum modelul JSON.

**Varianta advanced**: Folosind ceea ce ați învățat de la cursul de Baze de date, puteți să vă definiți tabele relaționale pentru datele voastre și să le citiți/scrieți folosind SQL.  
**Obs!** În general, SQLite este mai ușor de folosit și există biblioteci de C++ care permit conectarea la SQLite.

### Session cookie

Idee inspirată din web development, aplicația client poate stoca detalii în fișier pe care să le citească la pornire și să simplifice inițializarea/logarea.

### Funcționalitate de search

Pentru funcționalitatea de search, vom folosi un [Inverted Index](https://en.wikipedia.org/wiki/Inverted_index). Este o bună structură de date în momentul în care datele noastre în care căutăm, numite documente, sunt formate din mai multe cuvinte. Are 2 componente: un index și liste de documente (posting lists). Indexul reprezintă mulțimea tuturor cuvintelor pe care le avem în documente. Fiecare poziție în index este un cuvânt care apare cel puțin o dată într-un document. Indexul nu conține duplicate. Fiecare cuvânt din index va trimite la o listă de documente care conțin respectivul cuvânt (fiecare cuvânt are un posting list asociat).

Inverted index este bun în general este folosit când dorim să facem căutări exacte.

În momentul în care dorim să căutăm cuvinte asemănătoare cu alte cuvinte, putem folosi [distanța Levenshtein](https://en.wikipedia.org/wiki/Levenshtein_distance). Ideea algoritmului este să calculeze numărul minim de operații pentru a transforma un cuvânt în celălalt (prin inserții, ștergeri sau substituții). Dacă nu găsim rezultate folosind căutarea exactă, putem căuta rezultate apropiate și le putem sorta conform acestei distanțe.

### Hashing

Un algoritm de hashing este un algoritm care primește date de intrare și îl transformă într-un număr sau într-o secvență de caractere - numit hash - care să reprezinte inputul. Dacă algoritmul este determinist, atunci hash-ul poate fi folosit pentru a identifica sau valida inputul prin comparația cu un alt hash deja salvat. Notabil este că inputul care a generat hashul salvat nu mai este disponibil.

Un algoritm simplu de hashing care acceptă orice secvență de numere drept input este [Fowler–Noll–Vo](https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function). Ideea este simplă și constă în înmulțirea numerelor cu niște numere prime și apoi însumarea lor. Atenție că trebuie folosite **anumite** numere prime, după caz!