Profiling – Platformă de recomandat filme

1. Prezentarea scenariului

Un cinefil a cărui listă de filme văzute depășește cu mult vârsta sa își dă seama că nu știe ce filme să mai vizioneze și îi este foarte greu să se organizeze ca să nu își încurce filmele din lista de filme văzute cu puținele rămase în wishlist. Astfel că, el decide să se aboneze unei platforme de recomandat filme, ajutându-l atât în ceea ce privește organizarea, cât și lipsa de idei, rezolvând ambele probleme în același timp.

În momentul în care cinefilul își face cont observă că i se cer diferite date de la cele mai simple cum ar fi numele, prenumele, username-ul și parola pe care dorește să le folosească în continuare pentru a se loga în aplicație, până la unele mai complexe cum ar fi raspunderea întrebărilor unui quiz care are ca scop „cunoașterea" utilizatorului pentru ca recomandările să fie cât mai precise.

Pe de altă parte, administratorii platformei vor să reînnoiască aplicația folosită deoarece cea curentă nu a mai fost actualizată deloc de la apariție. Datele fiecărui user alături de răspunsurile sale la quiz-ul de la register se găsesc într-o tabelă a unei baze de date menită să rețină informațiile fiecărui user în parte pentru ca mai târciu aceste date să furnizeze informații în găsirea recomandărilor, dar și pentru a ajuta organizatorul să își organizeze filmele în două categorii, *seen movies* și *wishlist movies.*

Fiecare user introdus în această bază de date are un *username* propriu și unic, iar preferințele în materie de genuri de filme se reține astfel: [0] – pentru genurile care nu îl atrag și [1] – pentru cele agreate. Totodată pe aceeași metodă mergem și pentru preferințele în materie de tipuri, anume filmele corespund cifrei [0], serialele cifrei [1], iar dacă utilizatorul le preferă pe ambele cifra [2]. În ceea ce privește parola, pentru a asigura securitatea, există câteva reguli în alegerea ei cum ar fi: minimum o literă de mare, minimum o literă mică, minimum o cifră și obligatoriu cel puțin 8 caractere.

2. Unelte externe

- Visual Studio 2022
- Vcpkg - package manager
- Sqlite_orm - baza de date - instalată prin vcpkg
- Qt 6.4.1
- Google Chrome Tracing (chrome://tracing/) - profiling vizual bazat pe un fișier json generat în timpul rulării programului

Deși există multe unelte specializate pentru partea de profiling la momentul actual, cele mai multe necesită o adaptare la mediu, iar primele utilizări vin cu destul de multe dificultăți, în special pentru utilizatorii mai neexperimentați. Pentru un utilizator experimentat complexitatea ridicată oferită de unele din aceste unelte este un factor important în alegerea lor.

Cu toate acestea, noi am ales aceast mediu pentru că este ușor de folosit și a îndemâna oricui fiind operat de Google Chrome (sau de orice alt browser bazat pe Chromium), acest browser găsindu-se pe aproape orice calculator, iar majoritatea sunt obisnuiți cu utilizarea lui, oferindu-le un spațiu prielnic de a-și studia codul și de a extrage informații despre acesta, chiar și pentru utilizatorii începători. Totodată, acest profiler se remarcă și prin ușurința cu care se folosește și accesibilitatea amintite anterior, este nevoie doar de un browser instalat și acces la fisierul de profiling de tip **.json.**

3. Arhitectură

Pentru a reține și structura informațiile despre fiecare user în parte am implementat o clasă *User* în care i-am reținut toate datele unui user: Prenume, Nume, username (cheia primară din baza de date), parolă, peliculă preferată, actor/actriță preferat(ă), genurile agreate și cele mai puțin agreate după metoda descrisă mai sus, dacă contează anul apariției, și dacă da, atunci perioada care ne interesează, și preferințele dintre filme și seriale, dacă ele există, desigur, după metoda prezentată din nou mai sus. Totodată dacă utilizatorul are deja cont și doar se loghează am creat clasa *LogIn* care conține doar username și parolă, dar preia toate informațiile userului respectiv din clasa *User*.

Pentru a studia partea de profiling vom urmări în special metodele de validare a parolei. Acestea au fost împărțite în metode ce folosesc expresii regulate și metode implementate fără ajutorul acestora în modul clasic. Seturile de metode ce țin de fiecare din tipurile de validare au fost incluse în namespace-uri separate și adăugate într-un namespace comun de validare aflat in headerul *PasswordValidator.h* .

În fișierul principal al proiectului sunt implementate funcții care fac apeluri către metodele discutate anterior și lucrează pe toti utilizatorii găsiți în baza de date, începând cu un singur utilizator, numărul acestora crescând la 5, ca mai apoi să ajunga la 10, acest lucru având ca scop studierea funcționalității treptat.

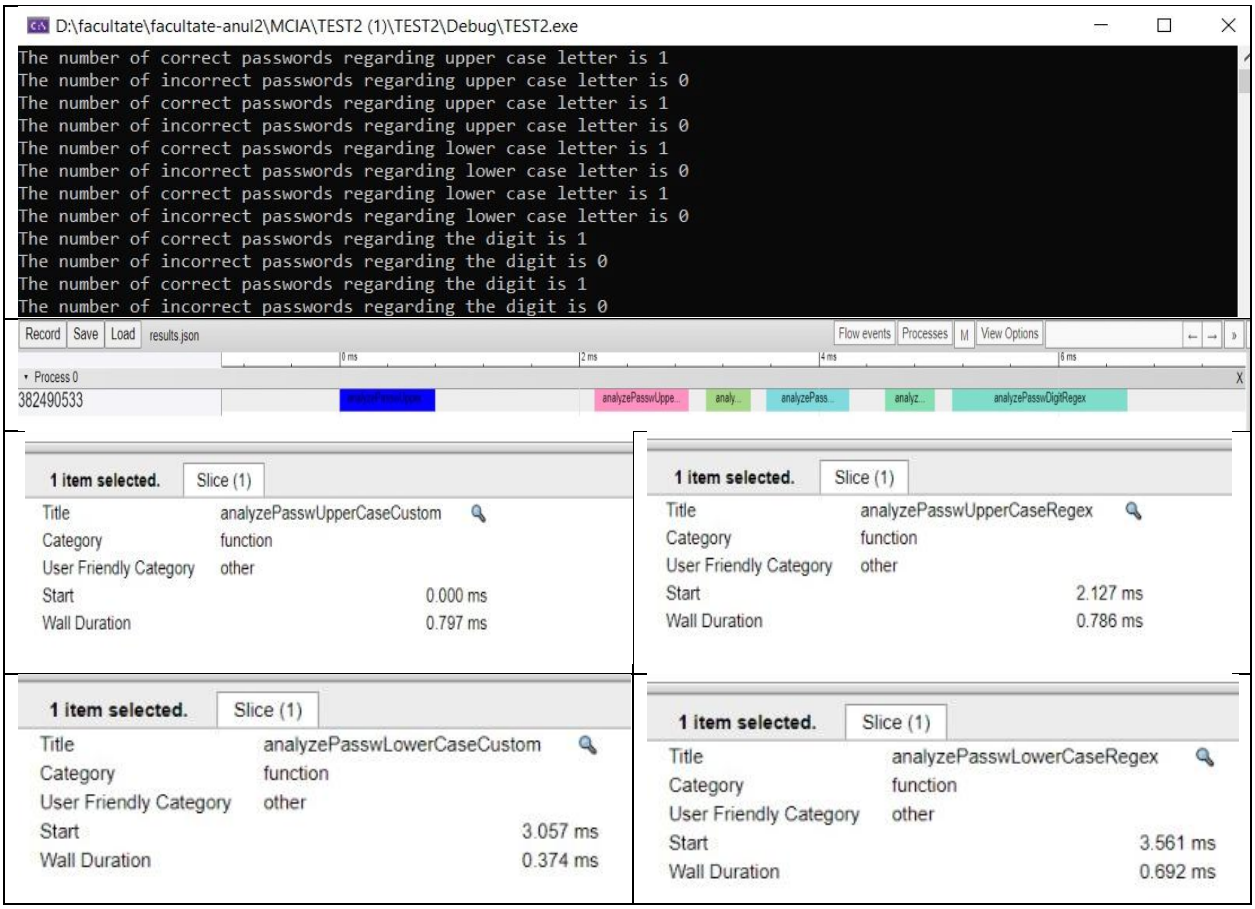
Pentru a genera fișierul de profiling s-a folosit headerul *Instrumentor.h* care conține două clase principale: **Intrumentor** și **IntrumentationTimer** [1]. **IntrumentationTimer** setează la inițializare timpul de start, iar la distrugere calculează diferența de timp dintre momentul actual (la care se întâmplă distrugerea) și timpul de start, setează thread-ul pe care s-au efectuat operațiile și cheamă funcția de scriere în fișier din **Instrumentor**.

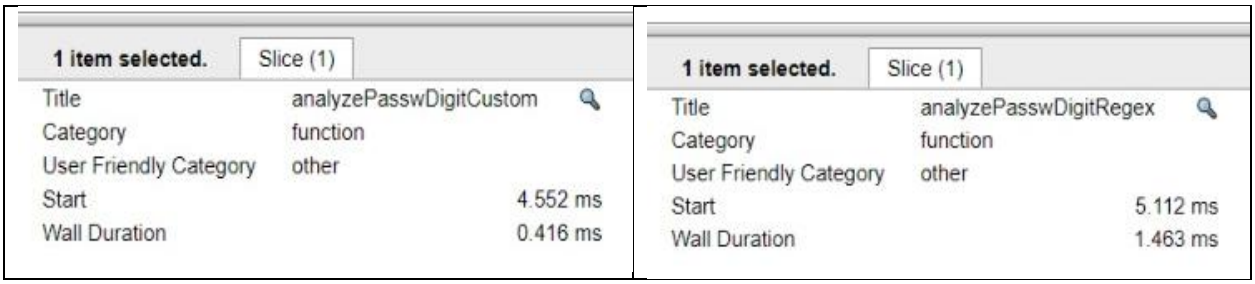
Instrumentor este o clasă *singleton* (crează premizele ca o anumită clasă să fie instanțiată doar o singură dată, permițându-se un acces global la instanța respectivă [2]) care se ocupă de scrierea în fișier și de gestiunea sesiunii pentru care se face profiling. Profilingul se poate salva în fișiere diferite, la nevoie, și fiecare va reține informații despre numele funcției (în cazul nostru), durata de execuție și threadul pe care a fost executată funcția.

Pentru folosirea lor au fost adăugate *două macrocomenzi* ce construiesc obiectul de tip **InstrumentationTimer** și una ce are ca scop o chemarea precedentei cu numele dat drept numele funcției din care a fost apelată macrocomanda [3] .

4. Studierea profilului și a performanței

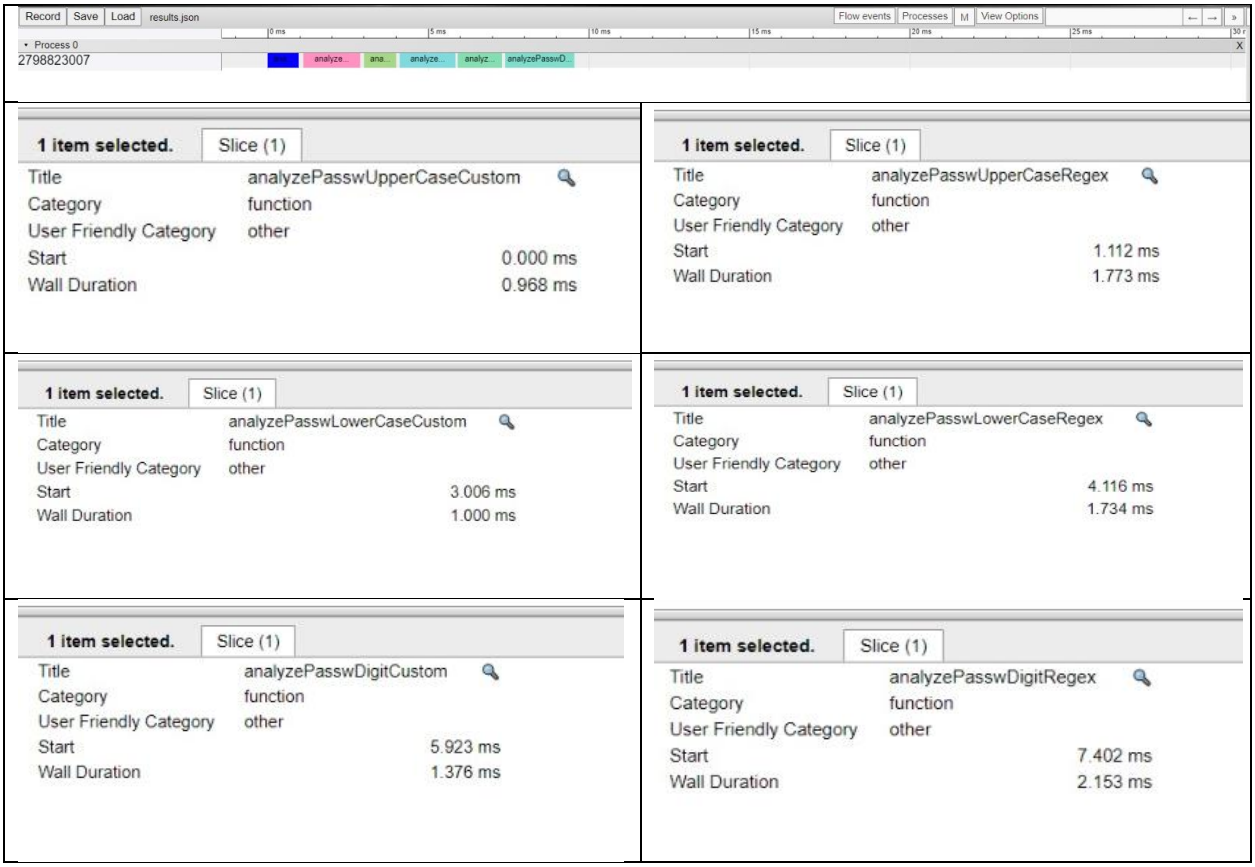
Tabelul următor prezintă comparația de performanță între validări de tip regex și validări personalizate pentru 1 user, folosind un *for* cu iterator pe fiecare parolă și functile predefinite din **string**, `isupper, islower, isdigit`.





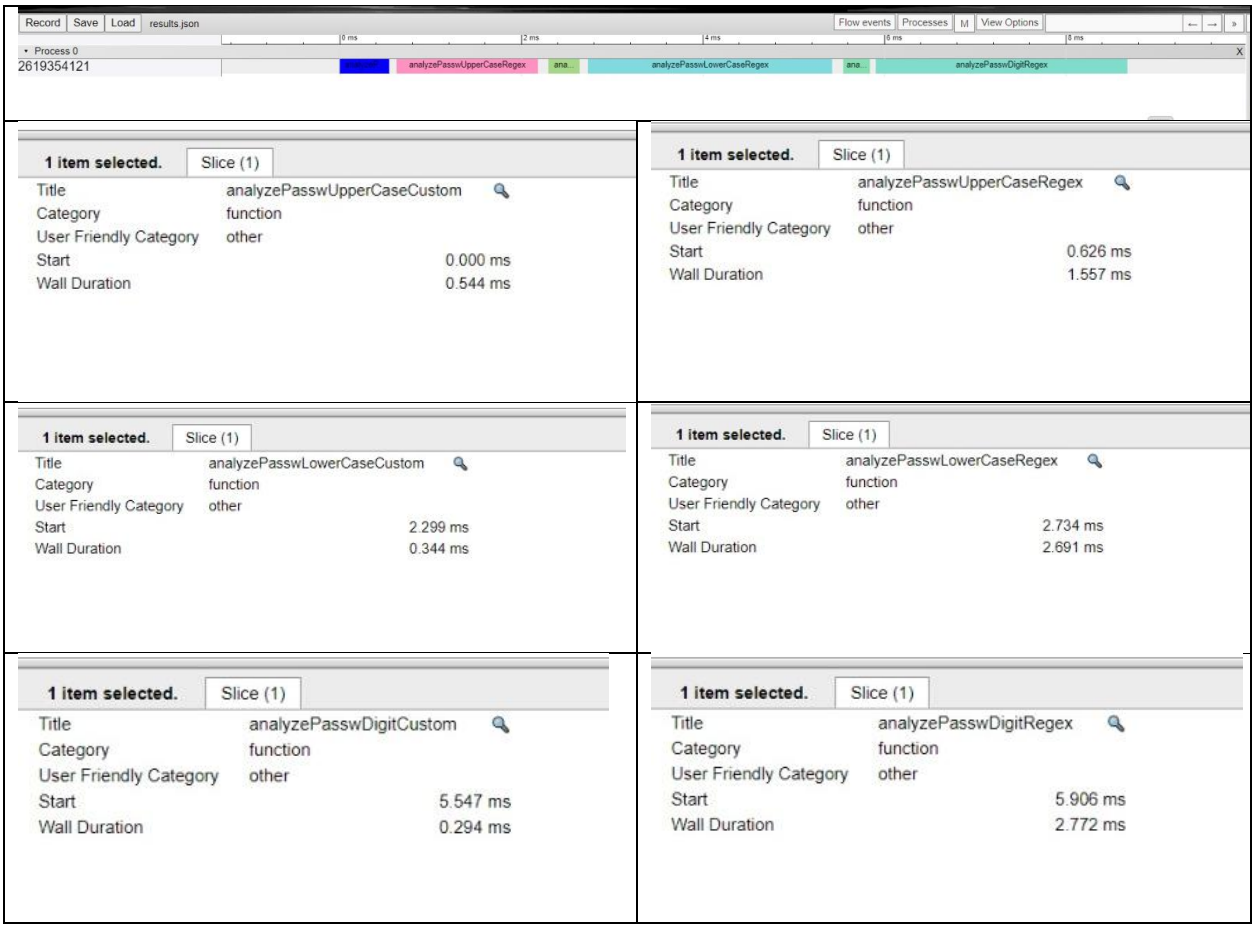
Tabel 1.1

Al doilea tabel prezintă comparația de performanță între validări de tip regex și validări personalizate pentru 5 useri, folosind un *for* cu iterator pe fiecare parolă și functile predefinite din **string**, `isupper, islower, isdigit`.



Tabel 1.2

Al treilea și ultimul tabel prezintă comparația de performanță între validări de tip regex și validări personalizate pentru 10 useri, folosind un *for* cu iterator pe fiecare parolă și functile predefinite din **string**, `isupper, islower, isdigit`.



Tabel 1.3

5. Comentarii asupra rezultatelor

Criteriile studiate au la bază diferențele dintre utilizarea expresiilor regulate (**REGEX**) pentru găsirea atributelor fiecărui parole pentru a fi considerată corectă sau folosirea unei funcții scrise de la zero exact pentru contextul de folosire.

♦ Tabel 1.1

Primul test definește punctul de start al comparației și demonstrează chiar pe un număr foarte mic de utilizatori (1 user) că diferențele de performanță dintre funcțiile personalizate și cele în care se utilizează expresii regulate sunt considerabile și nu ar trebui ignorate în momentul în care se vrea a se optimiza la maxim timpul de execuție. Se observă diferențe chiar de 3 ori între varianta custom de găsire a unei cifre în parolă și cea ce utilizează clasa regex.

♦ Tabel 1.2

În cel de-al doilea test se poate observa că diferențele între variațiile de implementare devin mult mai vizibile. Observăm din analizele făcute, ca diferențele în ceea ce privește fiecare validare în parte sunt deja aproape duble în fiecare caz.

♦ Tabel 1.3

Pentru 10 useri diferențele încep cu adevărat să devină resimțite de către utilizator din punct de vedere al timpului de așteptare, iar acest lucru duce în final la pierderea unui număr considerabil de oameni care utilizează aplicația. Deși acest lucru se observă la nivelul fiecărei funții, cel mai semnificativ test este tot funcția care se ocupă de cautarea cifrei, unde varianta cu regex durează aproape 3 milisecunde, comparativ cu sfertul de milisecundă de așteptare necesară celeilalte implementări.

În testele efectuate *for*-ul bazat pe indice nu este întotdeauna mai eficient, după cum se poate observa în *primul tabel* la ***functia custom de verificare dacă există litere mari***. Pentru funcțiile de dimensiuni mici diferențele sunt oricum nesemnificative, deci nu se poate stabili o variantă optimă de a structura codul. Totuși, ***mai ales pe ultimul test din tabelul 3, dar chiar și pe tot tabelul 3 in sine,*** diferența este notabilă între cele două abordări și poate constitui motivul pentru care s-ar alege o variantă în detrimentul celeilalte, în contextul respectiv ținând cont că se lucrează pe seturi mari de date – aplicația poate ajunge la mii de utilizatori (iar *for*-ul bazat pe index este constant mai rapid cu 2,478 ms aici).

6. Concluzii

Astfel, ce putem observa în urma acestui studiu este că fiecare detaliu de implementare poate deveni important de analizat utilizând un profiler vizual, exemplul de mai sus fiind unul minimal pentru a demonstra importanța acestuia. În general, ca scop, un profiler are utilizarea de a observa lucrul pe mai multe fire de execuție, pe un număr mare de funcții și pe o perioadă îndelungată pentru a putea stabili cu exactitate de unde apare o problemă de performanță, unde s-ar putea face îmbunătățiri și ce poate fi fatal.

Cu toate că se observă că, deși varianta clasică cu iterator este cea recomandată în general, chiar și utilizarea unui *for* bazat pe indice poate duce la câștigarea de timp în contexte critice și merită a fi luat în considerare în momentul profilingului.

[1] Basic Instrumentation Profiler (github.com) - Yan Chernikov

[2] http://labs.cs.upt.ro/labs/SPM/html/SPM10.html - Obiecte unicat (Singleton)

[3] VISUAL BENCHMARKING in C++ (how to measure performance visually) - YouTube - Yan Chernikov

[4] chrome://tracing/ - pentru accesarea profillerului

Grigoraș Ana-Maria - 10LF312

Ioniță Daniel Andrei - 10LF312

Mărica Larisa Anca - 10LF312

Renghea Răzvan Andrei - 10LF313