

Documentație

SISTEM DE MANAGEMENT AL COZILOR



AprilIE 11, 2024

Universitatea Tehnică din Cluj-Napoca

GRIGORAȘ VICTOR-ANDREI

Cuprins

[– Capitolul 1 – 2](#_Toc161770129)

[Cerința 2](#_Toc161770130)

[- Capitolul 2 - 3](#_Toc161770131)

[Proiectare 3](#_Toc161770132)

[Model 4](#_Toc161770133)

[Implementare 4](#_Toc161770134)

[- Capitolul 3 - 7](#_Toc161770135)

[Testarea Unitara 7](#_Toc161770136)

[- Capitolul 4 - 8](#_Toc161770137)

[Concluzie 8](#_Toc161770138)

[Bibliografie 8](#_Toc161770139)

# – Capitolul 1 –

## Cerința

Proiectul se concentrează pe dezvoltarea unei aplicații software specializate în gestionarea cozilor, care va permite utilizatorilor vizualizarea și simularea funcționării unei cozi, precum și implementarea eficientă a diferitelor strategii de gestionare a acesteia. Interfața grafică va fi proiectată pentru a fi intuitivă, facilitând utilizatorilor accesul la funcționalitățile sistemului.

* Procesul debutează prin identificarea și analizarea necesităților utilizatorilor, stabilind cerințe clare.
* Urmează proiectarea detaliată a arhitecturii software și a unei interfețe care să răspundă cerințelor stabilite anterior.
* Testarea are un rol crucial în validarea corectitudinii funcționalităților și în asigurarea stabilității programului.

Analiza problemei și prelucrarea acestora

|  |  |
| --- | --- |
| Analiza nevoilor utilizatorilor | Utilizatorii doresc o aplicație simplă și ușor de utilizat pentru operațiile pe polinoame. |
| Aceștia doresc o interfață intuitivă care să permită introducerea rapidă a polinoamelor și a operațiilor dorite. |
| Cerinte functionale | Aplicația permite simularea sistemului de cozi, inclusiv configurarea numărului de cozi, numărul de clienți și intervalele de așteptare și servire. |
| Cerinte non-functionale | Performanța, ușurința de utilizare, fiabilitatea și corectitudinea |

# - Capitolul 2 -

## Proiectare

|  |
| --- |
|  |

Această aplicație pentru calcularea polinoamelor utilizează un model arhitectural de tip MVC (Model-View-Controller), împărțit în trei componente distincte:

**Model**: Această componentă conține modelele de date de bază, precum Server, unde este rulata folosind fire de execuție . Această structură permite gestionarea cozilor în paralel, si clasa Task unde sunt reținute informații generale despre client.

**Logic**: Aici este inclusă logica programului. Modelul conține clasa SimulationManager, responsabilă de generarea, calcularea și distribuirea clienților în cozi conform strategiei specificate. În plus, aceasta creează o interfață pentru simularea în timp real a activităților și înregistrează detaliile acestora într-un fișier text denumit "output.txt".

**GUI**: Această componentă conține interfața grafică și modelul de afișare a datelor către utilizator. Prin intermediul controllerului, putem obține datele din calculatorul creat și le putem afișa corespunzător use-case-ului în care ne aflăm.

## Model

A screenshot of a computer program

Description automatically generated

## Implementare

În vederea asigurării unei înțelegeri complete a funcționalității Sistemului de managment al cozilor, se va prezenta o descriere formală a fiecărei clase menționate anterior în secțiunile anterioare.

Clasa Server

Clasa Server include funcționalități de coadă și definește operații cum ar fi adăugarea unei sarcini (addTask) și selectarea următorului client (nextClient). Aceasta extinde clasa Thread, permițând paralelizarea clienților în cozi pentru a economisi timp atunci când există mai multe cozi. De asemenea, oferă metode de acces (gettere) pentru a obține totalul timpului de așteptare și totalul timpului de servire, pentru a facilita calculul mediei acestora.

Unele metodele sunt sincronizate pentru a nu exista interferente intre thread-uri. Funcțiile precum addTask și terminate sunt sincrone pentru a preveni coliziunile între thread-uri asupra resurselor comune. Această abordare ajută la reducerea riscului de răspunsuri slabe sau chiar mai rău, erori.

Clasa Task

Clasa Task reprezintă un client care urmează să fie introdus în coadă. Aceasta conține parametri precum id-ul clientului, momentul sosirii (arrival time) și timpul de servire (service time).

Clasa SimulationManager

În proiect, există o clasă numită SimulationManager. Aceasta se ocupă de lucruri importante, cum ar fi pregătirea și distribuirea clienților în cozi și de crearea unei chestii pe ecran care arată ce se întâmplă în timp real. De asemenea, înregistrează tot ce se petrece într-un document numit "output.txt".

Interfața StringOperation

Interfața StringOperation oferă un set de funcții generale care facilitează manipularea datelor, făcându-le ușor de înțeles. Aceste funcții sunt salvate atât într-un fișier pentru stocare pe termen lung, cât și integrate în interfața grafică (GUI), asigurându-se astfel că sunt accesibile și utilizatorilor care interacționează cu aplicația.

Clasa SimulationInterface

A screenshot of a computer

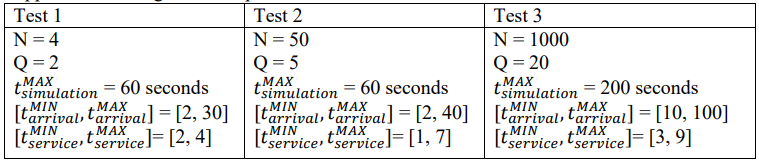
Description automatically generatedSimulationInterface este interfața prin care utilizatorul va interacționa cu simularea. Acesta include un JTable care afișează starea cozilor din simulare, oferind informații despre numele cozii, clienții care se află în coadă și starea lor (activă sau inactivă). Cozile sunt colorate cu verde dacă mai au locuri disponibile pentru clienți și cu roșu dacă sunt pline. De asemenea, sunt afișate informații referitoare la numărul total de cozi, clienți, timpul și numărul de clienți rămași, precum și o secțiune în care sunt listate clienții care așteaptă să fie adăugați în cozi.

# - Capitolul 3 -

## Testarea Unitara

Pentru a asigura acuratețea și eficiența sistemului, am efectuat teste pe mai multe categorii de seturi de date. Am împărțit datele în trei categorii distincte: mici, medii și mari. Aceste categorii au fost selectate pentru a acoperi o gamă largă de scenarii posibile și pentru a evalua performanța sistemului în diferite condiții.

În cadrul fiecărei categorii de seturi de date, am testat ambele strategii disponibile în sistem. Acest lucru ne-a permis să comparăm performanța și eficacitatea fiecărei strategii în diferite contexte. Testele au inclus diverse scenarii de utilizare și sarcini diferite pentru a simula condiții variate de funcționare a sistemului. Această practică este crucială pentru verificarea și validarea continuă a acurateței și integrității fiecărei componente individuale a aplicației.



A screenshot of a computer

Description automatically generated

# - Capitolul 4 -

## Concluzie

În concluzie, proiectul de implementare a calculatorului de polinoame a fost o oportunitate valoroasă de învățare, evidențiind două aspecte cheie:

* **Lizibilitatea codului**: Am constatat importanța redactării unui cod clar și ușor de înțeles. Prin utilizarea denumirilor semnificative pentru variabile, adăugarea comentariilor explicative și structurarea logică a codului, am reușit să îmbunătățesc lizibilitatea acestuia și să facilităm întreținerea ulterioară.
* **Transformarea problemelor în subprobleme**: Am învățat să abordez problemele complexe prin împărțirea lor în subprobleme mai mici și mai ușor de rezolvat. Această strategie modulară ne-a permis să gestionăm dezvoltarea în mod eficient și să menținem un cadru organizat al proiectului.
* **Gestionarea eficientă a proceselor simultane:** Am învățat să utilizez threaduri pentru a gestiona calculele simultane și pentru a îmbunătăți performanța aplicației prin distribuirea sarcinilor între diferite fire de execuție.

Aceste aspecte reprezintă atât competențe tehnice esențiale, cât și abilități practice necesare pentru dezvoltarea și întreținerea aplicațiilor software de succes.

## Bibliografie

<https://dsrl.eu/courses/pt/>

<https://www.geeksforgeeks.org/>