**Universitatea Tehnică “Gheorghe Asachi” din Iași**

**Facultatea de Automatică și Calculatoare**

**Domeniul Calculatoare și Tehnologia Informației**

**Specializarea Tehnologia Informației**

# 

**EVALUAREA PERFORMANŢELOR**

**TEMĂ DE CASĂ**

**Motfolea Silviu-Andrei, 1410B**

**Susanu Alexandru-Cătălin, 1410B**

**Vornicu Floris-Diana, 1410B**

**An universitar 2021-2022**

**Documentaţia aplicaţiei**

**Aplicaţie pentru gestiunea clienţilor unei firme furnizoare de servicii TV şi a pachetelor TV oferite**

**Conținutul documentului**

[**1.** **Enunţarea temei** 3](#_Toc89004665)

[**2.** **Arhitectura aplicatiei** 3](#_Toc89004666)

[**3.** **Functionalitatea** 4](#_Toc89004667)

[**4.** **Rolul fiecarui membru al echipei** 5](#_Toc89004668)

[**5.** **Analiza complexitatii si a corectitudinii unui algoritm** 5](#_Toc89004669)

[**6.** **Explicatia testelor automate create pentru demonstrarea bunei functionari a programului** 6](#_Toc89004670)

7. **Explicatii suplimentare legate de complexitatea algoritmului InsertionSort** 7

# **Enunţarea temei**

Aplicatia modeleaza activitatea unei firme furnizoare de servicii TV cu privire la gestiunea clientilor, a grilei de programe si a pachetelor TV oferita de catre firma. Mai exact, utilizatorul va avea posibilitatea de a introduce noi clienti sau de a sterge clientii existenti din baza de date, de a introduce noi programe TV in grila, de a crea noi pachete TV si de a introduce programele aferente acestora.

# **Arhitectura aplicatiei**

Fiind o aplicatie desktop ce permite stocarea anumitor informatii intr-o baza de date locala, arhitectura este una minimala continand aplicatia monolit care a fost creata utilizand Java 8, iar pentru interfata grafica a fost utilizata platforma JavaFX. Aplicatia este conectata la o baza de date Oracle, pentru realizarea conexiunii folosindu-se libraria ojdbc8.jar .

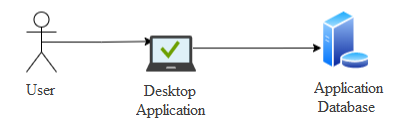


Fig 1. Arhitectura aplicatiei

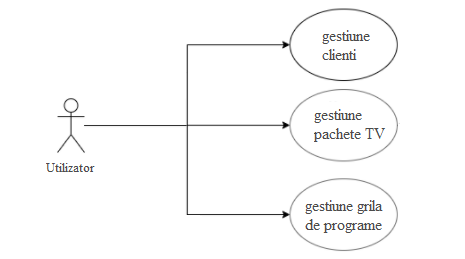


Fig 2. Diagrama UML de tip use-case

Pentru design-ul aplicatiei a fost utilizat pattern-ul MVC (Model-View-Controller), astfel realizandu-se o separare clara intre date, interfata grafica si logica de business.

Pentru realizarea modelului s-a utilizat o baza de date Oracle in care au fost create tabele pentru stocarea datelor despre clienti, pachete si grila de programe, dar si un tabel in care se vor stoca relatiile dintre pachetele TV si posturile aferente acestora. La nivel de aplicatie, accesul la baza de date se realizeaza prin intermediul unei clase handler care a fost creata utilizand design pattern-ul Singletone intrucat este necesara o singura instanta a acestei clase, iar aceasta instanta trebuie sa aiba vizibilitate globala.

Pentru realizarea view-ului au fost create diferite fisiere fxml care contin codul aferent fiecarei menu al aplicatiei. Un astfel de fisier contine informatii despre componente grafice precum butoanele din aplicatie, tabele, liste, input-uri etc. In acest mod fiecarei componente ii este asignat un id unic si o anumita functionalitate care este definita la nivel de controller si care este apelata oridecateori utilizatorul apasa un buton sau modifica textul dintr-un input, de exemplu. Realizarea UI-ului a fost usurata de utilizarea tool-ului JavaFX Scene Builder care permite crearea diferitelor interfete grafice doar prin comenzi de tip drag and drop.

La nivel de controller, dupa cum am precizat si anterior, este definita logica de business sau functionalitatile aferente componentelor grafice. Controller-ul actioneaza ca un coordonator intre View si Model, primind input de la utilizator prin intermediul View-ului, procesand acest input pentru a putea fi stocat permanent la nivelului modelului si returnand rezultatul pentru a putea fi afisat de catre View. In cazul nostru, controller-ul este cel care permite trecerea prin diferitele scene ale aplicatiei, permite crearea de noi entitati precum clientii sau posturile TV.

# **Functionalitatea**

Volumul mare de informaţii existente în cadrul unei firme cu numerosi clienţi determină necesitatea fluidizarii fluxurilor de date ce traverseaza sistemele informationale ale acesteia, gestiunea acestora fiind o mare provocare. Aplicatia se va ocupa de gestiunea unei firme furnizoare de servicii TV, ceea ce implică o muncă intensă privind gestiunea clienţilor sau a posturilor TV aferente pachetelor de programe oferite de catre firma.

Informatiile de care avem nevoie sunt cele legate de :

* **clienti:** pentru identificarea fiecarui client trebuie sa-i cunoastem id-ul (acesta este generat automat ) si numele, un numar de telefon prin intermediul caruia clientul sa poata fi informat cu privire la promotiile existente si, optional, o adresa de e-mail.
* **pachete**: pachetele vor fi identificare prin intermediul unui id de pachet, iar fiecare pachet va avea o denumire care va fi unica. Vor mai exista de asemenea o data de start, care va preciza data la care firma a inceput sa ofere pachetul respectiv, si o data de end care va preciza data la care firma nu va mai oferi spre serviciu pachetul respectiv, dar si un camp care va indica pretul fiecarui pachet.
* **posturi**: tabela posturi se afla in relatie de unu la unu cu tabela detalii\_posturi si in acestea vor fi salvate informatii precum id-ul postului, denumirea, data la care postul a fost introdus in grila de posturi a firmei, data la care postul a fost scos din grila, tipul postului (analogic/digital/HD/4K). Suplimentar se vor mai salva frecventa la care acesta este transmis, dar si canalul din lista de programe la care acesta poate fi gasit.

Principalele funcţii indeplinite de firma sunt:

* evidenţa clienţilor
* evidenţa pachetelor oferite spre serviciu
* evidenţa grilei de posturi

# **Rolul fiecarui membru al echipei**

* X
* Y
* Z

# **Analiza complexitatii si a corectitudinii unui algoritm**

Algoritmul supus analizei este algoritmul de sortare Insertion Sort care permite sortarea alfabetica a unei liste de programe TV in functie de denumirea acestora.

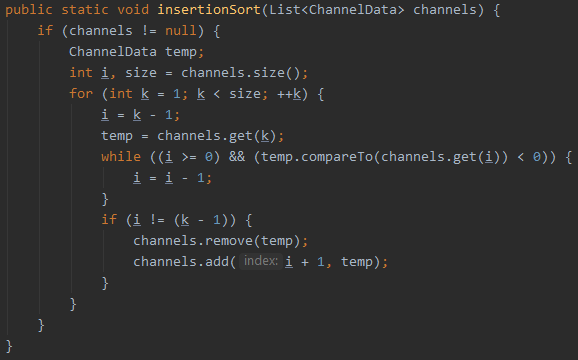


Fig 3. Implementare a algoritmului Insertion Sort

Preconditii: o lista care sa contine cel putin 2 elemente de sortat, n >= 2.

Postconditii: elementele listei sa fie sortare corespunzator.

Pentru determinarea complexitatii s-au luat in vedere operatiile de comparare.

Complexitatea in cazul favorabil: 1 + (n - 1) \* 4 = 4n – 3

Complexitatea in cazul nefavorabil: 1 + (n - 1) + (n\*(n-1)/2)\*2 - (n-1) + (n-1) = n + n \* ( n - 1) = n + n2 – n = n2

Cazul favorabil corespunde situatiei in care lista este deja sortata, algoritmul realizand o simpla iterare asupra elementelor listei, iar cazul nefavorabil corespunde situatiei in care lista este sortata descrescator.

Complexitatea in cazul mediu: ((4n – 3) + n2 ) / 2 = (n2 + 4n – 3) / 2

Clasele de complexitate: Ω(n), O(n2).

Pentru simplitate s-a facut abstractie de apelul functiilor channels.get(i), channels.remove(temp), channels.add(i+1, temp), clasele de complexitate ramand aceleasi.

Invariantul pentru bucla for: in urma pasului k elementele de pe pozitiile 0, 1 … k sunt sortate corespunzator.

# **Explicatia testelor automate create pentru demonstrarea bunei functionari a programului**

Pentru testarea aplicatiei a fost utilizata libraria JUnit4, astfel au fost create 20 de teste unit care sa testeze logica de business implementata in diferite module.

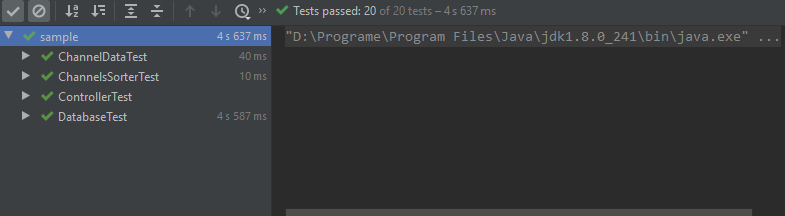


Fig 4. Rezultatul rularii testelor functionale create

Luand in considerare faptul ca aplicatia contine 3 module, cate unul pentru clienti, pachete si programe TV, iar logica de business pentru aceste module este asemanatoare, s-a decis testarea modului ce se ocupa de gestiunea grilei de programe. Astfel, au fost create teste pentru testarea:

* algoritmului InsertionSort
* controller-ului modulului de programe tv
* clasei ChannelData, astfel incat compararea a 2 obiecte de acest tip sa se realizeze corespunzator
* comunicatiei cu baza de date

Pentru testarea algoritmului InsertionSort s-a verificat ca elementele sunt sortate corespunzator, ca programul isi continua rularea in cazul in care lista primita de algoritm este nula sau ca sortarea are loc cu succes in cazul in care denumirea unui program TV incepe cu o cifra samd.

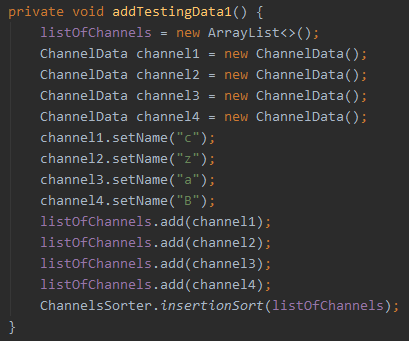


Fig 5. Date supuse testarii

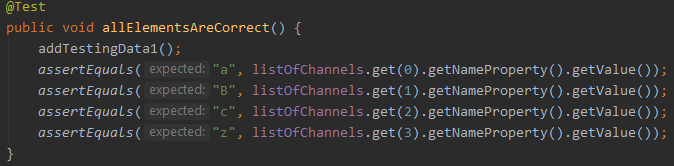


Fig 6. Test ce verifica corectitudinea rularii algoritmului InsertionSort

In ceea ce priveste comunicatia cu baza de date, s-a verificat faptul ca nu pot fi stocate programe care nu contin date valide sau date complete. De exemplu, baza de date nu permite crearea unui unui program TV daca acestuia nu ii este setat un nume, sau daca data de adaugare in grila a programului TV nu este precizata sau daca numele programului TV nu are lungimea corespunzatoare.

1. **Explicatii suplimentare legate de complexitatea algoritmului InsertionSort**

Conform analizei de la punctul 5 numarul de operatii de comparare in cazul favorabil este

4 \* (n - 1) + 1 = 4n - 3. Acest rezultat a fost obtinut in modul urmator: pentru o lista cu n elemente care este deja sortata bucla for va fi executa de n – 1 ori deoarece algoritmul InsertionSort considera ca primul elemente este deja sortat. In aceasta bucla for se realizeaza 4 operatii de comparare, de aici produsul 4 \* (n - 1), la care se mai adauga si verificarea initiala ca lista nu este nula => 4n – 3.

Numar de operatii in cazul nefavorabil: 1 + (n - 1) + (n\*(n-1)/2)\*2 - (n-1) + (n-1) = n2

Pentru cazul nefavorabil, adica atunci cand lista este sortata descrescator, bucla for va fi executata de asemenea de n – 1 ori (lista continand n elemente).

Termenul 1 + (n - 1) apare deoarece se verifica initial ca lista nu este vida, iar compararea din bucla for are loc de (n -1) ori => 1 + (n - 1).

Termenul (n\*(n-1)/2) apare deoarece la fiecare pas al buclei for numarul de comparari (i >= 0) va creste cu 1.

Exemplu:

* k = 1, se face o comparare (i >= 0)
* k = 2, se fac 2 comparari (i >= 0)

.

.

* k = n – 1, se fac n – 1 comparari (i >= 0)

Astfel, avem de-a face cu o suma a numerelor naturale de la 1 la n – 1, rezultand un numar de operatii de comparare egal cu (n\*(n-1)/2).

Acest termen este inmultit cu 2 deoarece in bucla while pe langa compararea (i >= 0) se mai realizeaza si compararea (temp.compareTo(channels.get(i)) < 0), rezultand astfel (n\*(n-1)/2)\*2.

Din acest rezultat se scade (n - 1) deoarece cea de-a 2 comparare din bucla while nu are loc in cazul in care rezultatul compararii (i >= 0) este false, acest lucru intamplandu-se de (n - 1) ori, adica de cate ori se executa bucla for => (n\*(n-1)/2)\*2 - (n-1).

La rezultat se mai ia in considerare si faptul ca acea comparare if(i != (k - 1)) are loc de (n – 1) ori => (n\*(n-1)/2)\*2 - (n-1) + (n-1).

Intr-un final adunand si compararile corespunzatoare buclei for si conditiei if initiale numarul de operatii este 1 + (n-1) + (n\*(n-1)/2)\*2 - (n-1) + (n-1) = n2.