



Universitatea  
Transilvania  
din Brașov

FACULTATEA DE MATEMATICĂ  
ȘI INFORMATICĂ

## **Lucrare de licență**

### **Aplicație de analizare a testelor automate**

**Autor:** Sdrăilă Valentin-Nicușor

**Mentori:** Duron Hendrik (SISW)  
Gothard Eduard (SISW)  
Daicu Raluca (SISW)

**Coordonator:** Lect. Dr. Băicoianu Alexandra

Brașov, 2023

# Cuprins

Rezumat .....	4
Lista de figuri.....	5
Lista de abrevieri.....	6
<b>1</b> Introducere .....	7
<b>2</b> <b>Aspecte tehnice</b> .....	8
2.1 Procesarea imaginilor digitale .....	8
2.2 Machine learning .....	11
<b>3</b> <b>Tehnologii folosite</b> .....	14
3.1 Limbaje de programare .....	14
3.1.1 Limbajul de programare C# .....	14
3.2 Limbajul de interogare SQL.....	14
3.3 Limbaje de marcare .....	15
3.3.1 Limbajul de marcare XML .....	15
3.3.2 Limbajul de marcare XAML.....	16
3.4 Biblioteci .....	16
3.4.1 Tesseract.....	16
3.4.2 OpenCvSharp .....	17
3.4.3 Interop.Excel .....	17
3.4.4 Kernel32 .....	17
3.4.5 ADO.NET .....	18
3.4.6 MahApps.Metro.....	19
3.5 NuGet Package Manager .....	19
3.6 WPF.....	20
3.7 SQL Server .....	21
3.8 LINQ .....	23
<b>4</b> <b>Implementare</b> .....	25
4.1    Aplicația componentă Watson .....	26
4.1.1 Clasa ATGroup.....	27
4.1.2 Clasa WatsonMain .....	29
4.2 Biblioteca SherlockUtil.....	29
4.2.1 Clasa ATReport.....	31

4.2.2 Clasa ErrorRule.....	33
4.2.3 Clasa FailedTests .....	33
4.2.4 Interfața IMessage .....	34
4.2.5 Clasa OutputFile.....	34
4.2.6 Clasa ReadWriteIniFile .....	36
4.2.7 Clasa TestObject .....	36
4.3 Aplicația componentă Sherlock .....	38
4.3.1 Clasa ConsoleMessage.....	39
4.3.2 Clasa SherlockMain.....	39
4.4 Componenta de interfață grafică SherlockUI .....	40
4.4.1 Clasele ListItem și LogMessage.....	41
4.4.2 Clasa WindowMessage .....	42
4.4.3 Clasele INIDialogViewModel și EditINIDialog .....	43
4.4.4 Clasele MainWindowViewModel și MainWindow .....	44
4.4.5 Clasa ViewModelBase .....	46
<b>5. Ghid de utilizare al aplicației .....</b>	<b>48</b>
<b>6. Concluzii.....</b>	<b>51</b>
6.1 Concluzii generale.....	51
6.2 Dezvoltare ulterioară .....	52
<b>7. Bibliografie .....</b>	<b>53</b>

# Rezumat

În fiecare zi, pentru mai multe echipe din cadrul SIEMENS Industry Software, o platformă de lucru internă rulează un lot de teste automate împotriva noilor compilații, care se întind pe mai multe lansări și versiuni. Pe timpul rulării testelor automate pot apărea anomalii care pot duce la eșuarea acestora, caz în care platforma de lucru semnalează acest lucru în baza de date, unde marchează testul ca eșuat, dar și în fișierele de ieșire ale testelor, unde poate adăuga anumite mesaje de eroare.

În prezent, rezultatele acestor teste trebuie verificate manual în fiecare zi. Acest lucru este realizat de dezvoltatorii din cadrul echipei conform unui rol în rotație, un sistem numit "jetonul AT". Persoana care are jetonul parcurge aceste rezultate și le grupează după lansare și după versiune, adăugând o scurtă notificare a tipului de defecțiune și, dacă este cunoscută, cauza. Aceste informații sunt găzduite într-o foaie Excel care este partajată echipei prin Microsoft Teams. Se adaugă, de asemenea, un responsabil pentru fiecare test nereușit pentru a investiga și/sau a remedia în continuare, sau pentru a identifica persoana care este eligibilă să facă acest lucru. Persoana cu jetonul poate fi nevoită să petreacă o cantitate substanțială de timp pentru a finaliza această investigație zilnică, în funcție de numărul de teste eșuate și de neclaritatea cauzelor eșecului.

Scopul aplicației este de a automatiza această sarcină, de a elimina necesitatea acestei investigații manuale și a jetonului AT prin intermediul creării unui raport în interiorul unei foi de lucru Excel cu tot felul de informații relevante despre testele automate care au eșuat.

# Lista de figuri

Figura 2.1: Transformata Gauss .....	9
Figura 2.2: Exemplu de aplicare a filtrului Gauss .....	10
Figura 2.3: Thresholding histeretic .....	10
Figura 2.4: Exemplu de aplicare a filtrului Canny .....	11
Figura 2.5: Model OCR general .....	12
Figura 3.1: Interfața grafică a administratorului de pachete NuGet .....	20
Figura 3.2: Diagrama UML a bazei de date AutomatedTesting .....	22
Figura 4.1: Flow-ul aplicației .....	25
Figura 4.2: Diagrama UML a clasei ATGroup .....	29
Figura 4.3: Diagrama UML a clasei WatsonMain .....	29
Figura 4.4: Diagrama UML a clasei ATReport.....	32
Figura 4.5: Diagrama UML a clasei ErrorRule.....	33
Figura 4.6: Diagrama UML a interfeței IMessage .....	34
Figura 4.7: Diagrama UML a clasei ReadWriteINIfile .....	36
Figura 4.8: Diagrama UML a clasei TestObject .....	37
Figura 4.9: Diagrama UML a clasei ConsoleMessage.....	39
Figura 4.10: Diagrama UML a clasei SherlockMain.....	39
Figura 4.11: Prezentare MVVM.....	40
Figura 4.12: Diagrama UML a clasei ListItem .....	42
Figura 4.13: Diagrama UML a clasei LogMessage .....	42
Figura 4.14: Diagrama UML a clasei WindowMessage .....	42
Figura 4.15: Diagrama UML a clasei INIDialogViewModel.....	43
Figura 4.16: Diagrama UML a clasei MainWindowViewModel .....	45
Figura 4.17: Realizarea legăturii dintre MainWindowViewModel și MainWindow prin intermediul limbajului XAML, extrasă din view-ul MainWindow .....	46
Figura 4.18: Diagrama UML a clasei ViewModelBase.....	47
Figura 5.1: Captură a aplicației consolă Sherlock .....	48
Figura 5.2: Mesaje de logging apărute în timpul executării aplicației.....	49
Figura 5.3: Marcarea unui test pentru reprogramare .....	50

# Lista de abrevieri

<b>OCR</b>	Optical Character Recognition
<b>SQL</b>	Structured Query Language
<b>XML</b>	Extensible Markup Language
<b>XAML</b>	Extensible Application Markup Language
<b>WPF</b>	Windows Presentation Foundation
<b>UML</b>	Unified Modeling Language
<b>MVVM</b>	Model View ViewModel
<b>DLL</b>	Dynamic Link Library
<b>AT</b>	Automated Tests
<b>UI</b>	User Interface
<b>ADO</b>	ActiveX Data Object
<b>LINQ</b>	Language Integrated Query

# 1 Introducere

Scopul aplicației este de a automatiza procesul de generare a rapoartelor pentru rezultatele testelor automate rulate zilnic cu diverse configurații. Aplicația are rolul de a simplifica și ușura analiza acestor rezultate, oferind un raport structurat și accesibil. Prin utilizarea aplicației, utilizatorii pot evita munca manuală de colectare a datelor și de întocmire a rapoartelor în mod manual. Aplicația preia rezultatele testelor automate și le procesează într-un mod coerent, astfel încât să poată fi prezentate într-un raport ușor de înțeles.

Raportul generat conține informații relevante despre rezultatele testelor automate eșuate, precum configurațiile utilizate, mesajele de eroare obținute, numele membrului echipei care este responsabil pentru test și orice alte detalii importante. Acest raport poate fi vizualizat și analizat de către utilizatori pentru a obține o înțelegere rapidă și eficientă a stării testelor automate și a eventualelor probleme sau îmbunătățiri necesare.

Această aplicație software conține patru componente principale: Watson, Sherlock, SherlockUI și SherlockUtil fiecare componentă fiind reprezentată de câte un proiect dezvoltat în limbajul de programare C#.

**Watson** reprezintă o aplicație consolă a cărei scop este colectarea datelor referitoare la testele automate din baza de date și aducerea lor într-un fișier XML care să fie folosit mai departe de celelalte componente pentru a crea raportul.

**SherlockUtil** este o bibliotecă dinamică de clase care conține majoritatea logicii aplicației, aici se află codul sursă pentru apelarea lui Watson, preluarea datelor din fișierul XML, dar și totalitatea procesărilor de date folosite pentru a ajunge la rezultatul final, și anume raportul testelor automate.

**Sherlock** este o aplicație consolă care se folosește de logica din SherlockUtil pentru a efectua sarcinile necesare creării raportului fără a avea nevoie de input de la utilizatori.

**SherlockUI** reprezintă opțiunea de a interacționa cu soluția software printr-o interfață grafică, în contrast cu Sherlock, care nu necesită nicio interacțiune. SherlockUI a fost dezvoltat cu ajutorul platformei de lucru WPF(Windows Presentation Foundation), folosind design pattern-ul Model View ViewModel. Componenta de interfață grafică oferă aceleași funcționalități pentru crearea raportului ca și **Sherlock**, dar conține în plus câteva componente de feedback și posibilitatea de a modifica input-ul.

## 2 Aspecte tehnice

În contextul acestui studiu, directoarele de ieșire asociate rezultatelor testelor automate eșuate pot conține în unele cazuri și imagini. Asupra acestor imagini sunt aplicate tehnici de procesare a imaginilor și tehnici de machine learning pentru a obține în cadrul codului sursă informațiile relevante din acestea. În continuare vor fi prezentate informații despre aceste două domenii de programare.

### 2.1 Procesarea imaginilor digitale

O imagine este o reprezentare vizuală a unui obiect, unei persoane sau a unei scene. O imagine digitală este o funcție bidimensională  $f(x,y)$  care reprezintă proiecția unei scene tridimensionale pe un plan de proiecție bidimensional, unde  $x$ ,  $y$  reprezintă locația elementului imagine sau pixel și conține valoarea intensității. Atunci când valorile lui  $x$ ,  $y$  și intensitatea sunt discrete, atunci imaginea este numită imagine digitală [1].

Printre cele mai frecvent întâlnite tipuri de imagini digitale se numără imaginile în tonuri de gri (grayscale), imagini color și imagini binare. În aceste trei tipuri de imagini digitale, fiecare pixel are o valoare numerică care reprezintă intensitatea culorii în acel punct. În cazul imaginilor în tonuri de gri, pixelii pot avea valori de la 0 la 255, unde 0 este o culoare foarte închisă, iar 255 este o culoare deschisă. Imaginile color au câte trei valori stocate pentru fiecare pixel, acestea reprezentând cantitatea de roșu (R), cantitatea de verde (G) și cantitatea de albastru (B). Imaginile binare sunt imagini care conțin doar două valori de intensitate, de obicei 0 și 1. Acestea sunt utilizate în special în aplicații de segmentare, unde se dorește separarea obiectelor de fundal.

Procesarea de imagini se referă la utilizarea algoritmilor și tehnicilor pentru a manipula și transforma imaginile digitale într-un mod semnificativ. Aceasta implică aplicarea unei serii de operații, filtre și analize pentru a extrage, îmbunătăți sau interpreta informațiile vizuale conținute într-o imagine.

Procesarea de imagini poate avea mai multe obiective, inclusiv:

- **Îmbunătățirea calității imaginii:** Aplicarea de tehnici de reducere a zgomotului, corecția contrastului, îmbunătățirea clarității sau corectarea distorsiunilor pentru a obține o imagine mai bună și mai ușor de analizat.
- **Detectarea și extragerea de caracteristici:** Identificarea și extragerea anumitor caracteristici sau obiecte din imagine, cum ar fi margini, contururi, regiuni de interes, puncte cheie sau modele specifice.
- **Segmentarea imaginii:** Separarea imaginii în regiuni distincte sau obiecte individuale bazate pe caracteristici specifice, cum ar fi culori, texturi sau intensități.



Imaginile digitale devin destul de des corupte de zgomot, concept care se referă la informațiile nedorite sau erorile adăugate în imagine. Acesta poate apărea în timpul obținerii, transmiterii sau chiar în timpul reproducerii imaginii.

Zgomotul poate afecta claritatea, detaliile și calitatea generală a unei imagini digitale. Prin urmare, în procesarea de imagini, este de multe ori necesar să fie aplicate tehnici de reducere a zgomotului pentru a îmbunătăți calitatea imaginii și pentru a obține o reprezentare mai precisă a informațiilor vizuale.

În realizarea acestei aplicații s-au folosit împreună următoarele două filtre pentru a identifica căsuțele de eroare prezente în imaginile din directoarele de ieșire corespunzătoare testelor automate eșuate.

**Filtrul Gauss** [2] este un tip de filtru utilizat în procesarea de imagini pentru a realiza o estompare sau o blurare a imaginii. Este numit după matematicianul Carl Friedrich Gauss, care a dezvoltat funcția Gaussiană asociată acestui filtru.

$$\vec{v}_i = \sum_j e^{-|\vec{p}_i - \vec{p}_j|^2/2} \vec{v}_j$$

Figura 2.1: Transformata Gauss [2]

În **Figura 2.1** se poate observa o transformată Gauss unde valorile de ieșire din  $\vec{v}_i$  sunt o sumă ponderată a valorilor de intrare  $\vec{v}_j$ , cu ponderile determinate de o distribuție Gaussiană în distanța dintre două poziții asociate  $\vec{p}_i$  și  $\vec{p}_j$ .

**Filtrul Gauss** aplică o operație de convoluție între imaginea originală și o matrice Gaussiană sau un kernel Gauss. Acest kernel este o matrice de valori ponderate, care este de obicei de dimensiune impară și simetrică, având cea mai mare valoare în centrul matricei.

**Filtrul Gauss** are capacitatea de a estompa zgomotul și de a reduce detaliile fine din imagine, ducând la o imagine mai netedă și mai uniformă. Este adesea utilizat în preprocesarea imaginilor înainte de aplicarea altor algoritmi sau tehnici, cum ar fi detecția de margini sau segmentarea imaginii.

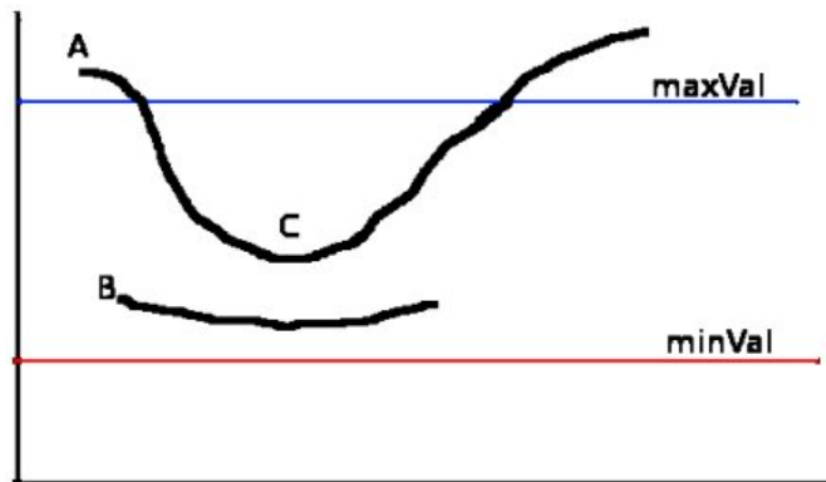


Figura 2.2: Exemplu de aplicare a filtrului Gauss [2]

**Filtrul Canny** [3] este o tehnică populară de detectare a marginilor în imagini. A fost dezvoltat de John F. Canny în 1986 și a devenit unul dintre cei mai utilizați algoritmi de detecție a marginilor în domeniul procesării imaginilor.

Aplicarea algoritmului de detectare a marginilor Canny este împărțită în cinci etape diferite:

1. Aplicarea unui filtru Gaussian pentru a netezi imaginea și a elimina zgomotul.
2. Determinarea gradientelor de intensitate ai imaginii.
3. Aplicarea unei praguri asupra magnitudinii gradientelor sau suprimarea inferioară a rezultatelor nesemnificative la detectarea marginilor.
4. Aplicarea unui prag dublu pentru a determina marginile potențiale.
5. Urmărirea marginilor prin histereză: Finalizarea detectării marginilor prin suprimarea tuturor celorlalte margini slabe care nu sunt conectate la margini



puternice.

Figura 2.3: Thresholding histeretic [3]

În **Figura 2.3** se observă cum este folosit thresholding-ul histeretic. Acest ultim pas decide care margini sunt cu adevărat margini și care nu. Pentru această acțiune, sunt folosite două valori de threshold, minVal și maxVal. Oricare margine care are gradientul de intensitate mai mare decât maxVal este automat considerată o margine „sigură” și este păstrată, iar orice margine care are gradientul de intensitate mai mic decât minVal este considerată non-margine și este eliminată. Marginile cu valori ale gradientului aflate între cele două valori sunt considerate margini sau non-margini pe baza legăturii lor. În cazul în care acestea sunt legate de o margine sigură( A), marginile aflate între cele două valori vor fi considerate și ele margini (C). În caz contrar, vor fi și ele eliminate (B).

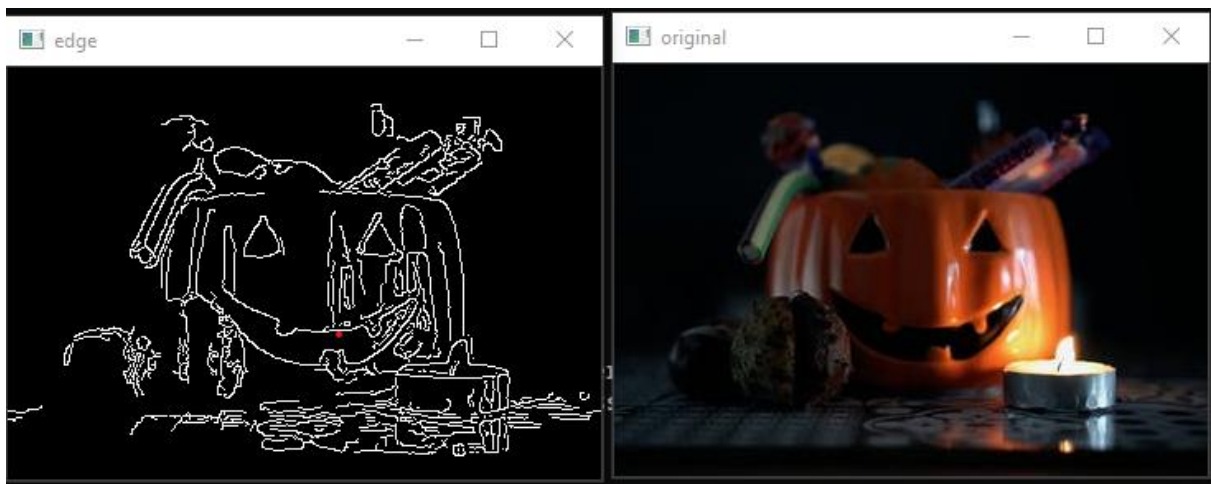


Figura 2.4: Exemplu de aplicare a filtrului Canny [4]

## 2.2 Machine learning

Machine learning [5] este o ramură a inteligenței artificiale care se ocupă cu dezvoltarea de algoritmi și modele care permit sistemelor să învețe și să facă predicții pe baza datelor disponibile, fără a fi explicit programate. Altfel spus, machine learning este o tehnică care îmbunătățește performanțele sistemului învățând din experiențele sale precedente, asemănător cu modul de a învăța al oamenilor. Scopul principal al machine learning este de a permite sistemelor să descopere și să extragă modele sau relații din datele de intrare, pentru a face predicții, clasificări sau alte tipuri de analize.

În loc să fie programate în mod tradițional, sistemele de machine learning sunt antrenate pe seturi de date, folosind algoritmi și modele matematice. Aceste seturi de date pot conține exemple de intrare și rezultate așteptate, pe baza cărora algoritmiile învață să facă conexiuni și să efectueze calcule. Odată ce sistemul a fost antrenat, poate

fi utilizat pentru a face predicții sau a lua decizii pe baza datelor noi, care nu au fost folosite în procesul de antrenare.

Machine learning poate fi aplicat într-o varietate de domenii, cum ar fi recunoașterea vocală, recunoașterea vizuală, traducerea automată, analiza datelor, recomandări personalizate și multe altele.

### 2.2.1 OCR

**OCR** (Optical Character Recognition) [6] este o aplicație a machine learning care permite recunoașterea automată a caracterelor de către calculator prin intermediul unui mecanism optic. **OCR** este o tehnologie care funcționează ca abilitatea oamenilor de a citi, deși încă nu este comparabilă cu aceasta.

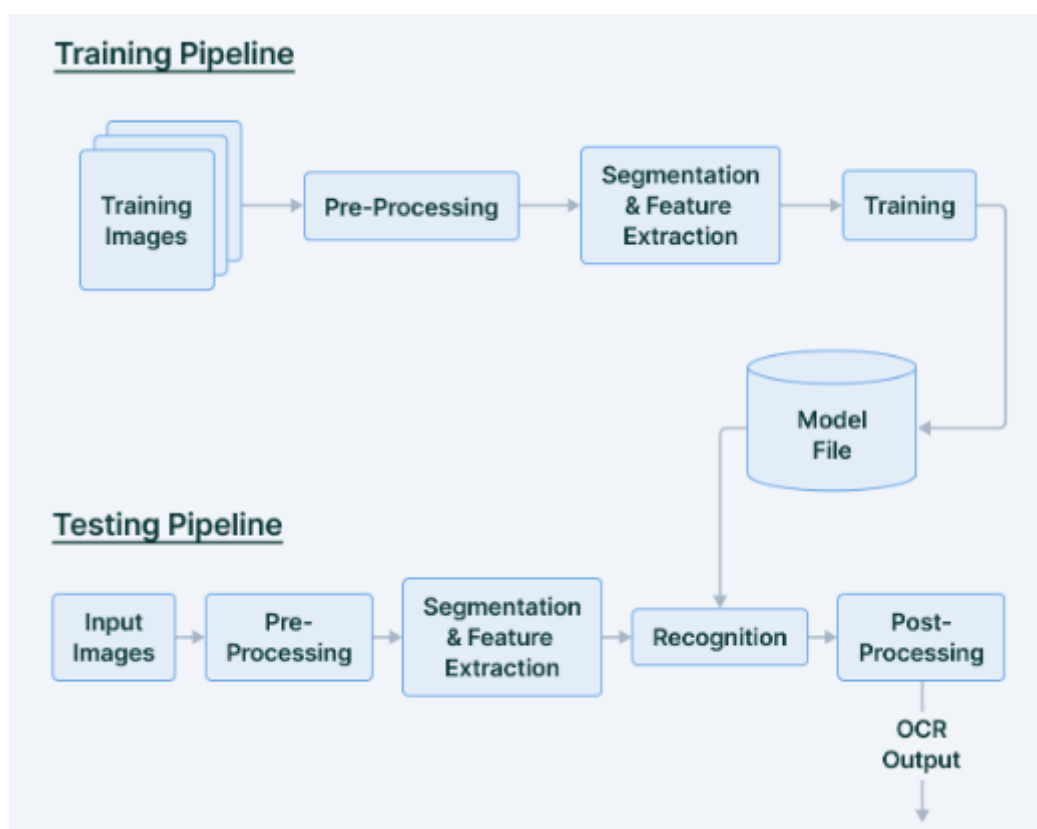


Figura 2.5: Model OCR general [7]

În **Figura 2.5** este prezentată derularea antrenării și testării tehnicii de recunoaștere optică a caracterelor. În aceasta se poate observa cum modelul antrenat este folosit mai departe în cadrul algoritmului de recunoaștere a caracterelor.

În contextul acestei aplicații software, **OCR** este folosit pentru a extrage textul din căsuțele de eroare detectate cu ajutorul **filtrului Canny** prezente în imaginile din interiorul directoarelor de ieșire a testelor eșuate.

## 3 Tehnologii folosite

### 3.1 Limbaje de programare

Există o multitudine de limbaje de programare care pot facilita scrierea de cod pentru realizarea unei aplicații. Acestea implementează mai multe paradigme de programare, cum ar fi programarea orientată pe obiecte, programarea procedurală sau programarea funcțională pentru a aborda diferite tipuri de probleme și pentru a atinge obiectivul dorit de către programator.

Alegerea limbajului de programare potrivit pentru dezvoltarea soluției este un pas foarte important, întrucât acesta afectează funcționalitatea, performanța și posibilitatea de scalabilitate și extensibilitate a aplicației.

#### 3.1.1 Limbajul de programare C#

C# (pronunțat C Sharp) este un limbaj de programare modern, obiect-orientat, care permite dezvoltatorilor software să creeze diverse aplicații sigure și robuste care rulează în .NET. C# este un limbaj asemănător cu C, C++ și Java, acesta fiind parte din familia C. Acest limbaj de programare oferă un limbaj lizibil pentru construirea logicii unei aplicații, întrucât ascunde o mare parte din complexitatea de bază a capacităților sale inerente.

În materie de administrare a memoriei, spre deosebire de strămoșii C și C++, în C# programatorul nu poate manipula direct memoria, oferind astfel un nivel mai înalt de abstractizare. C# folosește gestionarea automată a memoriei prin intermediul colectorului de gunoi(garbage collector) care preia responsabilitatea pentru alocarea și dealocarea memoriei.

În cadrul acestei aplicații, C# este folosit în toate proiectele componente ce au legătură cu logica preluării datelor din baza de date sau cu logica creării și modificării raportului testelor automate. Am ales acest limbaj de programare în soluția mea deoarece oferă o gamă largă de biblioteci și oferă posibilitatea de a dezvolta cu ușurință multe tipuri de aplicații, printre care și cele consolă și WPF(Windows Presentation Foundation), care sunt folosite în cazul nostru.

### 3.2 Limbajul de interogare SQL

SQL (de la numele englez Structured Query Language - limbaj de interogare structurat) este un limbaj de programare specific pentru manipularea datelor în sistemele de manipulare a bazelor de date relaționale (RDBMS), iar la origine este un limbaj bazat pe algebra relațională. Acesta are ca scop inserarea datelor, interogații,

actualizare și ștergere, modificarea și crearea schemelor, precum și controlul accesului la date.

SQL poate fi folosit eficient pentru preluarea informațiilor relevante despre testele automate din baza de date. Deoarece dorim să aflăm rezultatul testelor, timpul de începere și de finalizare al acestora, dar și alte informații importante, cu ajutorul SQL sunt dezvoltate în această soluție niște interogări specifice care să obțină aceste date.

### 3.3 Limbaje de marcare

#### 3.3.1 Limbajul de marcare XML

Extensible Markup Language (XML) [10] este un limbaj de marcare și un format de fișier pentru stocarea, transmiterea și reconstrucția datelor arbitrare. Acesta definește un set de reguli pentru codificarea documentelor într-un format care este atât ușor de citit de către oameni, cât și de citit de către mașini. Specificația XML 1.0 a Consortiumului World Wide Web din 1998 și mai multe alte specificații relevante - toate fiind standarde deschise gratuite - definesc XML.

Obiectivele de proiectare ale XML pun accentul pe simplitate, generalitate și utilizabilitate în întregul internet. Este un format de date textual cu suport puternic pentru diferite limbi umane prin intermediul Unicode. Deși proiectarea XML se concentrează pe documente, limbajul este utilizat pe scară largă pentru reprezentarea structurilor de date arbitrare, cum ar fi cele utilizate în serviciile web.

Scopul principal al XML este serializarea, adică stocarea, transmiterea și reconstrucția datelor arbitrare. Pentru ca două sisteme diferite să poată schimba informații, trebuie să se înțeleagă asupra unui format de fișier. XML standardizează acest proces. Este, prin urmare, analog cu o lingua franca pentru reprezentarea informațiilor.

Limbajul de marcare oferă o facilitate valoroasă pentru crearea fișierului de tip XML care facilitează comunicarea dintre aplicația consolă responsabilă cu preluarea datelor despre testele automate eșuate din baza de date și aplicația consolă responsabilă cu crearea raportului acestora.

Prin utilizarea limbajului de marcare XML, putem defini o structură bine definită și coerentă pentru fișierul de tip XML. Acest lucru permite să avem un format standardizat în care putem încapsula informațiile relevante despre testele automate eșuate, precum identificatorul testului, locația fișierelor de ieșire și de intrare, scenariul de teste din care face parte, etc.

Fișierul XML poate fi considerat un fel de limbaj comun între cele două aplicații consolă, deoarece oferă o modalitate convenabilă de a transmite și interpreta datele

relevante pentru generarea raportului. Prin respectarea structurii și semanticii XML, putem asigura că ambele aplicații pot citi și procesa corect informațiile.

Mai mult decât atât, limbajul de marcare XML permite adăugarea de attribute și elemente personalizate, permițându-ne să adăugăm informații suplimentare în fișierul XML, dacă este necesar. Aceasta oferă flexibilitate și extensibilitate în comunicarea și transmiterea datelor între cele două aplicații.

### 3.3.2 Limbajul de marcare XAML

XAML (Extensible Application Markup Language) [14] este o variantă a limbajului de marcare XML (Vezi [Subsecțiunea 3.3.1](#)) dezvoltată de către Microsoft pentru descrierea unei interfețe grafice.

În această aplicație software, prin folosirea XAML în cadrul Windows Presentation Foundation (Vezi [Subsecțiunea 3.6](#)), acest limbaj declarativ a făcut posibilă crearea și gestionarea elementelor vizuale, stilurilor și a altor aspecte ale interfeței grafice într-un mod eficient și flexibil.

## 3.4 Biblioteci

O bibliotecă reprezintă o colecție de cod predefinit și resurse asociate, care oferă funcționalități și instrumente dezvoltatorilor pentru a simplifica dezvoltarea software-ului. Bibliotecile sunt create pentru a fi reutilizabile și sunt proiectate pentru a oferi un set de funcții și componente standardizate, care pot fi utilizate în diferite proiecte software.

Bibliotecile pot conține funcții, clase, structuri de date, metode și alte resurse, care sunt scrise și testate de dezvoltatori experimentați și puse la dispoziție pentru a fi utilizate de către alți dezvoltatori. Acestea pot include funcționalități diverse, cum ar fi manipularea fișierelor, criptografia, comunicarea în rețea, interacțiunea cu baze de date, grafică, procesare de imagini și multe altele.

### 3.4.1 Tesseract

Tesseract este o bibliotecă open source de recunoaștere optică a caracterelor (OCR) dezvoltată inițial la HP Labs în jurul anilor 1980, a fost lansată ca open source în anul 2005 și ulterior continuată și menținută de Google din anul 2006. Această bibliotecă se folosește pentru a extrage text din imagini și fișiere PDF. Tesseract este scrisă în C++, dar se poate folosi cu diverse limbaje de programare, printre care se află și C#.

În cadrul acestei aplicații software, biblioteca Tesseract a fost folosită deoarece poate fi integrată cu ușurință și oferă accesarea simplă a algoritmilor de extragere a textului pentru imaginile prezente în fișierele de ieșire ale testelor automate eșuate.



### 3.4.2 OpenCvSharp

OpenCvSharp este o bibliotecă open source folosită pentru programarea computer vision în limbajul C#. Este o implementare a funcționalităților OpenCv (Open Source Computer Vision Library) pentru .NET, oferind un set bogat de capacități pentru prelucrarea imaginilor și analiza video. OpenCvSharp permite dezvoltatorilor să utilizeze algoritmi avansați de prelucrare a imaginilor, cum ar fi detectarea de obiecte, recunoașterea facială, segmentarea imaginilor, corecția culorilor, filtrarea imaginilor, extragerea de caracteristici și multe altele.

Prin intermediul OpenCvSharp, programatorii pot utiliza funcționalitățile OpenCV în proiectele lor C#, fără a fi nevoie să scrie cod în C++ sau să utilizeze wrapper-e complexe.

În această aplicație C#, am integrat biblioteca OpenCvSharp pentru a ne permite să realizăm un proces de prelucrare a imaginilor eficient. Cu ajutorul acestei biblioteci, am aplicat pentru imaginile din fișierele de ieșire ale testelor automate eșuate un filtru Gauss pentru a atenua zgomotul și pentru a obține o imagine cu margini mai clare. Apoi, am aplicat algoritmul Canny pentru a detecta și evidenția contururile casuțelor de eroare în imagine. Rezultatul final este o imagine procesată care pune în evidență clar zonele de interes, facilitând identificarea și analiza casuțelor de eroare într-un mod vizual plăcut și intuitiv. Această procesare ajută foarte mult la obținerea unor rezultate cât mai exacte și relevante în urma folosirii OCR-ului Tesseract pe imaginile respective.

### 3.4.3 Interop.Excel

Interop.Excel este o bibliotecă de interfețe de programare a aplicațiilor (API) care permite interacțiunea cu Microsoft Excel în cadrul limbajului de programare C#. Această bibliotecă face parte din pachetul Microsoft Office și este folosită pentru a manipula și automatiza operațiuni în fișierele Excel, cum ar fi crearea, citirea și modificarea datelor, formatarea celulelor, adăugarea de diagrame și grafice, executarea de calcule și multe altele.

### 3.4.4 Kernel32

Kernel32.dll este o bibliotecă de legături dinamică (DLL) în sistemul de operare Windows care conține funcții esențiale pentru interacțiunea între aplicații și nucleul (kernelul) sistemului de operare. Această bibliotecă este parte a nucleului Windows și oferă funcționalități de bază pentru gestionarea proceselor, memoriei, fișierelor și altele.

Kernel32 a fost folosit în această aplicație întrucât acesta oferă funcții pentru manipularea fișierelor și directoarelor: Aceste funcții permit deschiderea, închiderea, citirea, scrierea și manipularea fișierelor și directoarelor. Așadar, funcțiile precizate

anterior sunt necesare pentru a transmite datele din fișierul de inițializare cu extensia „.ini” specific aplicațiilor consolă la acestea. Mai mult de atât, aplicațiile consolă pot și ele modifica conținutul fișierului de inițializare.

Pentru modificarea și preluarea datelor din fișierul .ini s-au folosit wrappere peste următoarele două funcții:

- **[DllImport("kernel32")]**  
**private static extern long WritePrivateProfileString(string name, string key, string val, string filePath);**  
unde **name** este numele secțiunii, **key** este o cheie prezentă în secțiunea aleasă, căreia i se va atașa valoarea prezentă în **val**. Parametrul **filePath** reprezintă calea către fișierul cu extensia ini.
- **[DllImport("kernel32")]**  
**private static extern int GetPrivateProfileString(string section, string key, string def, StringBuilder retVal, int size, string filePath);**  
unde **section** este numele secțiunii, **key** este o cheie prezentă în această secțiune, **def** este un string implicit (default), **retVal** reprezintă un buffer care conține valoarea returnată, **size** este numărul de caractere regăsite în **retVal**, iar **filePath** este calea către fișierul cu extensia ini. În cazul în care cheia specificată nu poate fi găsită, **retVal** va lua valoarea **def**.

### 3.4.5 ADO.NET

ADO.NET (ActiveX Data Objects .NET) [11] este o componentă cheie a platformei .NET pentru accesarea și manipularea datelor în aplicații. ADO.NET oferă un set de clase și funcționalități care permit dezvoltatorilor să interacționeze cu diverse surse de date, printre care și baze de date relaționale. Această componentă a platformei .NET separă accesul la date de manipularea lor prin componente discrete care pot fi folosite pe cont propriu sau împreună. ADO.NET include furnizori de date .NET Framework pentru conectarea la o bază de date, executarea comenzilor și preluarea rezultatelor. Exemple de furnizori de date .NET Framework folosiți în ADO.NET sunt următoarele:

- **Connection** – oferă conectivitate la o sursă de date
- **Command** – oferă accesul la comenzi asupra bazei de date pentru a returna și modifica date, pentru a rula proceduri stocate, sau pentru a trimite și prelua parametrii.
- **DataReader** - oferă un flux de date de înaltă performanță de la sursa de date.

Un alt instrument important al ADO.NET este **SqlParameter**. Acesta ajută la prevenirea atacurilor de tip SQL injection în aplicațiile care utilizează ADO.NET pentru a interacționa cu bazele de date SQL Server. SQL injection este o vulnerabilitate de securitate în care un atacator încearcă să insereze sau să execute cod SQL rău intenționat în interogările sau comenzile SQL ale unei aplicații.

Pentru crearea acestei soluții software, toate cele patru instrumente menționate anterior au fost folosit împreună pentru a prelua date din baza de date AutomatedTests din SQL Server.

### 3.4.6 MahApps.Metro

MahApps.Metro [15] este o bibliotecă open-source pentru platforma Windows Presentation Foundation (WPF) care oferă un set de stiluri, șabloane și controale personalizate pentru a crea interfețe grafice moderne și atractive în aplicațiile desktop.

MahApps.Metro simplifică dezvoltarea aplicațiilor WPF, oferind o colecție extinsă de stiluri predefinite, care imită aspectul și comportamentul interfețelor utilizate în aplicațiile moderne de Windows, cum ar fi cele din Windows 10. Aceasta include stiluri pentru ferestre, butoane, bara de titlu, bara de meniu, tab-uri, casete de dialog, liste, controale de introducere a textului și multe altele.

## 3.5 NuGet Package Manager

Un administrator de pachete(package manager) este un instrument care ajută dezvoltatorii software să administreze dependențele și bibliotecile din cadrul proiectelor lor. Acesta automatizează procesul de descărcare, instalare și actualizare a bibliotecilor sau pachetelor care sunt necesare pentru un anumit proiect software.

NuGet este un administrator de pachete special conceput pentru ecosistemul Microsoft și este folosit în special pentru a simplifica administrarea de pachete și biblioteci în aplicațiile .NET. Acest administrator de pachete poate fi utilizat atât din linia de comandă cât și din interfața grafică din cadrul mediului de dezvoltare Visual Studio(Figura 3.1).

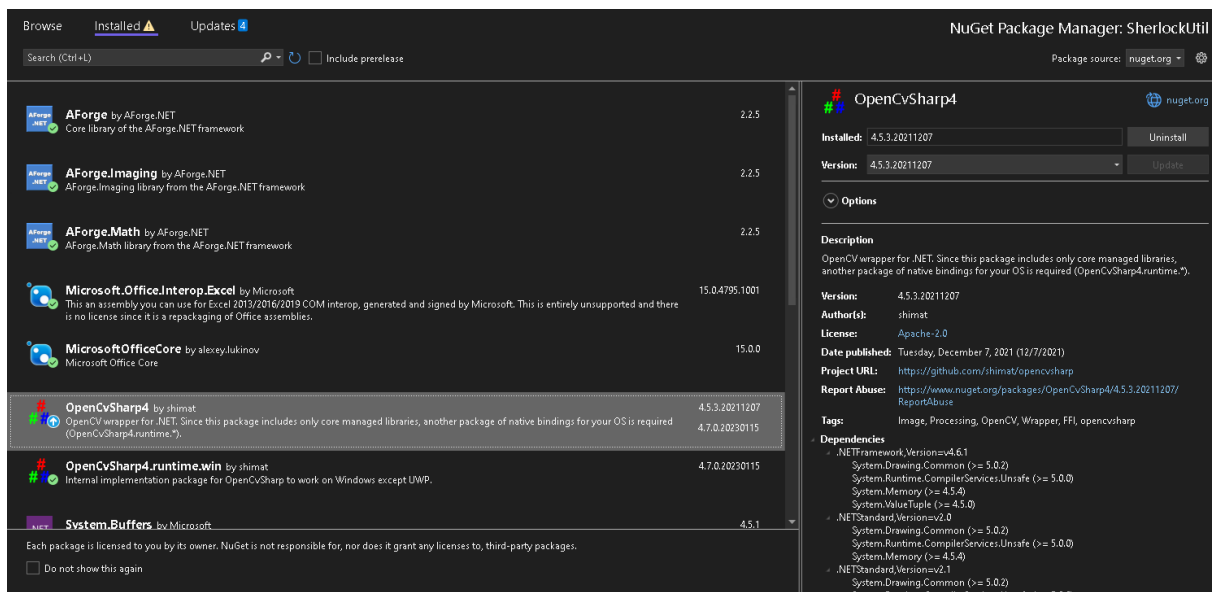


Figura 3.1: Interfața grafică a administratorului de pachete NuGet

În această soluție software, NuGet a facilitat descărcarea, instalarea și adăugarea directă și ușoară a majorității bibliotecilor menționate anterior, Tesseract, OpenCvSharp și Interop.Excel.

### 3.6 WPF

Windows Presentation Foundation [9] este un framework de dezvoltare software creat de Microsoft, care permite crearea și afișarea interfețelor grafice pentru aplicații Windows. Nucleul WPF este un reprezentat de un motor de randare independent de rezoluție și bazat pe vector. Acest nucleu este extins cu ajutorul conceptului de XAML (Extensible Application Markup Language), un limbaj de marcare care permite definirea interfeței grafice a aplicației într-un mod declarativ. Acesta separă logica aplicației de aspectul și stilul interfeței grafice, ceea ce facilitează dezvoltarea și mentenanța aplicațiilor.

Prin utilizarea WPF, dezvoltatorii pot crea interfețe grafice moderne și interactive, cu suport pentru elemente grafice 2D și 3D, animații, transparentă, gradient, text stilizat și multe altele. WPF oferă, de asemenea, suport pentru legarea datelor (data binding), adică conectarea datelor aplicației la elementele interfeței grafice, ceea ce permite actualizarea automată a interfeței grafice pe măsură ce se schimbă datele.

Windows Presentation Foundation oferă o gamă largă de elemente de control pentru crearea interfețelor grafice în aplicațiile desktop Windows. Aceste elemente de

control sunt elementele vizuale cu care utilizatorii interacționează și care permit afișarea și manipularea datelor. Printre tipurile de elemente de control din WPF utilizate în această aplicație software se regăsesc următoarele:

1. **Elemente de control de bază:** WPF oferă o serie de elemente de control de bază, cum ar fi Button (buton), TextBox (căsuță de text), Label (etichetă), CheckBox (căsuță de bifare), RadioButton (buton radio), ListBox (listă), ComboBox (caseta de selectare), etc. Acestea permit interacțiunea utilizatorului și introducerea/daterea datelor.
2. **Elemente de control de afișare a datelor:** WPF include o serie de elemente de control specializate pentru afișarea datelor, cum ar fi DataGrid (tabel de date), ListView (vizualizare listă), TreeView (vizualizare arbore), și altele. Aceste controluri facilitează afișarea și manipularea datelor tabulare sau ierarhice.

Fiecare element de control din Windows Presentation Foundation are proprietăți, evenimente și funcționalități specifice, care permit personalizarea și adaptarea interfeței grafice la nevoile aplicației.

### 3.7 SQL Server

SQL Server [12] este un sistem de gestionare a bazelor de date relaționale dezvoltat și promovat de către Microsoft. SQL Server este construit pe baza SQL([subsecțiunea 3.2](#)), care este un limbaj folosit pentru interacțiunea cu bazele de date relaționale. SQL Server folosește Transact-SQL, implementarea Microsoft a SQL care adaugă peste acesta un set de instrucțiuni specifice care se pot folosi pentru a îndeplini o varietate de sarcini.

Componenta de bază a SQL Server este engine-ul bazei de date. Engine-ul bazei de date constă într-un engine relațional care procesează interogări și un engine de stocare care administrează fișierele bazei de date, paginile, indexele etc.

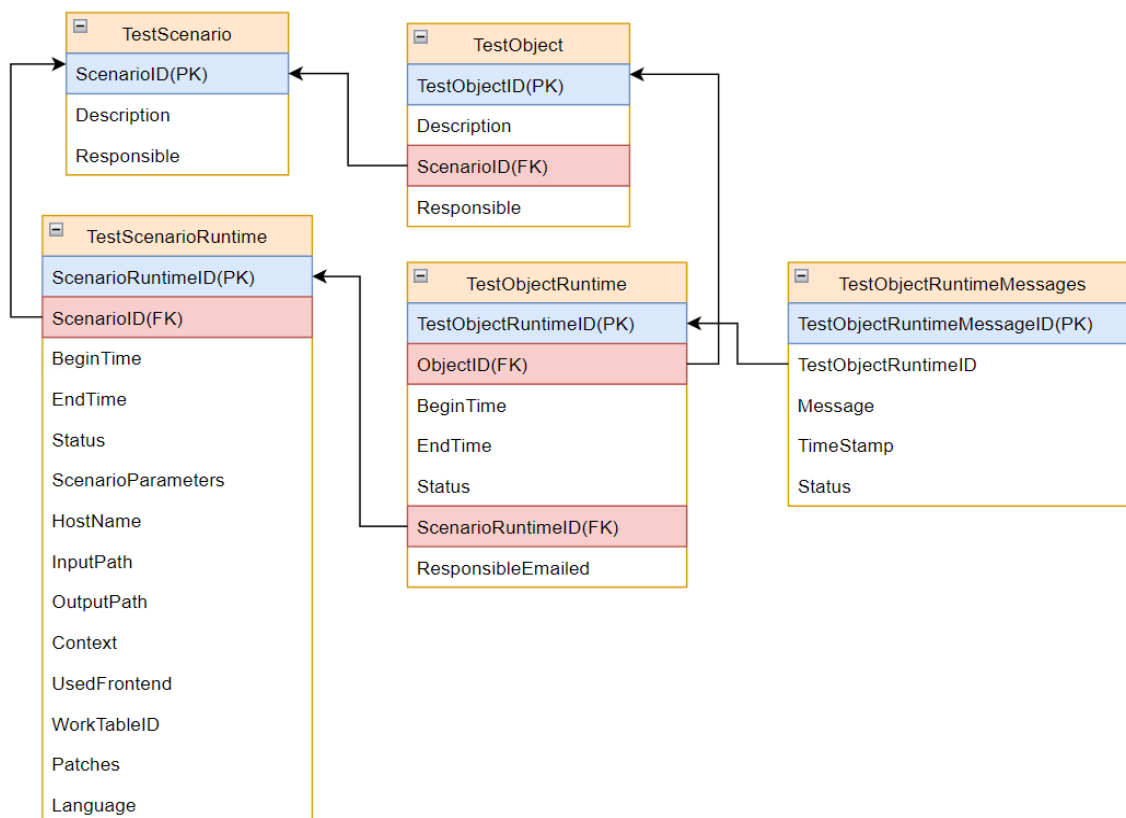


Figura 3.2: Diagrama UML a bazei de date AutomatedTesting

În SQL Server a fost creată și baza de date la care se conectează aplicația, **AutomatedTesting** (Figura 3.2). Baza de date este una relațională întrucât aceasta conține următoarele caracteristici principale ale bazelor de date relaționale:

- **Tabele:** Datele sunt organizate în tabele, care sunt structuri bidimensionale formate din rânduri și coloane. Fiecare tabelă reprezintă o entitate sau un tip de obiect în cadrul sistemului.
- **Coloane și tipuri de date:** Coloanele reprezintă atributele sau caracteristicile unei entități și definesc tipurile de date ale datelor stocate în tabelă, cum ar fi întregi, șiruri de caractere, valori de tip boolean etc.
- **Rânduri și înregistrări:** Rândurile reprezintă înregistrările sau instanțele entităților și conțin date specifice pentru fiecare atribut al tablei.
- **Chei primare și chei străine:** Cheia primară este o coloană sau un set de coloane care identifică în mod unic fiecare înregistrare dintr-o tabelă. Cheile străine sunt utilizate pentru a stabili relații între diferite tabele, bazându-se pe valori comune ale cheilor primare și cheilor străine.
- **Relații:** Relațiile sunt conexiuni logice între tabele, care permit interogarea și accesarea datelor din mai multe tabele într-un mod structurat și coerent.

- **Integritate referențială:** Bazele de date relaționale aplică reguli de integritate referențială pentru a asigura coerența datelor și a menține integritatea relațiilor dintre tabele.

Baza de date relațională **AutomatedTesting** realizată în SQL Server este o componentă foarte importantă a acestei soluții pentru că în ea se află toate informațiile de care aplicația are nevoie pentru a începe analiza testelor automate eşuate.

În contextul realizării aplicației, cele mai de interes elemente din baza de date sunt **TestObject** și **TestObjectRuntime**. La fiecare rulare a unui **TestObject**, în baza de date va apărea o nouă înregistrare în tabelul **TestObjectRuntime** care va reține rezultatul acestei acțiuni, dar și alte informații relevante. Aceste două tabele au o relație de tip one-to-many, un **TestObject** având mai multe **TestObjectRuntime** asociate și sunt legate între ele prin cheia străină **ObjectID**.

Tabelul **TestObjectRuntimeMessages** conține mai multe mesaje apărute în timpul executării testelor automate și este legat de tabelul **TestObjectRuntime** prin cheia străină **TestObjectRuntimeID**. Aceste două tabele au o relație de agregare întrucât fiecare **TestObjectRuntime** poate avea mai multe înregistrări de tipul **TestObjectRuntimeMessages** care să îi corespundă.

În contextul acestei soluții software, mai multe **TestObject**-uri sunt grupate într-un singur **TestScenario**. Pentru a executa un anumit **TestObject**, va trebui să fie executat întregul **TestScenario**. Aceste două tabele conțin și ele o relație de tip one-to-many, un scenariu conținând mai multe obiecte de test.

**TestScenarioRuntime** conține cele mai multe informații din baza de date, aici fiind prezente date despre execuția scenariilor cum ar fi statusul scenariului (doar dacă toate testele componente scenariului sunt rulate cu succes acesta va fi considerat reușit), frontend-ul folosit, mașina de lucru pe care s-a rulat scenariul, dar și locația fișierelor de intrare și de ieșire corespunzătoare testelor componente. Între tabelul **TestScenario** și **TestScenarioRuntime** există o relație one-to-many. De altfel, și între **TestScenarioRuntime** și **TestObjectRuntime** există o relație one-to-many realizată prin cheia străină **ScenarioRuntimeID**.

## 3.8 LINQ

LINQ (Language Integrated Query) [13] este un set puternic de tehnologii bazat pe integrarea capacităților de interogare direct în limbajul C#. Interogările LINQ sunt construcții de limbaj de primă clasă în C# .NET, la fel ca clasele, metodele și evenimentele. LINQ oferă o experiență coerentă de interogare pentru obiecte (LINQ to Objects), baze de date relaționale (LINQ to SQL) și XML (LINQ to XML).

Una dintre caracteristicile cheie ale LINQ este utilizarea metodelor de extensie definite în namespace `System.Linq`, metode care permit operații de interogare și manipulare asupra colecțiilor din C# care implementează interfața `IEnumerable`. Câteva exemple de astfel de metode sunt `Where`, `Select`, `FirstOrDefault`, `First`, `Any`, etc.

Metodele de extensie ale LINQ au fost utilizate în această aplicație pentru a facilita manipularea cât mai simplă a colecțiilor folosite, la fel ca în următorul exemplu:

- **`var enabledAtGroups = ATGroupList.Where(e => e.Enabled);`**  
unde **`ATGroupList`** este o listă de grupuri de teste automate. Rezultatul obținut în interiorul variabilei **`enabledAtGroups`** va fi o sublistă a lui `ATGroup` în care toate elementele componente vor avea în proprietatea `Enabled` setată valoarea „true”.



## 4 Implementare

Pentru a reduce timpul și efortul necesare în analizarea testelor eșuate și pentru a genera rapoarte mai estetice, aplicația propusă va implementa o funcționalitate de analiză automată a testelor eșuate, cu generarea ulterioară a unui raport intuitiv. Aceasta va permite utilizatorilor să identifice rapid și ușor erorile și să înțeleagă mai bine rezultatele testelor.

În acest capitol vor fi prezentate noțiuni care țin de implementarea propriu-zisă a aplicației software, cum ar fi arhitectura ei și detalii tehnice amănunțite despre cum au fost realizate funcționalitățile din cadrul acestei soluții.

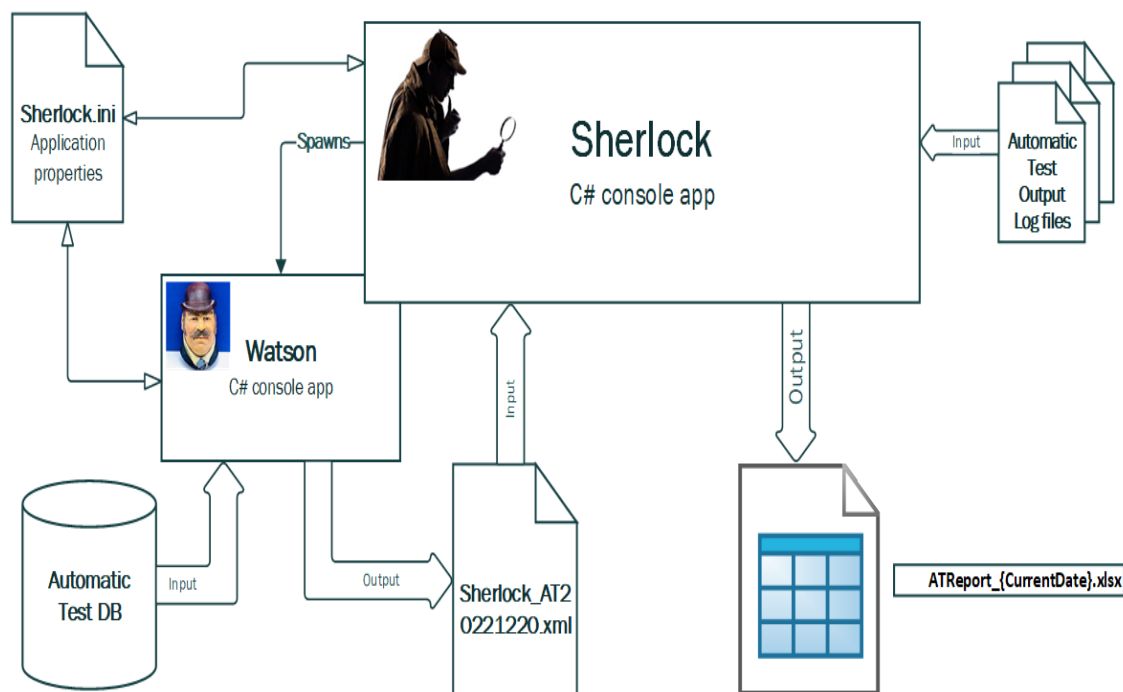


Figura 4.1: Flow-ul aplicației

În **Figura 4.1** se poate observa parcursul aplicației, dar și unele dintre componentele acesteia, precum:

- Aplicația consolă **Sherlock** responsabilă pentru generarea raportului din fișierul Excel

- Aplicația consolă **Watson** responsabilă pentru extragerea informațiilor din baza de date **AutomatedTesting** pentru a face posibilă începerea analizei lui **Sherlock**.
- Fișierul de inițializare **Sherlock.ini**, de unde cele două aplicații consolă extrag informații despre ce își dorește utilizatorul să apară în raport.
- Fișierul **XML** rezultat în urma execuției lui **Watson**, unde sunt prezente informații despre testele automate eșuate, și care va fi folosit mai departe de **Sherlock** pentru a finaliza procesul.

Execuția pornește de la proiectul **Sherlock/SherlockUI**, care preia informațiile din fișierul de inițializare **Sherlock.ini** și îl inițiază pe **Watson**, trimițându-i ca argument locația în care va trebui creat fișierul XML. În continuare, aplicația consolă **Watson** verifică ce grupuri de teste automate au fost introduse de utilizator în fișierul **Sherlock.ini** și pe baza acestora preia informații din baza de date și le împachetează în fișierul XML care va servi mai departe drept input pentru **Sherlock**. După ce **Watson** și-a îndeplinit scopul și a introdus informațiile relevante din baza de date în fișierul XML, **Sherlock** începe analiza testelor automate eșuate și le verifică fișierele de ieșire ale acestora. În final, după ce **Sherlock** termină de analizat testele, acesta creează raportul și introduce în el informații despre teste, incluzând numele, cine este responsabil pentru ele, statusul lor, dar și mesajele de eroare care au apărut în timpul rulării lor.

Rezultatul final al procesării este un raport care conține toate informațiile relevante despre testele automate eșuate, prezentate într-un format plăcut, care permite utilizatorului să vizualizeze și să modifice toate informațiile dintr-un singur loc.

## 4.1 Aplicația componentă Watson

**Watson** reprezintă o aplicație consolă folosită pentru a comunica cu baza de date **AutomatedTesting** în care sunt logate informații despre testele automate cum ar fi lansările, versiunile, timpul la care a început și la care s-a finalizat rularea acestora, eticheta de generare, etc. Rolul acestei aplicații este să colecteze aceste informații din baza de date cu ajutorul mai multor query-uri SQL, ca mai departe să le proceseze și să le integreze într-un fișier de tip XML pentru a facilita execuția următorilor pași necesari pentru crearea raportului testelor eșuate. Acesta a fost implementat în limbajul de programare C# și se folosește de un fișier de inițializare numit **Sherlock.ini** de unde ia informații despre grupurile de teste pe care utilizatorul curent își dorește să le găsească în raportul generat.

**Watson** prezintă următoarele clase:

1. **ATGroup**
2. **WatsonMain**

Aceste două clase au o relație de asociere strânsă, întrucât clasa **ATGroup** conține informații aferente testelor automate și clasa **WatsonMain** folosește o listă de

obiecte de tipul **ATGroup** pe care o iterează pentru a afla cu exactitate care sunt datele de care utilizatorul este interesat.

#### 4.1.1 Clasa ATGroup

Testele automate au mai multe atribute, cum ar fi versiuni, platforme, configurații sau limbaje și din acest motiv ele au fost grupate în funcție de parametrii lor în mai multe **ATGroup**-uri (Automated Tests Group). Atributele respective sunt preluate din fișierul Sherlock.ini, unde sunt prezente atât informații despre grupuri dar și instrucțiuni pentru analiza sintactică pe care urmează să o facă **Sherlock**. Un exemplu simplificat despre cum arată Sherlock.ini:

- [GlobalSettings]
- Releases=22A,23A
- ; All supported releases e.g Releases=23A, 22A, ...
- Platforms=64 Bit, 32 Bit
- ; All supported platforms
- Configurations=Debug,DebugRelease, \_Setup
- ; All supported configurations
- Languages=EN,SYM
- ;All supported languages e.g. Languages = EN, SYM
- Responsibles=msojhp
- ; List of AT responsables
- ATGroups=23Debug64EN,23Debug64SYM
- ; Names of the AT groups for which a report will be generated. Each name corresponds with a separate section in the ini file.
- ; For each item listed in the ATGroups setting in the GlobalSettings section, a separate section with the same name has to be made.
- WatsonOutputPath=xmlOutput.xml
- ; The path to the location where the Watson XML output file will be saved.
- EnableTextExtraction=1
- ; 0 = Image text will NOT be extracted, 1 = Image text will be extracted
- [23Debug64EN]
- Release=23A
- Platform=64 Bit
- Configuration=Debug
- Language=EN
- Responsible=msojhp
- LastBuildReported=230228203730\_23A.230228\_2022.23\_debug\_64\_en
- ; Name of the build that is the last one for which a report was generated

- LastBuildPendingTests=
- ; The number of pending tests that were still not executed when the last report was made for it.
- [23Debug64SYM]
- Release=23A
- Platform=64 Bit
- Configuration=Debug
- Language=SYM
- Responsible=msojhp
- LastBuildReported=230228203730\_23A.230228\_2022.23\_debug\_64\_en
- LastBuildPendingTests=
  
- [ATOutputFiles]
- ; The list of test output files to examine. Each item will have the following format:
- Output=ATFSystem\output.txt
- RuntimeMessages=runtimeMessages.txt
- LogError=ATFSystem\error.txt
- StdError=ATFSystem\stderr\_\*.txt
- ErrorPicture=ATFSystem\1.jpg
- ; For each item in ATOutputFiles an associated section must be made

Structura clasei **ATGroup** este ilustrată în Figura 4.2.

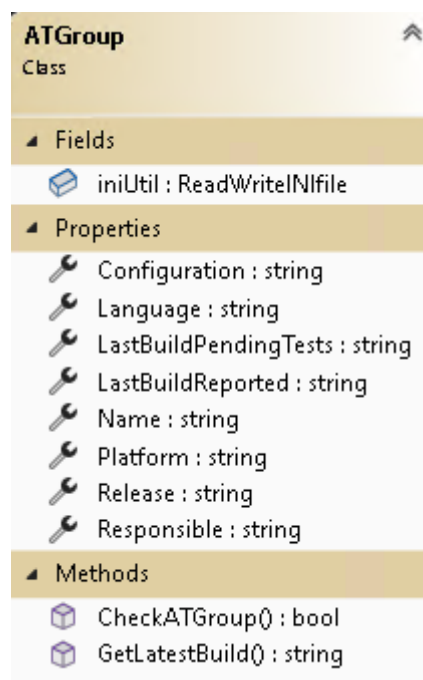


Figura 4.2: Diagrama UML a clasei ATGroup

După cum se poate observa din diagrama UML prezentă în [Figura 4.2](#), clasa conține două funcții:

Funcția **CheckATGroup** verifică dacă datele introduse în fișierul .ini de către utilizator sunt corecte, caz în care execuția merge mai departe, sau în cazul contrar afișează în consolă mesaje de eroare și oprește execuția programului.

Clasa conține și o legătură cu baza de date, în funcția **GetLatestBuild** se construiește un query folosind proprietățile prezente în **ATGroup** și se obține cea mai recentă etichetă de generare pentru grupul respectiv.

#### 4.1.2 Clasa WatsonMain

În clasa **WatsonMain** se află logica colectării tuturor informațiilor despre testele eșuate pe eticheta de generare curentă din baza de date, cum ar fi timpul și data în care au fost rulate, dispozitivele pe care au fost executate, numele, ID-ul lor și în unele cazuri chiar și motivul eșuării.

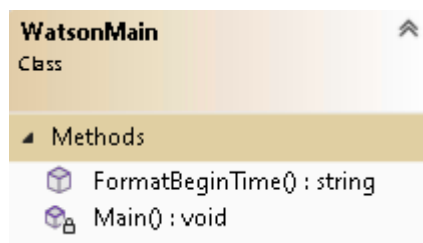


Figura 4.3: Diagrama UML a clasei WatsonMain

În [Figura 4.3](#) se poate observa faptul că această clasă conține două funcții, una dintre ele este **FormatBeginTime**, funcție care prelucrează un atribut din baza de date pentru a coincide cu locațiile testelor automate de pe server. Funcția **Main** este locul unde se face legătura cu baza de date **AutomatedTesting**, se preiau informațiile despre testele eșuate din ea care sunt mai departe prelucrate și introduse într-un fișier XML creat de la zero, fișier necesar mai departe lui Sherlock sau SherlockUI pentru a rula cu succes, întrucât acesta conține mai multe atribute referitoare la locația fișierelor lor de output de pe server.

## 4.2 Biblioteca SherlockUtil

**SherlockUtil** este o bibliotecă dinamică(DLL) de clase scrisă integral în limbajul de programare C#, care conține majoritatea logicii din spatele creării raportului. Aici se găsesc funcționalități pentru preluarea datelor scrise de Watson în fișierul XML și

algoritmi folosiți la extragerea datelor din fișierele de output ale testelor într-un mod cât mai relevant posibil. Pe lângă acestea, componenta mai conține funcționalități pentru crearea, compararea și salvarea raportului, cât și posibilitatea de a reprograma unul sau mai multe teste eșuate, în funcție de ce date au fost introduse de utilizator în fișierul Excel după ce raportul a fost creat.

Scopul bibliotecii **SherlockUtil** este de a oferi un nivel de abstractizare și modularitate, astfel încât funcționalitățile comune și reutilizabile să poată fi separate de logica specifică a proiectelor **Sherlock** și **SherlockUI**. Aceasta promovează o mai bună organizare a codului și favorizează o dezvoltare mai eficientă și mai ușoară.

Fiind o bibliotecă dinamică, **SherlockUtil** prezintă următoarele avantaje:

1. **Modularitate și reutilizare:** Bibliotecile dinamice permit împărțirea funcționalității în module independente, care pot fi utilizate de mai multe aplicații. Aceasta duce la o mai mare reutilizare a codului și la o organizare mai clară a funcționalității. Modulele pot fi actualizate sau înlocuite separat, fără a afecta întreaga aplicație.
2. **Economisirea resurselor:** Deoarece bibliotecile dinamice sunt încărcate la cerere în memoria aplicației, acest lucru poate duce la o utilizare mai eficientă a resurselor. Doar modulele necesare sunt încărcate și utilizate în funcție de nevoile aplicației. Acest lucru poate duce la o încărcare mai rapidă a aplicației și la o utilizare mai eficientă a memoriei.
3. **Actualizări și corecții ușoare:** Bibliotecile dinamice pot fi actualizate sau înlocuite independent, fără a fi necesară recompilarea sau redistribuirea întregii aplicații. Acest lucru facilitează implementarea de actualizări, corecții sau îmbunătățiri ale bibliotecii fără a afecta funcționalitatea aplicației.
4. **Flexibilitate și extensibilitate:** Utilizarea bibliotecilor dinamice permite încărcarea și utilizarea de funcționalități suplimentare sau extensii la cerere. Aceasta oferă o modalitate eficientă de extindere a funcționalității aplicației fără a modifica codul existent sau a necesita recompilarea.

Biblioteca **SherlockUtil** conține o multitudine de clase:

1. ATReport
2. ErrorRule
3. FailedTests
4. IMessage
5. OutputFile
6. ReadWriteIniFile

## 7. TestObject

Dintre aceste clase, **ATReport**, **ErrorRule**, **OutputFile** și **TestObject**, sunt esențiale în procesul de generare a rapoartelor, deoarece ele conțin informațiile necesare pentru construirea rapoartelor. Prin intermediul acestor clase, se formează o structură complexă care reflectă detaliile relevante despre erori și rezultatele testelor automate. Acestea sunt interconectate pentru a aduna și organiza informațiile necesare pentru a genera rapoartele finale. **TestObject** reprezintă un obiect central în procesul de generare, conținând o listă de obiecte **OutputFile** în care se prezintă informații despre fișierele de ieșire rezultate în urma executării testelor automate. În interiorul fiecărui **OutputFile**, găsim o listă de obiecte **ErrorRule**, care stochează informații specifice despre erori și reguli asociate. **ATReport** este responsabilă de instanțierea obiectelor **TestObject** pentru fiecare test automat eșuat identificat în fișierul XML.

Restul claselor, **IMessage**, **ReadWriteIniFile** și **FailedTests** sunt clase utilitare în proiect. Acestea ajută clasa **ATReport** să afișeze mesaje personalizate, să citească fișierul de inițializare și respectiv să preia informațiile din fișierul XML.

### 4.2.1 Clasa ATReport

**ATReport** este cea mai importantă clasă a bibliotecii, aceasta folosindu-se în interiorul ei de toate celelalte clase existente în cadrul proiectului. În această clasă se creează propriu zis aplicația și workbook-ul Excel folosind pachetul NuGet Microsoft.Office.Interop.Excel, acestea urmând să fie prelucrate în interiorul funcțiilor componente.

**ATReport** este instanțiată de **Sherlock** și **SherlockUI** pe parcursul execuției lor și are rolul de a ține cont de absolut tot ceea ce se întâmplă de când s-a terminat execuția **Watson** și s-a produs fișierul XML de intrare și până când sunt parsate informațiile despre testele automate în fișierul Excel.

Procesul de scriere a fișierului Excel din interiorul clasei constă în preluarea datelor din fișierul XML de intrare, urmată de iterarea testelor automate eșuate într-o manieră multithreaded pentru a găsi exact mesajul de eroare apărut în interiorul fișierelor output ale fiecărui test. În urma acestor acțiuni, în cazul în care utilizatorul a bifat opțiunea de extragere a textului din imagini din fișierul **Sherlock.ini**, algoritmul caută căsuțe de eroare în imaginile prezente în fișierele de ieșire rezultate în urma executării testelor automate și extrage textul din acestea cu ajutorul bibliotecilor **OpenCvSharp** și **Tesseract**, urmând să fie și ele adăugate în tabelul Excel în rândul corespunzător testului. În cele din urmă, după ce se adaugă în tabel toate informațiile aferente testelor automate eșuate, acesta este comparat cu raportul precedent produs de aplicație(dacă acesta exista) pentru a prelua datele introduse de utilizator în cazul în

care acestea există și testul este încă prezent la următoarea generare a raportului. După ce toate sarcinile precedente au fost îndeplinite, fișierul se salvează și raportul este pregătit să fie verificat de către membrul echipei care este responsabil pentru teste în săptămâna curentă.

Structura clasei **ATReport** este dată de diagrama UML prezentată în [Figura 4.4](#).

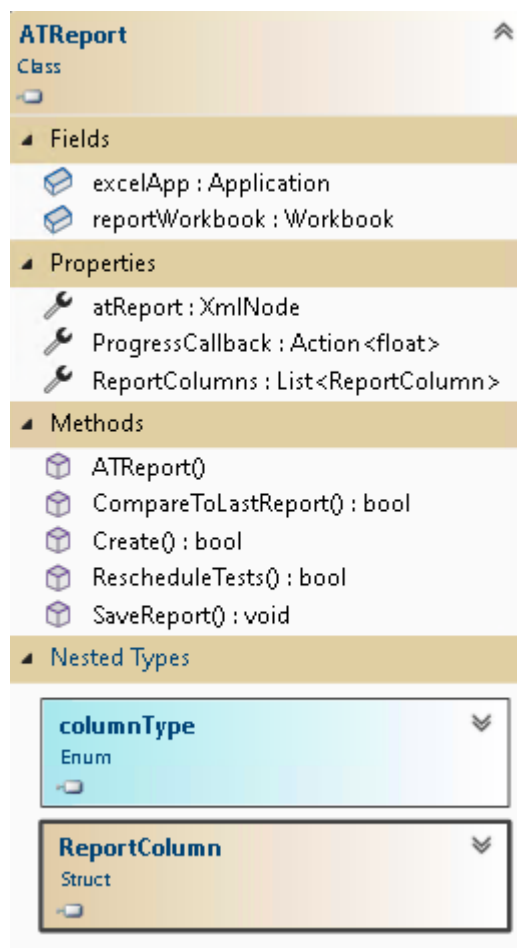


Figura 4.4: Diagrama UML a clasei ATReport

După cum am menționat mai sus, **ATReport** reține date despre coloanele din tabelul Excel și conținutul lor, motiv pentru care s-au folosit structura **ReportColumn**, unde sunt păstrate numele și alte proprietăți ale coloanelor și enum-ul **ColumnType** care ajută la introducerea și modificarea cât mai simplă a datelor din workbook, acestea urmând să fie folosite în funcția **CreateSheet** pentru a formata aspectul paginii. Enum-ul **ColumnType** conține valorile `eCategory`, `eTest`, `eErrorMessage`, `eInvestigator`, `eStatus`, `eComments`, `eResponsible`, `eInput`, `eIMG`, `eImageText`, `eMOV`, fiecare dintre aceste valori reprezentând un nume al unei coloane existente în fișa de lucru Excel creată.



### 4.2.2 Clasa ErrorRule

**ErrorRule** este clasa care conține logica folosită pentru parsarea erorilor relevante din fișierele text de output ale testelor automate eșuate. Aceasta se folosește de expresii regulate și așa numite „reguli” de parsare preluate din fișierul de inițializare **Sherlock.ini** pentru a identifica și a extrage erorile.

Regulile preluate sunt împărțite în cuvinte cheie și acțiuni specifice care trebuie luate la întâlnirea cuvintelor cheie pentru ca într-un final să se poată extrage după caz exact linia sau liniile unde se loghează erorile.

Astfel, clasa **ErrorRule** este reprezentată de diagrama UML prezentă în **Figura 4.5**.

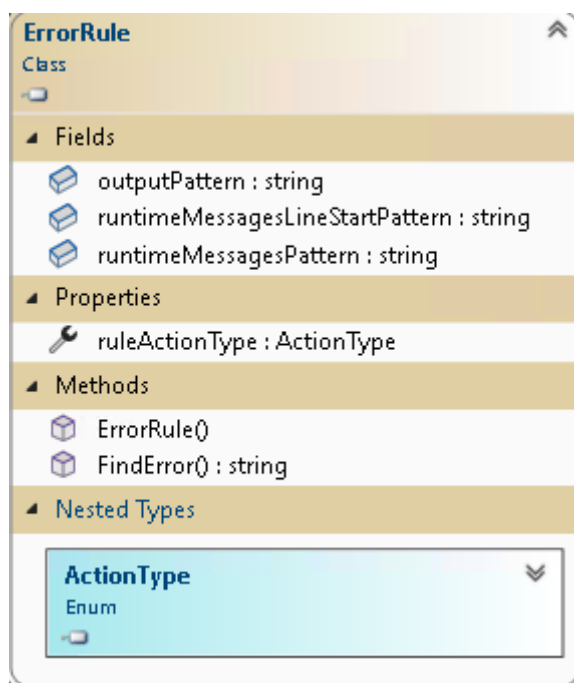


Figura 4.5: Diagrama UML a clasei ErrorRule

### 4.2.3 Clasa FailedTests

Scopul clasei **FailedTests** este acela de a aduna datele despre teste și de a le aduce în contact cu restul funcționalităților. Această clasă conține o singură funcție statică numită **Retrieve**. Această funcție statică este apelată în interiorul codului sursă din clasa **ATReport**. Funcția **Retrieve** invocă aplicația consolă **Watson** și o așteaptă pe

aceasta să termine de creat fișierul XML care va fi apoi încărcat în variabila de tip XmlNode **atReport** prezentă în interiorul clasei **ATReport**.

#### 4.2.4 Interfața IMessage

**Interfața IMessage** este folosită pentru a facilita transmiterea mesajelor de interes către utilizator sub diferite forme. **Sherlock** și **SherlockUI** au ambele implementări proprii ale interfeței **IMessage**, și anume **ConsoleMessage** și **LogMessage**, care vor fi amănunțite mai jos.

Diagrama interfeței **IMessage** este afișată în **Figura 4.6**.

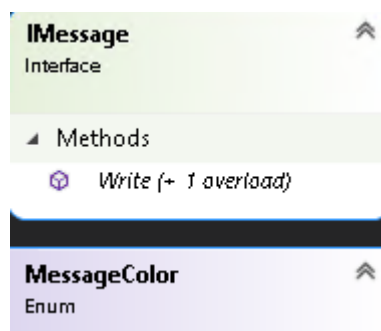


Figura 4.6: Diagrama UML a interfeței IMessage

#### 4.2.5 Clasa OutputFile

După cum îi sugerează și numele, clasa **OutputFile** conține detalii despre fișierele de output, care în cazul nostru au același nume pentru fiecare test, dar sunt aflate în locații diferite. Informațiile despre aceste fișiere, cum ar fi numele lor complet dar și regulile de parsare specifice lor se găsesc în fișierul de inițializare Sherlock.ini:

- [ATOutputFiles]
- ; The list of test output files to examine. Each item will have following format:
- Output=ATFSystem\output.txt
- RuntimeMessages=runtimeMessages.txt
- LogError=ATFSystem\error.txt
- StdError=ATFSystem\stderr\_\*.txt
- ErrorPicture=ATFSystem\1.jpg
- ; For each item in ATOutputFiles an associated section must be made.

- ; It will contain as items the keywords that have to be searched for in this file, together with the action that should be taken on it.
- ; So the syntax is:
- ; {Keyword}={Action}
- ; With
- ; Keyword = a keyword to look for in this file e.g. Error,Exception. It can also consist of more than one word.
- ; Action = a number corresponding with an action. Currently following actions are defined:
- ; 1 = Extract the text part of the log file where the keyword was found, starting from the preceding datetime tag and until the next datetime tag encountered.
- ; 2 = Extract only the line in which the keyword was found
- ; 3 = Take entire content of the file
- ; 4 = In case this keyword is present and no other known anomaly was encountered during the parsing process,
- ; it means the anomaly is not a known one yet and slips through the mazes of the keywords web.
- ; In this case, the message that must be composed for the output is "Unidentified error occurred.
- ; 5 = Extract all the lines where the keyword was found
- ; Default action is 1.

#### • [Output]

- Rule1=Differences found,1
- Rule2=ERROR[1],RESULT[0],1
- Rule3=Exception[1],at[2],5
- Rule4=Nothing to compare,2
- Rule5=Failed,4
- ;[0] = Does not contain keyword
- ;[1] = contains keyword
- ;[2] = optional(for further parsing)

#### • [RuntimeMessages]

- Rule1=Differences found,1
- Rule2=ERROR[1],RESULT[0],1
- Rule3=Exception[1],at[2],5
- Rule4=Nothing to compare,2
- Rule5=Failed,4

#### • [LogError]

- Rule1=All,3

#### • [StdError]

- Rule1=All,3

În cadrul acestui studiu, au fost testate mai multe seturi de reguli și s-a ajuns la concluzia că în general, la momentul de față, pentru a obține cele mai relevante rezultate trebuie folosite regulile prezentate la descrierea fișierului **Sherlock.ini**. Așadar, aceste reguli din fișierul de inițializare nu ar să fie schimbate de utilizator, întrucât această acțiune poate duce la obținerea unor rezultate mai puțin relevante.

Obiectele de tip **OutputFile** conțin o listă de asemenea reguli și sunt inițializate în interiorul clasei **ATReport**, unde urmează să fie investigate.

#### 4.2.6 Clasa ReadWriteIniFile

**ReadWriteIniFile** este o clasă de utilitate care conține funcționalități necesare pentru interacționarea cu fișierul de inițializare Sherlock.ini. Acesta se folosește de librăria Kernel32 pentru a scrie, citi, crea și șterge informații în acest fișier de inițializare.

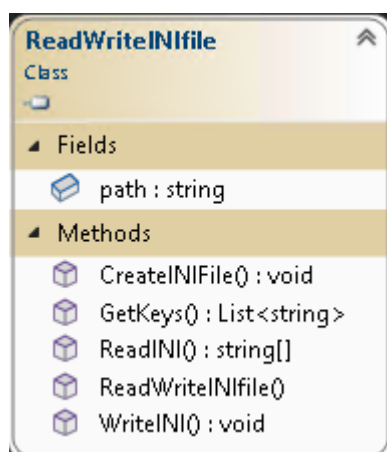


Figura 4.7: Diagrama UML a clasei ReadWriteIniFile

#### 4.2.7 Clasa TestObject

**Clasa TestObject** combină informații din clasa **OutputFile** împreună cu informațiile preluate din fișierul XML de intrare. Un obiect de tipul **TestObject** reprezintă un singur test automat eşuat găsit în baza de date.

Scopul clasei **TestObject** este acela de a aduna informațiile unui test într-un singur un loc pentru a facilita găsirea erorii.

Structura clasei **TestObject** este dată de Figura 4.8.

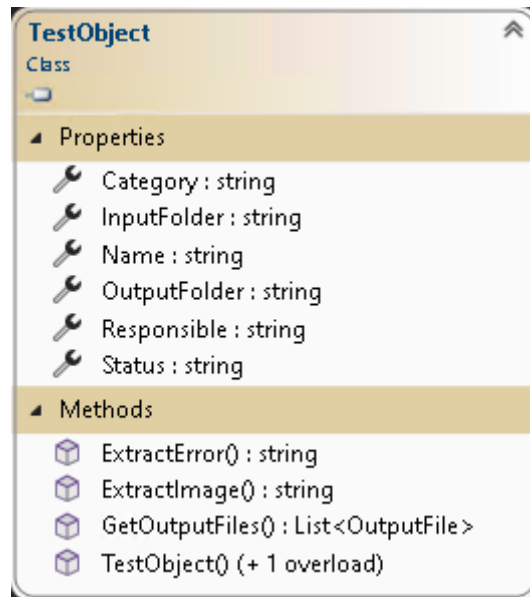


Figura 4.8: Diagrama UML a clasei TestObject

Clasa conține o funcționalitate prin care extrage eroarea din fișierele text de output ale testelor pe baza informațiilor și setului de reguli provenite din **OutputFile**, dar are și rolul de a extrage text din căsuțele de eroare care apar în screenshot-urile realizate de framework-ul care rulează testele automate în momentul în care acestea eșuează.

Funcționalitatea de extragere a textului din căsuțele de eroare se folosește de librăria **OpenCvSharp** pentru a prelucra imaginile prin intermediul apelării funcțiilor acestora precum:

- **Cv2.CvtColor(img, img\_gray, ColorConversionCodes.BGR2GRAY);**  
Funcție care convertește o imagine color într-o imagine în tonuri de gri (grayscale), unde **img** este imaginea originală, **img\_gray** este imaginea rezultată, iar ultimul parametru, **ColorConversionCodes.BGR2GRAY** reprezintă modul de convertire al imaginii, din RGB în grayscale.
- **Cv2.GaussianBlur(img\_gray, img\_blur, new OpenCvSharp.Size(5, 5), 1);**  
Metodă care aplică **filtrul Gauss** asupra imaginii grayscale rezultată din prelucrarea anterioară, obținându-se o imagine o imagine fără zgomot, corespunzătoare pentru a fi aplicat **filtrul Canny** pe ea. Parametrul **img\_blur** reprezintă rezultatul aplicării filtrului Gauss, **OpenCvSharp.Size(5,5)** reprezintă mărimea kernel-ului, 5x5, iar **1** reprezintă valoarea deviației standard a distribuției gaussiene pe axa x.
- **Cv2.Canny(img\_blur, edges, 100, 200, 3, false);**  
Funcție care aplică **filtrul Canny** asupra imaginii **img\_blur**. Rezultatul aceste procesări este o imagine binară în care sunt evidențiate marginile căsuțelor de eroare. Parametrul **edges** reprezintă imaginea rezultată în urma procesării, **100** și **200** sunt cele două valori asociate pragurilor de valoare minimă și maximă

([Vezi secțiunea 2.1](#)), **3** este dimensiunea nucleului utilizat pentru calculul gradientului, iar **false** indică metoda de calcul a magnitudinii, norma L1, în cazul în care acest parametru avea valoarea true, calculul magnitudinii se folosea de norma L2.

- **Cv2.FindContours(edges, out contours, out hierarchy, RetrievalModes.Tree, ContourApproximationModes.ApproxSimple);**

Această metodă returnează în interiorul variabilei **contours** lista de contururi găsite în imagine în interiorul variabilei contours. Parametrul **hierarchy** reprezintă informații ierarhice despre contururi, **ContourApproximationModes.ApproxSimple** specifică metoda de aproximare a conturilor, iar **RetrievalModes.Tree** specifică modul în care se găsesc conturile.

În urma acestor procesări de imagini, aplicația utilizează tehnologia **OCR** (Optical Character Recognition) ([Vezi Secțiunea 2.2.1](#)) pentru a identifica și extrage textul din căsuța de eroare. Aceasta se realizează prin găsirea celui mai potrivit contur în lista de contururi, luând în considerare parametri precum lungimea, înălțimea și rata de aspect a acestuia. Odată ce a fost identificat conturul adecvat, se apelează metoda **Process** din biblioteca **Tesseract** pentru a extrage efectiv textul din căsuța respectivă de eroare.

Metoda **Process** este apelată în felul următor:

```
ocr.Process(image, PageSegMode.Auto, TesseractEngineMode.Default);
```

unde **image** este imaginea din care urmează să fie extras textul, **PageSegMode.Auto** specifică modul în care va fi segmentată imaginea, iar **TesseractEngineMode.Default** specifică modul de funcționare a engine-ului OCR.

## 4.3 Aplicația componentă Sherlock

**Sherlock** este o aplicație consolă care creează raportul testelor automate eşuate fără a avea nevoie de niciun input de la utilizator, pe lângă fișierul de inițializare **Sherlock.ini** care trebuie configurat o singură dată.

**Sherlock** se folosește de **SherlockUtil** pentru a realiza majoritatea sarcinilor care îi sunt asiguate. Astfel, **Sherlock** conține doar următoarele 2 clase:

1. ConsoleMessage
2. SherlockMain

### 4.3.1 Clasa ConsoleMessage

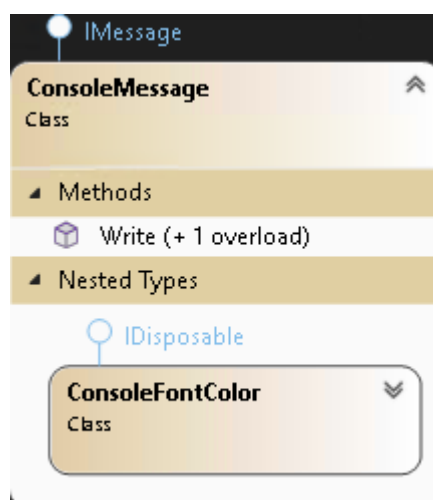


Figura 4.9: Diagrama UML a clasei ConsoleMessage

**Clasa ConsoleMessage** este o implementare a interfeței **IMessage** menționată în cadrul bibliotecii SherlockUtil, în [secțiunea 4.2.4](#). Această clasă are rolul de a facilita afișarea în consolă a mesajelor cu culori diferite în funcție de parametrii primiți la apelarea funcției **Write** din interiorul acesteia.

### 4.3.2 Clasa SherlockMain

Clasa **SherlockMain** este folosită doar pentru pornirea efectivă a execuției aplicației, aceasta creează un obiect de tipul **ATReport** și apelează funcția acestuia de creare a raportului testelor automate.

Diagrama UML a clasei **SherlockMain** este prezentată în Figura 4.10.

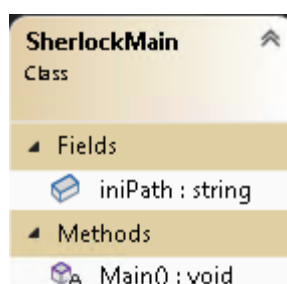


Figura 4.10: Diagrama UML a clasei SherlockMain

## 4.4 Componenta de interfață grafică SherlockUI

**SherlockUI** reprezintă opțiunea de a interacționa cu aplicația software printr-o interfață grafică, în contrast cu **Sherlock**, care nu necesită nicio interacțiune de la începerea execuției și până la sfârșitul acesteia. **SherlockUI** a fost dezvoltat în WPF, C#, folosind design patern-ul Model-View-ViewModel.

Model-View-ViewModel [8] este un design pattern menit să separe logica aplicației de controalele din interfața grafică. MVVM ajută la organizarea codului și împărțirea logicii în module pentru a face dezvoltarea, actualizarea și reutilizarea codului mai simple și mai rapide. Separarea codului în MVVM este împărțită în View, ViewModel și Model astfel:

- **Model** reprezintă datele și logica de bază a aplicației. Modelul reprezintă structura datelor și funcționalitățile care lucrează cu aceste date. Un model poate fi o clasă simplă sau poate implica și interacțiunea cu o bază de date sau servicii externe.
- **View** este partea care conține cod și primește input de la utilizator prin intermediul controalelor prezente în interfața grafică. Conținutul din View nu interacționează direct pentru a schimba ceea ce este afișat.
- **ViewModel** acționează ca intermediar între Model și View. ViewModel conține logica de afaceri și funcționalitatea necesară pentru manipularea datelor și interacțiunea cu Modelul. De asemenea, expune proprietăți și comenzi care sunt legate de interfața utilizatorului și pot fi legate direct la View pentru a actualiza și afișa datele.

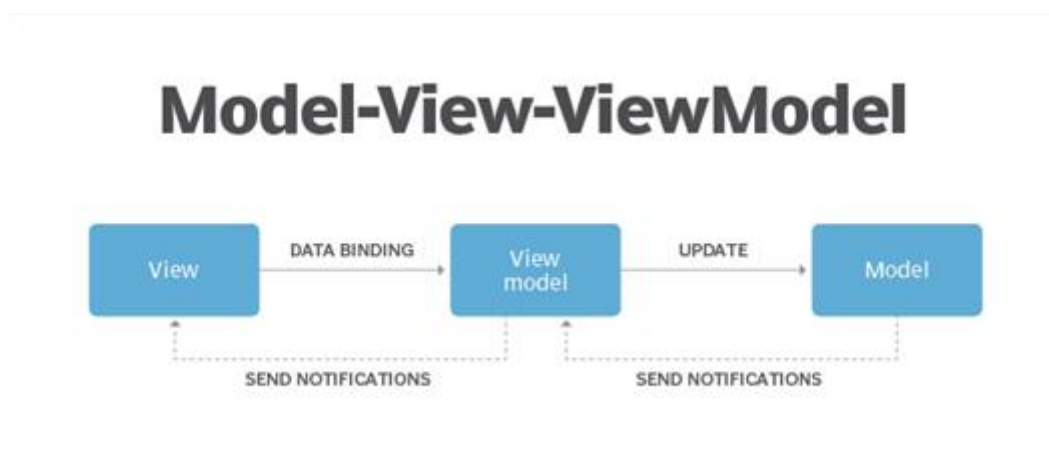


Figura 4.11: Prezentare MVVM [8]

Componenta de interfață grafică oferă aceleași funcționalități pentru crearea raportului ca și **Sherlock**, dar conține în plus câteva componente de feedback și



posibilitatea de a modifica input-ul direct din aceasta, fără a mai fi nevoie să fie modificat fișierul de inițializare în mod direct.

Altă diferență dintre **SherlockUI** și **Sherlock** este că după ce se realizează raportul în interfața grafică, acesta este deschis automat și utilizatorul are posibilitatea de a reprograma testele eșuate apăsând un buton. Funcționalitatea de reprogramare a testelor automate eșuate va reprograma toate testele din fișierul Excel a căror valoare din câmpul de status a fost schimbată în „To reschedule”.

**SherlockUI** este împărțit într-o multitudine de clase. Respectând design pattern-ul MVVM, acesta are clase de tip Model, ViewModel și View. Așadar, SherlockUI conține următoarele clase:

1. Model:
  - ListItem
  - LogMessage
  - WindowMessage
2. ViewModel:
  - INIDialogViewModel
  - MainWindowViewModel
  - ViewModelBase
3. View:
  - EditINIDialog
  - MainWindow

#### 4.4.1 Clasele ListItem și LogMessage

Scopul claselor **ListItem** și **LogMessage** este acela de a simplifica afișarea mesajelor de logging în interfața grafică. Acestea au o implementare relativ simplă, conținând doar câteva proprietăți fiecare și corespund diagramelor UML din **figurile 4.12 și 4.13**.

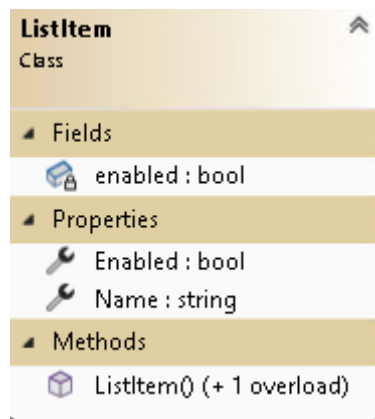


Figura 4.12: Diagrama UML a clasei ListItem

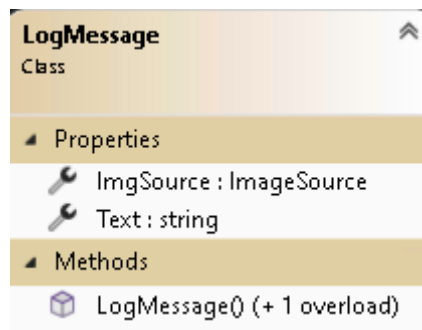


Figura 4.13: Diagrama UML a clasei LogMessage

#### 4.4.2 Clasa WindowMessage

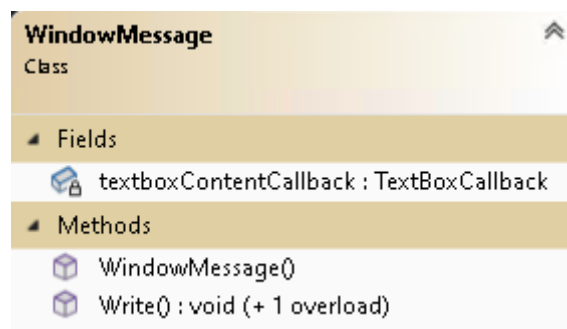


Figura 4.14: Diagrama UML a clasei WindowMessage

Clasa **WindowMessage** este o implementare a interfeței **IMessage** menționată în cadrul bibliotecii SherlockUtil, în [secțiunea 4.2.4](#).

Scopul lui **WindowMessage** este acela de a afișa mesaje de tipul **LogMessage** în interiorul interfeței grafice. Fiecare dintre mesajele respective conține text, dar și o imagine care să ajute utilizatorul să își dea seama dacă mesajul respectiv a apărut din cauza unei erori sau este un simplu mesaj produs de aplicație care are ca scop să țină utilizatorul la curent cu ceea ce se întâmplă în aplicație.

Este de menționat faptul că **textboxContentCallback** este un delegat care trimite un semnal către **MainWindowViewModel** ([Subsecțiunea 4.4.4](#)). Atunci când este apelată metoda **Write** de către un obiect **ATReport**, delegatul notifică viewmodel-ul și îi trimite acestuia informațiile necesare pentru a afișa mesajul în interfața grafică.

#### 4.4.3 Clasele **INIDialogViewModel** și **EditINIDialog**

**INIDialogViewModel** este viewmodel-ul care acționează asupra view-ului **EditINIDialog**, conform design pattern-ului Model View ViewModel. Diagrama UML a clasei **INIDialogViewModel** este reprezentată în [Figura 4.15](#).

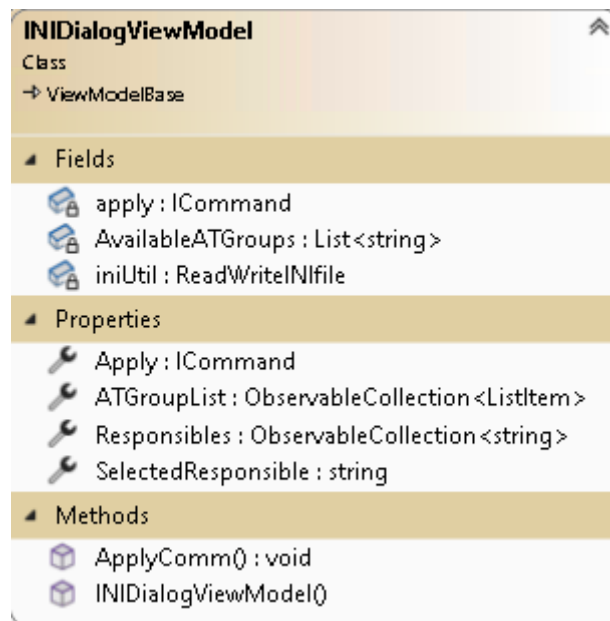


Figura 4.15: Diagrama UML a clasei **INIDialogViewModel**

Scopul acestor clase este acela de a crea o fereastră prin intermediul căreia se pot schimba datele aferente din fișierul de inițializare **Sherlock.ini** prin simpla apăsare a unor butoane, fără a mai fi nevoie ca utilizatorul să modifice manual informațiile din fișier.

În design pattern-ul MVVM, un view reprezintă interfața utilizatorului sau stratul de prezentare a unei aplicații, așadar clasa **EditINIDialog** este responsabilă de afișarea datelor utilizatorului și de primirea input-ului acestuia și a fost creată utilizând XAML în Windows Presentation Foundation(WPF).

Pe de altă parte, un view model acționează ca intermediar între vedere și datele sau logica aplicației. **INIDialogViewModel** furnizează date și comportamente de care **EditINIDialog** are nevoie pentru se afișa și a interacționa, separând eficient interfața utilizator de date.

**INIDialogViewModel** furnizează date precum lista de ATGroup-uri disponibile și care dintre acestea sunt folosite în fișierul ini și lista de responsabili disponibili care sunt legate ulterior în **EditINIDialog** pentru a fi afișate.

#### 4.4.4 Clasele MainWindowViewModel și MainWindow

La fel ca și în cazul claselor **INIDialogViewModel** și **EditINIDialog**, clasele **MainWindowViewModel** și **MainWindow** sunt legate între ele, **MainWindowViewModel** punând la dispoziție datele de care **MainWindow** are nevoie pentru a asigura toate funcționalitățile care apar în interfața utilizatorului.

Pe lângă acestea, **MainWindowViewModel** furnizează date și pentru a altera puțin modul în care funcționează funcțiile din clasa **ATReport**(Vezi [Subsecțiunea 4.2.1](#)) a lui **SherlockUtil** față de cum ar funcționa în cazul aplicației consolă **Sherlock**.

Diagrama UML a clasei **MainWindowViewModel** este reprezentată în [Figura 4.16](#).

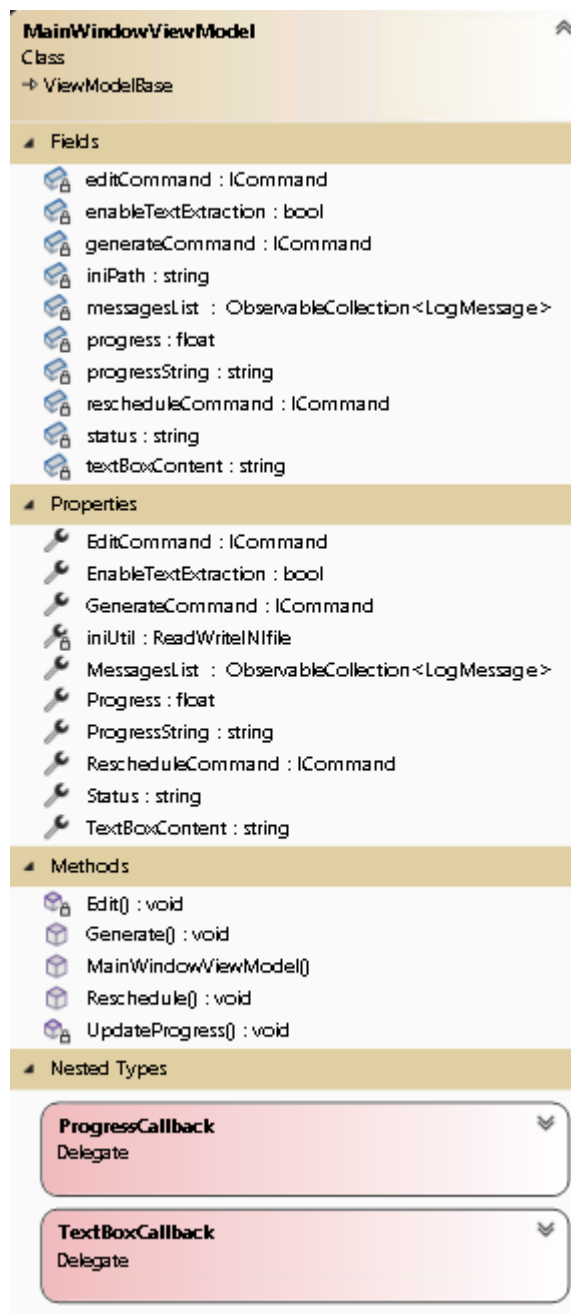


Figura 4.16: Diagrama UML a clasei **MainWindowViewModel**

În secțiunea de proprietăți se pot observa datele trimise către view-ul **MainWindow** pentru a facilita acțiunile acestuia. Membrii de tip **Command** ajută la legarea butoanelor din interfață de secțiunile de cod pe care acestea trebuie să le execute.

Comenzile de **Generate** și **Reschedule** cheamă funcțiile clasei **ATReport** pentru creare a raportului și respectiv reprogramare a testelor din raport. În timp ce acestea se execută, delegatul **ProgressCallback** actualizează proprietatea de progres în funcție de ceea ce se întâmplă în aceste funcții din **ATReport**. Mai exact, progresul este actualizat de fiecare dată când s-a găsit o eroare în fișierele testelor eșuate sau de

fiecare dată când datele unui test eşuat au fost introduse în tabelul Excel. În continuare, progresul este trimis la **MainWindow** care, pe baza acestuia, completează o bară de progres prezentă în interfața grafică a utilizatorului pentru a oferi un feedback cât mai precis posibil.

Pe lângă **ProgressCallback**, clasa mai conține și delegatul **TextBoxCallback**, responsabil pentru schimbarea valorii mesajului de status aflat în interfața grafică pe baza a mesajelor produse de clasa **ATReport**.

Cele două conțin de asemenea și o funcționalitate pentru afișarea pe ecran a view-ului **EditINIDialog**, dar și o listă de mesaje de tip **WindowMessage** care sunt afișate în interfața grafică pe parcursul executării generării raportului testelor automate, dar și în timpul reprogramării acestora.

Încă o funcționalitate este reprezentată de posibilitatea utilizatorului de a specifica dacă la timpul generării raportului acesta dorește ca textul din căsuțele de eroare corespunzătoare testelor să fie extras sau nu (printr-un câmp prezent în fișierul **Sherlock.ini**), întrucât acesta poate crește considerabil timpul de execuție al generării.

În interiorul clasei **MainWindow**, pentru a fi realizată interfața grafică s-au folosit limbajul XAML precum cel ce se poate observa în **Figura 4.17**.

```
<Button FontWeight="Bold" Content="Edit INI File" Height="50" Width="200"
VerticalAlignment="Top" HorizontalAlignment="Left" Command="{Binding EditCommand}"
Margin="51,156,0,0"></Button>
<ProgressBar Value="{Binding Progress}" Minimum="0" Maximum="100" Width="800"
Height="20" HorizontalAlignment="Center" VerticalAlignment="Center" Margin="0,467,42,32"/>
<TextBlock Text="{Binding ProgressString}" TextAlignment="Center" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="800" Height="20" Margin="0,467,42,32"></TextBlock>
<Expander IsExpanded="False" ExpandDirection="Up" Width="800" Height="140"
VerticalAlignment="Bottom" HorizontalAlignment="Left" Margin="0, 0, 0, 0">
<ListView x:Name="MyListView" ItemTemplate="{StaticResource ItemsTemplate}"
ItemsSource="{Binding MessagesList}">
</ListView>
</Expander>
```

Figura 4.17: Realizarea legăturii dintre MainWindowViewModel și MainWindow prin intermediul limbajului XAML, extrasă din view-ul MainWindow

În **Figura 4.17** codul XAML definește aspectul și comportamentul interfeței grafice a aplicației. Fiecare element vizual, cum ar fi caseta de selectare, butoanele, bara de progres și expander-ul, este legat de proprietăți și comenzi definite în **MainWindowViewModel**.

#### 4.4.5 Clasa ViewModelBase

Clasa **ViewModelBase** implementează interfața **INotifyPropertyChanged** și îi corespunde diagramei UML din **Figura 4.18**.

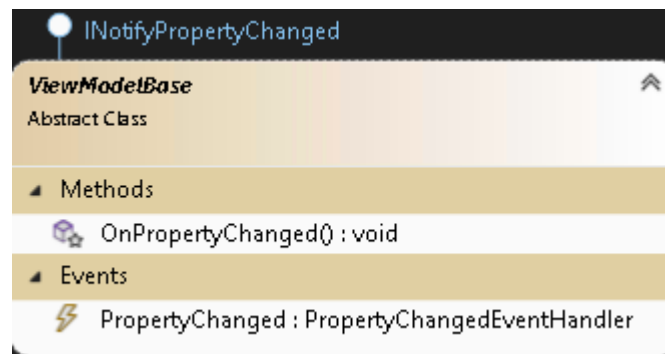


Figura 4.18: Diagrama UML a clasei **ViewModelBase**

Scopul clasei **ViewModelBase** este de a furniza o funcționalitate comună pentru clasele derivate, precum **MainWindowViewModel** și **INIDialogViewModel**, în ceea ce privește implementarea interfeței **INotifyPropertyChanged**. Implementarea interfeței **INotifyPropertyChanged** este necesară pentru a permite actualizarea automată a interfeței utilizatorului (UI) atunci când datele din viewmodel se modifică.

Prin moștenirea clasei **ViewModelBase**, clasele derivate pot beneficia de implementarea deja existentă a interfeței **INotifyPropertyChanged**, eliminând necesitatea rescrierii aceleiași funcționalități în fiecare viewmodel în parte. Aceasta reduce duplicarea codului și crește eficiența dezvoltării.

Această abordare de a avea o clasă de bază (**ViewModelBase**) care implementează funcționalitate comună și care este moștenită de alte view model-uri este un exemplu de principiul „Don't Repeat Yourself” (DRY) în dezvoltarea software, care promovează reutilizarea codului și eliminarea duplicării inutile, drept urmare obținând un design al aplicației mai eficient, mai ușor de întreținut și mai scalabil.

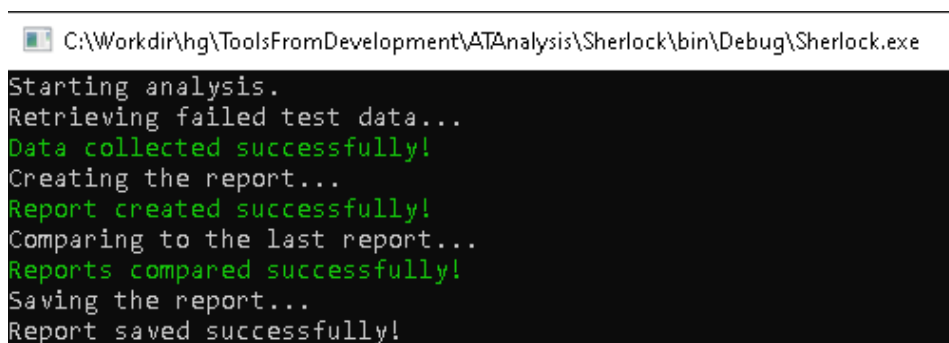
## 5. Ghid de utilizare al aplicației

Există două moduri prin care un utilizator poate folosi aplicația, și anume:

- Prin intermediul executabilului aplicației consolă **Sherlock**, care va crea raportul testelor automate eșuate fără a fi nevoie de input de la utilizator
- Prin intermediul aplicației **SherlockUI** care așteaptă input-uri de la utilizator, pentru a începe analiza sau pentru a reprograma testele eșuate marcate cu statusul „To reschedule” în fișierul Excel.

Totuși, ambele module depind de un fișier de inițializare, și anume **Sherlock.ini**. Utilizatorul trebuie să introducă în acesta informații despre testele pe care vrea să le vizualizeze în raportul său. Aceste informații de care componentele aplicației au nevoie se referă la grupuri de teste automate și configurările acestora, cum se poate observa în exemplul din [subsecțiunea 4.1.1](#). Singurul tip de informații care ar putea fi schimbate în mod regulat după dorința utilizatorului din acest fișier sunt acelea despre grupurile de teste automate, dar pe lângă acestea, fișierul poate rămâne neschimbat pentru o perioadă îndelungată de timp, până la apariția noilor configurații de teste.

După ce utilizatorul și-a configurat fișierul **Sherlock.ini** după propriile sale nevoi, acesta poate inițializa aplicația. În cazul în care utilizatorul nu își dorește să interacționeze cu aplicația printr-o interfață grafică, ci dorește doar să primească raportul testelor automate eșuate într-un timp cât mai scurt, acesta va folosi executabilul **Sherlock**. În [Figura 5.1](#) este prezentată o execuție de succes a aplicației consolă **Sherlock** în urma căreia s-a întocmit fișierul Excel.



```
C:\Workdir\hg\ToolsFromDevelopment\ATAnalysis\Sherlock\bin\Debug\Sherlock.exe
Starting analysis.
Retrieving failed test data...
Data collected successfully!
Creating the report...
Report created successfully!
Comparing to the last report...
Reports compared successfully!
Saving the report...
Report saved successfully!
```

Figura 5.1: Captură a aplicației consolă Sherlock

În cazul în care utilizatorul își dorește să utilizeze interfața grafică a aplicației, acesta va folosi executabilul aplicației **SherlockUI**.

Când utilizatorul apasă butonul **Generate AT Report**, aplicația începe generarea raportului la fel ca în cazul lui **Sherlock**. În timpul creării fișierului Excel, bara de progres din partea de jos a interfeței grafice este umplută în mod progresiv și precis de fiecare dată când se realizează un pas din generarea raportului. De asemenea, pe lângă bara de



progres vor fi afișate și mesaje de logging care îi arată utilizatorului în mod constant statusul generării raportului testelor.

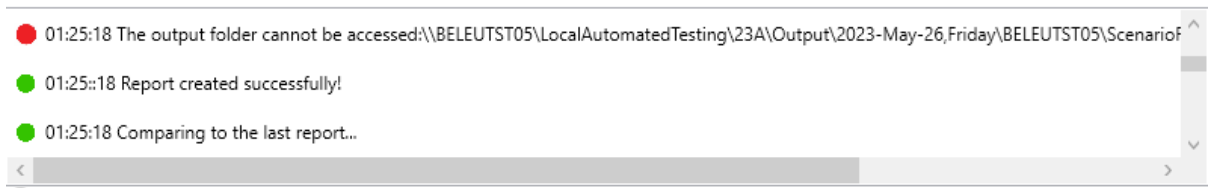


Figura 5.2: Mesaje de logging apărute în timpul executării aplicației

În **Figura 5.2** se pot observa câteva dintre mesajele care apar pe parcursul generării raportului. Acestea vor avea o bulină verde pentru situația în care mesajul apărut semnifică un simplu mesaj care reprezintă o informație despre statusul aplicației, sau o bulină roșie în cazul apariției unei erori. Este de menționat faptul că mesajul „Comparing to the last report...” se referă la compararea valorilor coloanelor **Investigator**, **Status** și **Comments** din ultimul raport creat cu cele apărute în raportul curent. În cazul în care un test eșuat din raportul precedent prezintă anumite valori în aceste câmpuri și acest test apare și în raportul curent, valorile câmpurilor sunt preluate și adăugate în raportul curent. De asemenea, sub coloana **Test** se poate regăsi numele testului împreună cu un hyperlink către fișierele sale de ieșire, în cazul în care utilizatorul își dorește totuși să vizualizeze manual rezultatele acestuia. Același lucru se întâmplă și sub coloana **Input**, diferența este că hyperlink-ul duce la fișierele de intrare ale acestuia. Sub coloana **IMG** vor apărea anumite iconițe în cazul în care în fișierele de ieșire a fost găsită o imagine, iconiță care, fiind apasată va deschide fișierul „.jpg” asociat imaginii. În cazul în care fișierele de ieșire nu conțin o imagine, această coloană nu va avea asociată nimic.

În cazul în care execuția a avut succes, la finalul acesteia se va deschide foaia de lucru Excel unde este plasat raportul.

În fișierul Excel creat toate testele noi care nu apar în cadrul ultimului raport creat precedent vor avea valoarea „**Unresolved**” în interiorul căsuței asociate coloanei status. Dacă utilizatorul își dorește să reprogrameze anumite teste eșuate, acesta va alege opțiunea **Reschedule Tests**. La apăsarea acestui buton, testele care au în dreptul coloanei **Status** afișată valoarea „**To reschedule**” vor fi reprogramate, iar valoarea din câmpul respectiv va fi schimbată în „**Rescheduled**”.

Category	Test	ErrorMessage	Investigator
ROT_Neo_GUI	<a href="#">FilterFRE</a>	[Error][09:11:19.367] ScenarioExecutor.PrintException(95): Type: System.Runtime.InteropServices.COMException [Error][09:11:19.429] ScenarioExecutor.PrintException(96): Message: Retrieving the COM class factory for component with CLSID {5Bf68339-120F-40C4-8196-C3EC36AC7906} failed due to the following error: 800706bf The remote procedure call failed and did not execute. (Exception from HRESULT: 0x800706bf). [Error][09:11:19.554] ScenarioExecutor.PrintException(97): StackTrace: at AutomationTest.ATTestContext.InitConfigLevels() in N:\TestLab\Cada-NT\UmsHq\RegressionTest\DesktopUIAutomationTest_v1\AutomationTest\ATTestContext.cs:line 173 at AutomationTest.ATTestContext.get_GroupConfig() in N:\TestLab\Cada-NT\UmsHq\RegressionTest\DesktopUIAutomationTest_v1\AutomationTest\ATTestContext.cs:line 112 at TestFramework.ScenarioExecutor.PrintCurrentTestContext() in N:\TestLab\Cada-NT\UmsHq\RegressionTest\DesktopUIAutomationTest_v1\TestFramework\ScenarioExecutor.cs:line 41 at TestFramework.ScenarioExecutor..ctor(ITestContext testContext, IList`1 scenarioNames) in N:\TestLab\Cada-NT\UmsHq\RegressionTest\DesktopUIAutomationTest_v1\TestFramework\ScenarioExecutor.cs:line 20 at AutomationTest.Program.Main(String[] args) in N:\TestLab\Cada-NT\UmsHq\RegressionTest\DesktopUIAutomationTest_v1\AutomationTest\Program.cs:line 27 [Error][09:11:19.617] ScenarioExecutor.PrintException(101): InnerException: null	


Status	Comments	Responsible	Input	Reschedule	IMG	Image Text	MOV
To reschedule		msqjhp - d2acrd	<a href="#">FilterFRE</a>				

Figura 5.3: Marcarea unui test pentru reprogramare

În **Figura 5.3** este prezentat modul în care trebuie marcat un test în cazul în care se dorește reprogramarea acestuia, marcarea fiind un pas absolut necesar. La finalizarea rulării logicii de reprogramare a testelor eşuate, utilizatorul poate verifica ce teste au fost reprogramate în 2 moduri: prin intermediul Excel sau prin intermediul interfeței grafice, unde sunt afișate mesaje referitoare la execuția reprogramării.

La folosirea opțiunii **Edit INI File** se va deschide o nouă fereastră unde utilizatorul poate schimba datele prezente în fișierul Sherlock.ini din cadrul interfeței grafice. Această fereastră conține un GridView în care este afișată o listă de grupuri de teste automate, unde fiecăruia îi corespunde câte un CheckBox care specifică dacă acel grup de teste figurează sau nu în fișierul de inițializare.

Ca ultimă opțiune prezentă în pagina principală a aplicației **SherlockUI**, utilizatorul poate bifa sau debifa checkbox-ul asociat textului **Image text extraction**. Consecințele acestei acțiuni vor fi regăsite în cadrul fișierului .ini și al Excel: în cazul în care checkbox-ul a fost bifat, setarea EnableTextExtraction din fișierul **Sherlock.ini** va primi valoarea 1, pe baza căreia în timpul generării raportului vor fi afișate sau nu în fișierul Excel mesajele de eroare corespunzătoare imaginilor prezente în fișierele de output ale testelor automate, dacă acestea există.

# 6. Concluzii

## 6.1 Concluzii generale

Lucrarea prezintă o aplicație software care are ca scop crearea automată unui raport care conține informații relevante despre testele automate eșuate în urma rulării lor de către o platformă de lucru. La momentul conceperii aplicației, pe piață nu există alte soluții care să îndeplinească această sarcină, întrucât platforma de lucru care execută aceste teste produce niște rezultate foarte specifice și niște informații care pot fi accesate doar în rețeaua de lucru a SIEMENS Industry Software.

Avantajele acestei aplicații sunt următoarele:

### **1. Ușurința de utilizare**

În cazul în care utilizatorul decide că tot ce vrea să obțină de la aplicație este generarea unui raport al testelor automate eșuate într-un timp cât mai scurt, acesta va folosi executabilul Sherlock, așadar, acesta trebuie doar să ruleze aplicația consolă și va obține raportul.

În cazul în care utilizatorul dorește să interacționeze cu aplicația printr-o interfață grafică, acesta va folosi executabilul SherlockUI de unde are și opțiunile să configureze fișierul de inițializare, dar în cele mai multe cazuri acesta nu necesită schimbări. În cazul acesta, aplicația este la fel de ușor de folosit, singura sarcină a utilizatorului fiind să apese butonul de generare a raportului.

### **2. Automatizarea procesului**

Această aplicație automatizează o sarcină care în trecut trebuia să fie efectuată de către membrii echipei, fapt care consuma timp și resurse umane. De asemenea, aceasta elimină și posibilitatea apariției erorilor umane și poate spori productivitatea prin economisirea timpului de lucru.

### **3. Timpul scurt de generare a raportului**

Generarea raportului poate dura oriunde între 10-15 secunde și câteva minute, în funcție de numărul testelor eșuate și de complexitatea rezultatelor produse de acestea. Așadar, acesta reduce foarte mult cantitatea de efort depusă de către echipa de dezvoltatori, cărora le putea consuma această sarcină o cantitate mare de timp.

#### **4. *Flexibilitate***

Datorită fișierului de inițializare utilizat în aplicație, aceasta oferă utilizatorului posibilitatea de a obține un raport personalizat pentru nevoile acestuia. Tot acest fapt face posibilă și utilizarea aplicației pentru mai multe echipe interesate.

Un dezavantaj notabil al aplicației este că aceasta este specializată pentru un domeniu restrâns de activitate, concentrându-se exclusiv asupra rezultatelor generate de platforma de lucru care rulează testele automate.

### **6.2 Dezvoltare ulterioară**

Pot fi considerate posibilități de extindere a aplicației următoarele:

#### **1. *Automatizarea și planificarea generării de rapoarte***

Având în vedere faptul că aplicația consolă Sherlock generează raportul fără a avea nevoie de input de la utilizator, poate fi luată în considerare posibilitatea de a programa execuția acesteia, pentru a obține rezultatele dorite de utilizator automat după ce platforma de lucru a terminat de rulat testele. De asemenea, o altă posibilitate de extindere în cazul în care planificarea va fi introdusă este posibilitatea aplicației de a notifica utilizatorii acesteia în momentul în care un raport a fost creat pentru configurațiile lor.

#### **2. *Extinderea suportului pentru diferite platforme***

O altă posibilitate de extindere a aplicației este dezvoltarea într-un mod și mai generic astfel încât aceasta să poată realiza rapoarte automate pentru mai multe configurații, nu doar pentru cele produse de către platforma de lucru internă.

# 7. Bibliografie

- [1] Vipin Tyagi, *Understanding Digital Image Processing*, Jaypee University of Engineering and Technology, 2018
- [2] Andrew Bensley Adams, *High-dimensional Gaussian Filtering for Computational Photography*, Stanford University, 2011
- [3] Canny Edge Detection, OpenCv  
[https://docs.opencv.org/3.4/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html)
- [4] Python OpenCv – Canny() Function, GeeksForGeeks  
<https://www.geeksforgeeks.org/python-opencv-canny-function/>
- [5] Zhi-Hua Zhou, *Machine Learning*, Nanjing University, 2021
- [6] Ravina Mithe, Supriya Indalkar, Nilam Divekar, *Optical Character Recognition*, International Journal of Recent Technology and Engineering (IJRTE) Volume-2, Issue-1, March 2013  
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=6a4b4f04d5ce3c3592832eb40c23cc8fc5a9131e>
- [7] Hmrishav Bandyopadhyay, Optical Character Recognition: What is it and How Does it Work, July 6, 2021  
<https://www.v7labs.com/blog/ocr-guide>
- [8] Model-View-ViewModel (MVVM)  
<https://www.techtarget.com/whatis/definition/Model-View-ViewModel>
- [9] WPF overview, Microsoft Article, August 12, 2021  
<https://learn.microsoft.com/en-us/dotnet/desktop/wpf/introduction-to-wpf?view=netframeworkdesktop-4.8&preserve-view=true>
- [10] XML, Wikipedia  
<https://en.wikipedia.org/wiki/XML>
- [11] ADO.NET Overview, Microsoft Article, September 15, 2021  
<https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview>
- [12] What is SQL Server  
<https://www.sqlservertutorial.net/getting-started/what-is-sql-server/>
- [13] Learn LINQ,  
<https://www.tutorialsteacher.com/linq>

[14] XAML, Microsoft

<https://learn.microsoft.com/en-us/visualstudio/xaml-tools/xaml-overview?view=vs-2022>

[15] MahApps.Metro – Documentation

<https://mahapps.com/docs/>

7%

SIMILARITY INDEX

4%

INTERNET SOURCES

1%

PUBLICATIONS

6%

STUDENT PAPERS

---

PRIMARY SOURCES

---

1

Submitted to West University Of Timisoara

Student Paper

2%

2

Submitted to Technical University of Cluj-Napoca

Student Paper

1%

3

Submitted to Politehnica University of Timisoara

Student Paper

1%

4

Submitted to University of Bucharest

Student Paper

1%

5

[www.coursehero.com](http://www.coursehero.com)

Internet Source

1%

6

Submitted to unibuc

Student Paper

<1%

7

Submitted to University Politehnica of Bucharest

Student Paper

<1%

8

[users.utcluj.ro](http://users.utcluj.ro)

Internet Source

<1%