

**Raport: Concepte și metode din articolul**  
***TestLab: An Intelligent Automated***  
***Software Testing Framework***

Stoinea Maria Miruna

Nazare Elena-Denisa

Ciurescu Irina Alexandra

Antonescu Ionut-Andrei

# 1. Introducere / Context

Deoarece sistemele software sunt peste tot, ele nu mai sunt doar opționale sau de nișă, ci parte fundamentală a vieții de zi cu zi. Astfel, aplicațiile au devenit din ce în ce mai mari și mai complicate. Nu mai e vorba de aplicații simple, ci de sisteme complexe, cu sute de mii de linii de cod și componente interconectate. Această complexitate are un cost: durează mai mult să proiectezi, implementezi, testezi și întreții software. [2]

Articolul abordează problema calității scăzute a software-ului, cauzată adesea de testarea insuficientă sau inexistentă, din dorința de a accelera ciclul de dezvoltare. În acest context, este propus *TestLab*, un cadru inteligent pentru testarea automată a software-ului, care îmbină diverse metode de testare cu tehnici de Inteligență Artificială (IA) pentru a asigura testarea continuă și eficientă a aplicațiilor software.

TestLab este gândit să funcționeze pe toată durata dezvoltării software:

- **Dezvoltatori:** verifică codul sursă.
- **Tester:** generează cazuri de test.
- **Utilizatori finali:** validează comportamentul aplicației.

Și face asta la toate nivelurile:

- unitate
- integrare
- sistem
- acceptanță

Este împărțit în 3 componente specializate, fiecare cu un rol: descoperă vulnerabilități în API-uri (FuzzTheREST), analizează codul pentru riscuri (VulnRISKatcher), generează automat teste (CodeAssert).

## 2. Concepte-cheie

Autorii subliniază cele două obiective fundamentale ale testării:

- Validare: sistemul construit răspunde cerințelor utilizatorului? (realizată de utilizatori și client)
- Verificare: implementarea respectă specificațiile și regulile? (realizată de o echipă tehnică)

### Testarea automată a software-ului:

Executarea testelor fără intervenție umană, cu scopul de a identifica erori și vulnerabilități în mod rapid și eficient.

## Testarea pe mai multe niveluri:

- Unit testing – pe componente izolate
- Integration testing – interacțiunea dintre module
- System testing – testarea întregului sistem
- Acceptance testing – validare de către utilizatori

## Tipuri de testare:

- White-box: Analizează logica internă a codului.
- Black-box: Testează funcționalitatea fără a cunoaște implementarea internă.
- Grey-box: Combină abordările de mai sus.

## Inteligență Artificială în testare:

Utilizarea tehnicilor de machine learning și reinforcement learning pentru generarea automată de cazuri de test și detectarea vulnerabilităților.

După compararea unor framework-uri populare, autorii au concluzionat că nu există o soluție completă care să acopere toate tipurile de testare (black, white, grey), toate nivelurile (unit, integration, system, acceptance) și să integreze totul într-un singur cadru automatizat — de aici necesitatea și originalitatea TestLab. [3]

## 3. Structura și modulele TestLab

TestLab este compus din trei module principale:

### a. FuzzTheREST

- Fuzzer inteligent pentru API-uri REST, bazat pe Reinforcement Learning
- Funcționează în mod black-box, generând input-uri deformate și folosind feedback-ul API-ului
- Poate fi extins la grey-box prin analiza execuției codului

## **b. VulnRISKatcher**

- Detectează vulnerabilități în codul sursă, folosind modele de machine learning
- Suportă mai multe limbaje și funcționează pe fragmente de cod

## **c. CodeAssert**

- Automatizează generarea de scripturi de test folosind analiza codului și NLP
- Asigură acoperire de 100% pentru testare unitară și de integrare (white-box)

# **4. Exemplu practic**

## **a. Aplicația: Developer Toolbox**

### **Descriere:**

Platformă interactivă pentru învățarea și exersarea programării. Utilizatorii pot adresa întrebări, rezolva exerciții, urmări progresul și participa la provocări săptămânale.

### **Funcționalități principale:**

- Întrebări și răspunsuri tehnice
- Exerciții de programare cu editor integrat
- Puncte de reputație, medalii, clasamente
- Provocări săptămânale
- Administrare și moderare

### **Beneficii:**

- Începătorii pot învăța și primi ajutor
- Utilizatorii avansați pot contribui și rezolva exerciții complexe
- Platforma încurajează competiția și învățarea continuă

### **Tehnologii utilizate:**

- ASP.NET Core, C# – aplicație web
- Entity Framework Core, LINQ – gestionarea datelor

- SQL Server – stocarea datelor
- JavaScript, HTML, CSS – interfață
- Python – server backend și compilator

### **Scopuri pentru testare:**

Validarea funcționalităților aplicației: logare, gestionare întrebări/răspunsuri, exerciții, recompense.

## **b. Implementarea primului modul – FuzzTheREST**

RESTler [4] – un instrument de fuzzing pentru API-uri REST.

### Caracteristici:

- Instrument „stateful”, analizează comportamentul anterior al aplicației
- Folosește Reinforcement Learning pentru generarea input-urilor
- Evită inputurile invalide, optimizează procesul de testare
- Poate integra analiza execuției pentru grey-box testing

Cum funcționează RESTler:

1. Input de la tester: fișier OpenAPI, scenarii, parametri RL
2. Generare input-uri: bazată pe feedback-ul de la API
3. Evaluarea vulnerabilităților: detectează erori/vulnerabilități și ajustează căutarea
4. Raport final: vulnerabilități descoperite, input-uri testate, metrice de acoperire

RESTler eficientizează testarea automată a API-urilor REST și poate identifica vulnerabilități greu de descoperit manual.

### **Pașii de utilizare RESTler:**

#### **1. compile – Generează testele din Swagger**

- Citește swagger.json
- Creează un fișier grammar.py cu toate endpointurile

- Asociază parametri, metode, body-uri

## **2. test - Testarea cu ajutorul gramaticii modificate a rutelor**

Pentru a putea rula testele:

- Modificăm noi mai întâi grammar.py, deoarece cel autogenerat funcționează mai mult ca și template
- Ne asigurăm că baza de date corespunde pentru teste

## **3. fuzz – Rulează testele**

- Trimite automat cereri către API (inclusiv date eronate)
- Încearcă combinații diferite
- Înregistrează coduri de răspuns, crash-uri, secvențe valide

(Opțional:)

## **4. analyze – Analizează rezultatele**

- Detectează ce requesturi au eșuat (ex: 404, 500)
- Generează rapoarte (loguri, acoperire, bug-uri)

În vederea realizării primului pas, acela de compile, am avut nevoie mai întâi să generăm, utilizând Swagger, fișierul Swagger.json.

## **Ce este Swagger?**

Swagger este un format standard pentru descrierea unui API REST. Fișierul "swagger.json" este generat automat prin intermediul framework-ului ASP.NET Core și conține:

- toate endpoint-urile unui API (GET, POST, etc.)
- ce parametri sunt acceptați
- ce tipuri de date returnează
- cum trebuie să arate un request activ

Generarea automată de către documentație cu ajutorul Swagger se realizează prin introducerea a următoarelor comenzi în fișierul Program.cs:

C#

```
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();  
...  
if (app.Environment.IsDevelopment())  
{  
    app.UseSwagger();  
    app.UseSwaggerUI();  
}
```

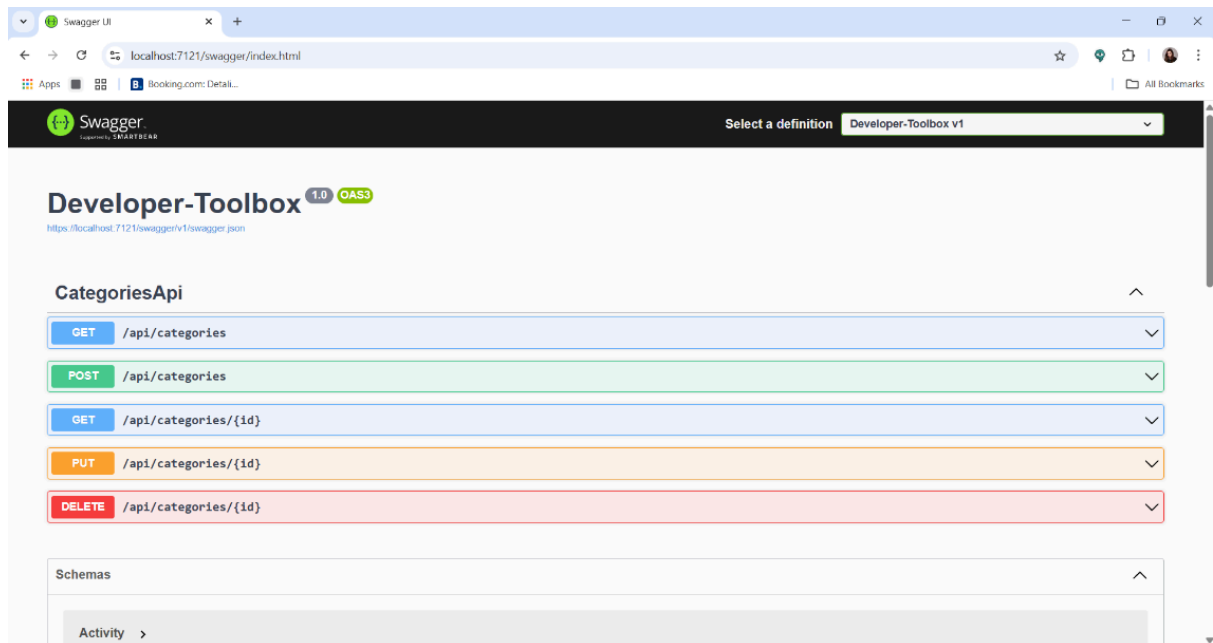
Pentru ca Swagger să poată detecta automat rutele și să realizeze maparea API-urilor, are nevoie de controllere speciale care să definească clar endpoint-urile cu ajutorul atributelor .NET (ex. [ApiController], [HttpGet] ..), trebuie să extindă clasa ControllerBase (nu Controller) și să expună metode care returnează JSON / IActionResult, nu Views.

Astfel, pentru a putea utiliza tool-ul, am creat un controller special de test numit "CategoriesApiController.cs". Iar odată cu realizarea acestui controller special, am putut accesa fișierul swagger autogenerat ([https://localhost: {port}/swagger/v1/swagger.json](https://localhost:{port}/swagger/v1/swagger.json)) cât și o vizualizare interactivă a rutelor ([https://localhost: {port}/swagger](https://localhost:{port}/swagger)).

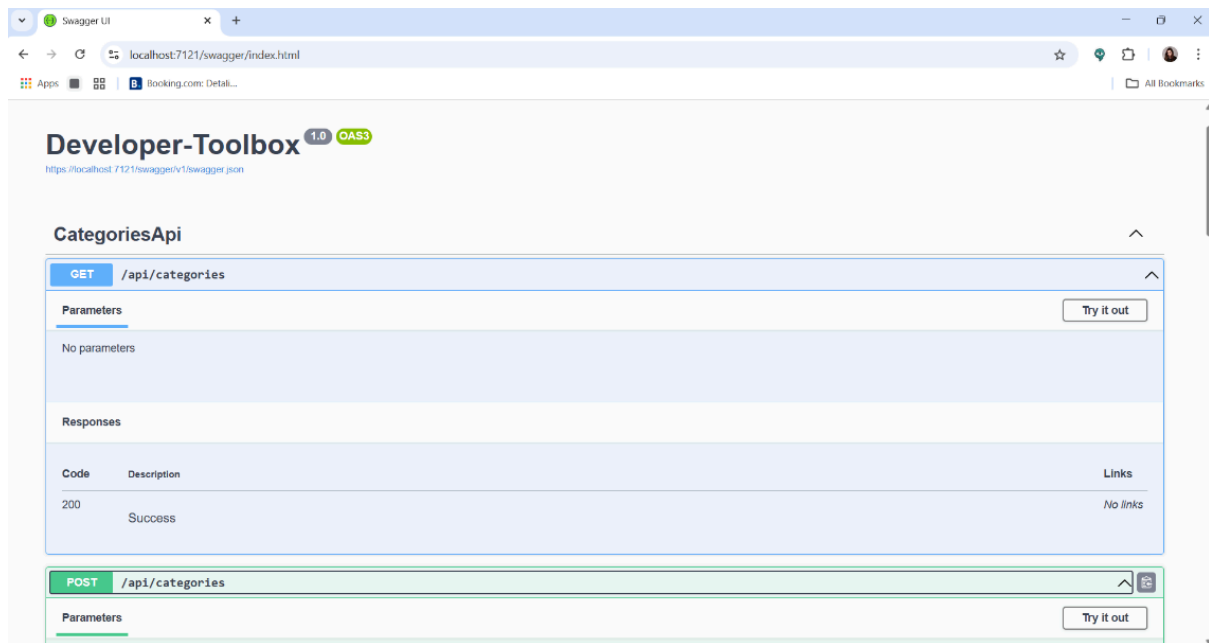
Fișierul "swagger.json" arată astfel:

```
1  {
2    "openapi": "3.0.1",
3    "info": {
4      "title": "Developer-Toolbox",
5      "version": "1.0"
6    },
7    "paths": {
8      "/api/categories": {
9        "get": {
10         "tags": [
11           "CategoriesApi"
12         ],
13         "responses": {
14           "200": {
15             "description": "Success"
16           }
17         }
18       },
19       "post": {
20         "tags": [
21           "CategoriesApi"
```

Pagina de vizualizare Swagger din browser:







După obținerea fișierului Swagger, am executat pașii de utilizare RESTler, menționați mai sus:

### 1) Generarea gramaticii de test

```
bash

dotnet restler\Restler.dll compile --api_spec
"C:\Users\Miru\Desktop\TSS\swagger1.json"
```

Ce face:

- Preia fișierul `swagger1.json` (definiția API-ului)
- Generează un set de fișiere în folderul `Compile\`:
  - **`grammar.py`**: modelul de cereri (requests) pe care le poate face RESTler
  - **`dict.json`**: un dicționar cu valori implicite pentru câmpuri
  - **`engine\_settings.json`**: setări pentru engine-ul de testare

```
C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin>dotnet restler\Restler.dll compile --api_spec "C:\Users\Miru\Desktop\TSS\swagger3.json"
Starting task Compile...
Task Compile succeeded.
Collecting logs...
```

## 2) Task Collection (`test`):

```
bash

dotnet
"C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\restler\Restler.dll"
" test ^

--grammar_file
"C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\Compile\grammar_fix
ed.py" ^

--dictionary_file
"C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\Compile\dict.json"
^

--host "localhost" ^

--target_port 7121
```

Ce face:

- Rulează cereri valide (conforme cu Swagger) pentru a identifica:
  - dacă endpoint-urile răspund corect
  - ce combinații de input-uri funcționează
- Creează un „baseline” de funcționare: RESTler învață care cereri sunt valide.

```
C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin>dotnet "C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\restler\Restler.dll" test ^
More? --grammar_file "C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\Compile\grammar_fixed.py" ^
More? --dictionary_file "C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\Compile\dict.json" ^
More? --host "localhost" ^
More? --target_port 7121
Starting task Test...
Using python: 'python.exe' (Python 3.13.1)
Request coverage (successful / total): 5 / 5
Attempted requests: 5 / 5
No bugs were found.
Task Test succeeded.
Collecting logs...
```

## 3) Fuzzing (`fuzz`)

```
bash

dotnet
"C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\restler\Restler.dll"
" fuzz ^
```

```
--grammar_file
"C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\Compile\grammar_fixed.py" ^

--dictionary_file
"C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\Compile\dict.json"
^

--host "localhost" ^

--target_port 7121
```

Ce face:

- Rulează testele generate automat, dar:
  - Include mutații (valori invalide, lipsă, excesive)
  - Testează comportamentul API-ului la cereri **neobișnuite** sau malformate
- Scopul este să detecteze **bug-uri, crash-uri, excepții necontrolate sau cod HTTP neașteptat (500, 403 etc.)**
- Stochează rezultatele în folderul `RestlerResults/<data>`:
  - `bug\_buckets.txt`: dacă a găsit bug-uri
  - `network.testing.log`: log complet al cererilor
  - `fuzzing\_summary.md`: rezumat

```
C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin>dotnet "C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\restler\Restler.dll" fuzz ^
More? --grammar_file "C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\Compile\grammar_fixed.py" ^
More? --dictionary_file "C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\Compile\dict.json" ^
More? --host "localhost" ^
More? --target_port 7121 ^
More? --time_budget 0.11
Starting task Fuzz...
Using python: 'python.exe' (Python 3.13.1)

ERROR: Results analyzer for logs in C:\Users\Miru\Desktop\TSS\restler-fuzzer\restler_bin\Fuzz failed.

Request coverage (successful / total): 5 / 5
Attempted requests: 5 / 5
No bugs were found.
Task Fuzz succeeded.
Collecting logs...
```

## c. Implementarea celui de-al doilea modul – VulnRISKatcher

Așa cum reiese din articol, **VulnRISKatcher** este un instrument inovator pentru identificarea și clasificarea vulnerabilităților în codul sursă, utilizând tehnici avansate de Machine Learning. Spre deosebire de instrumentele tradiționale de revizuire a codului care folosesc abordări statice, VulnRISKatcher oferă o alternativă mai eficientă care nu necesită înțelegerea completă a contextului codului.

Caracteristici principale:

**1. Utilizarea tehnicilor ML pentru verificarea codului** - Permite analiza codului fără a necesita contextul complet, făcând instrumentul aplicabil la diferite niveluri de testare.

**2. Suport pentru diverse limbaje de programare** - Instrumentul analizează cod lexical din multiple limbaje de programare, crescând versatilitatea soluției.

**3. Pipeline de procesare structurat** - Procesul include:

- Furnizarea codului și specificarea limbajului de programare
- Preprocesarea datelor (curățarea și divizarea în unități discrete)
- Analiza pentru identificarea tiparelor asociate vulnerabilităților
- Clasificarea vulnerabilităților identificate
- Generarea unui raport detaliat pentru utilizator

**4. Identificarea și clasificarea riscurilor** - Pe lângă detectarea vulnerabilităților, instrumentul evaluează și riscurile asociate acestora.

VulnRISKatcher reprezintă un pas important în evoluția instrumentelor de testare software, oferind o soluție mai rapidă și mai precisă pentru identificarea vulnerabilităților, contribuind astfel la îmbunătățirea calității software-ului și la reducerea costurilor de dezvoltare și mentenanță.

## Proof of concept

Pentru a ne apropia în practică de conceptul teoretic de VulnRISKatcher am decis să încercăm să folosim un model de Machine Learning, întrucât învățarea automată este una dintre caracteristicile care stau la baza conceptului.

Am ales să folosim modele de pe HuggingFace și astfel am găsit modelul *microsoft/codebert-base*, care însă ar fi avut nevoie de fine-tuning, așa că am încercat să găsim un model deja fine-tunat. Acest lucru s-a dovedit a fi un task destul de complicat, dar în urma căutărilor am ajuns la două variante:

1. *mahdin70/CodeBERT-VulnCWE* - antrenat pe un set de date curatoriat și îmbogățit pentru detectarea vulnerabilităților și clasificarea CWE. Modelul poate prezice dacă un fragment de cod este vulnerabil și, în caz afirmativ, poate identifica ID-ul CWE specific asociat. (părea o variantă promițătoare, dar am avut probleme cu importarea acestuia, deci am fost nevoiți să renunțăm)

2. *mrm8488/codebert-base-finetuned-detect-insecure-code* [8] - Modelul analizează secvențe de cod sursă pentru a determina dacă acestea conțin vulnerabilități de securitate (precum scurgeri de resurse sau atacuri DoS), clasificându-le binar ca sigure (0) sau nesigure (1). (deși ne oferă doar o clasificare binară a erorilor, acest model s-a dovedit a fi o variantă mai sigură și cu care am reușit să lucrăm)

Deși modelul oferă doar o clasificare binară (cod sigur vs. cod nesigur), l-am integrat într-o analiză mai nuanțată. Pentru a face analiza mai detaliată, am implementat:

- Analiza globală a codului pentru o evaluare generală a întregului fișier
- Analiza pe secțiuni consecutive de cod (denumită "ferestre glisante" sau "sliding windows") - aceasta împarte codul în bucăți mai mici de câte 15 linii, care se suprapun parțial (cu 5 linii), pentru a identifica mai precis zonele problematice din cod

Tot pentru a compensa limitarea clasificării binare a modelului ML, am implementat și un sistem extins de reguli bazat pe expresii regulate, care poate identifica zece categorii specifice de vulnerabilități:

### SQL Injection

Detectează construirea nesigură a interogărilor SQL prin concatenare de șiruri

```
"patterns": [  
    r"string\s+sql\s*=.*\s+",  
    r"executeQuery\(.*\s+",  
    # ... alte modele  
]
```

*Python*

**Cross-Site Scripting (XSS)** - Identifică manipularea nesigură a DOM-ului și execuția de cod dinamic

```
"patterns": [  
    r"string\s+sql\s*=.*\s+",  
    r"executeQuery\(.*\s+",  
    # ... alte modele  
]
```

```
r"innerHTML\s*=",  
  
r"document\.write\(",  
  
# ... alte modele  
  
]
```

**Probleme de autentificare** - Găsește credențiale hardcodate și configurări nesigure de autentificare

```
"patterns": [  
  
r"password\s*=\s*\"[^"]]+\\"",  
  
r"hardcoded.*password",  
  
# ... alte modele  
  
]
```

**Bypass-uri de autorizare** - Detectează modificări nesigure ale rolurilor și permisiunilor

```
"patterns": [  
  
r"\.Authorize\(.*false\)",  
  
r"isAdmin\s*=\s*true",  
  
# ... alte modele  
  
]
```

**Referințe directe nesigure la obiecte** - Identifică accesarea directă a obiectelor fără verificare

```
"patterns": [  
    r"Request\.QueryString\[\"id\\\"\\",  
    r"Request\.Params\[\"id\\\"\\",  
    # ... alte modele  
]
```

**Configurări greșite de securitate** - Găsește setări de debug active sau configurări HTTPS/SSL dezactivate

```
"patterns": [  
    r"debug\s*=\s*true",  
    r"IsDebug\s*=\s*true",  
    # ... alte modele  
]
```

**Expunere de date sensibile** - Detectează utilizarea algoritmilor criptografici slabi sau manipularea datelor sensibile

```
"patterns": [  
    r"\.CreateEncryptor\(",  
    r"MD5\.",  
    # ... alte modele  
]
```

**Lipsă control acces la nivel de funcție** - Identifică potențiale probleme în verificarea accesului

```
"patterns": [  
    r"\.IsAdmin\(\)",  
    r"UserManager\.",  
    # ... alte modele  
]
```

**Cross-Site Request Forgery (CSRF)** - Găsește endpoint-uri vulnerabile la CSRF

```
"patterns": [  
    r"\[HttpPost\](?!.*(?:\[ValidateAntiForgeryToken\]|\[AutoValidateAntiforg  
    eryToken\]))",  
    # ... alte modele  
]
```

**Utilizarea componentelor cu vulnerabilități cunoscute** - Detectează versiuni vechi și vulnerabile ale bibliotecilor comune

```
"patterns": [  
    r"jquery.{0,10}[\\"']1\.[0-9]\.[0-9][\\"']",  
    r"bootstrap.{0,10}[\\"']2\.[0-9]\.[0-9][\\"']",  
    # ... alte modele  
]
```



```
]
```

Fiecare vulnerabilitate este evaluată pe o scară de risc cu 5 niveluri:

```
RISK_LEVELS = {  
    0: "Sigur",  
    1: "Scăzut",  
    2: "Mediu",  
    3: "Ridicat",  
    4: "Critic"  
}
```

## Integrarea API-ului Flask cu aplicația .NET

API-ul de analiză a vulnerabilităților a fost implementat ca un serviciu independent în Flask, care comunică cu aplicația principală .NET prin intermediul cererilor HTTP. Această arhitectură cu microservicii oferă flexibilitate și permite evoluția independentă a celor două componente.

API-ul Flask oferă un endpoint REST (/analyze) care:

- Primește cod sursă și (opțional) limbajul de programare
- Aplică analiza bazată pe reguli
- Aplică analiza bazată pe ML (dacă modelul este disponibil)
- Combină rezultatele, eliminând duplicatele și ajustând nivelurile de încredere
- Generează un raport detaliat cu vulnerabilitățile identificate, nivelurile de risc și recomandări pentru remediere

## Structura aplicației .NET pentru scanarea de vulnerabilități

Pentru a integra API-ul Flask în aplicația .NET, am implementat o structură modulară cu următoarele componente principale:

```
|— Controllers/
|   |— ProjectScanController.cs
|
|— Services/
|   |— Vulnerability/
|       |— IVulnerabilityService.cs
|       |— VulnerabilityService.cs
|       |— VulnerabilityAnalysisResult.cs
|       |
|       |— CodeScanner/
|           |— ICodeScannerService.cs
|           |— CodeScannerService.cs
|           |— FileVulnerabilityReport.cs
|           |— ProjectVulnerabilityReport.cs
|           |
|— Views/
|   |— ProjectScan/
|       |— Index.cshtml
|       |— ScanResults.cshtml
|       |— FileScanResult.cshtml
|
|— appsettings.json (configurare pentru API-ul de vulnerabilități)
```

## IVulnerabilityService & VulnerabilityService

Serviciul de vulnerabilități din .NET acționează ca un client pentru API-ul Flask. Acesta gestionează comunicarea HTTP și transformarea datelor între cele două sisteme.

## CodeScannerService

Serviciul de scanare a codului este responsabil pentru:

- Identificarea fișierelor care trebuie scanate în proiect
- Trimiterea fiecărui fișier către VulnerabilityService
- Agregarea rezultatelor într-un raport complet al proiectului

## Controllerul și vizualizările

Controllerul ProjectScanController expune funcționalitatea de scanare a vulnerabilităților în interfața web, permițând utilizatorilor să inițieze scanări și să vizualizeze rezultatele într-un format prietenos.

## Fluxul de comunicare între componente

Procesul complet de analiză a vulnerabilităților funcționează astfel:

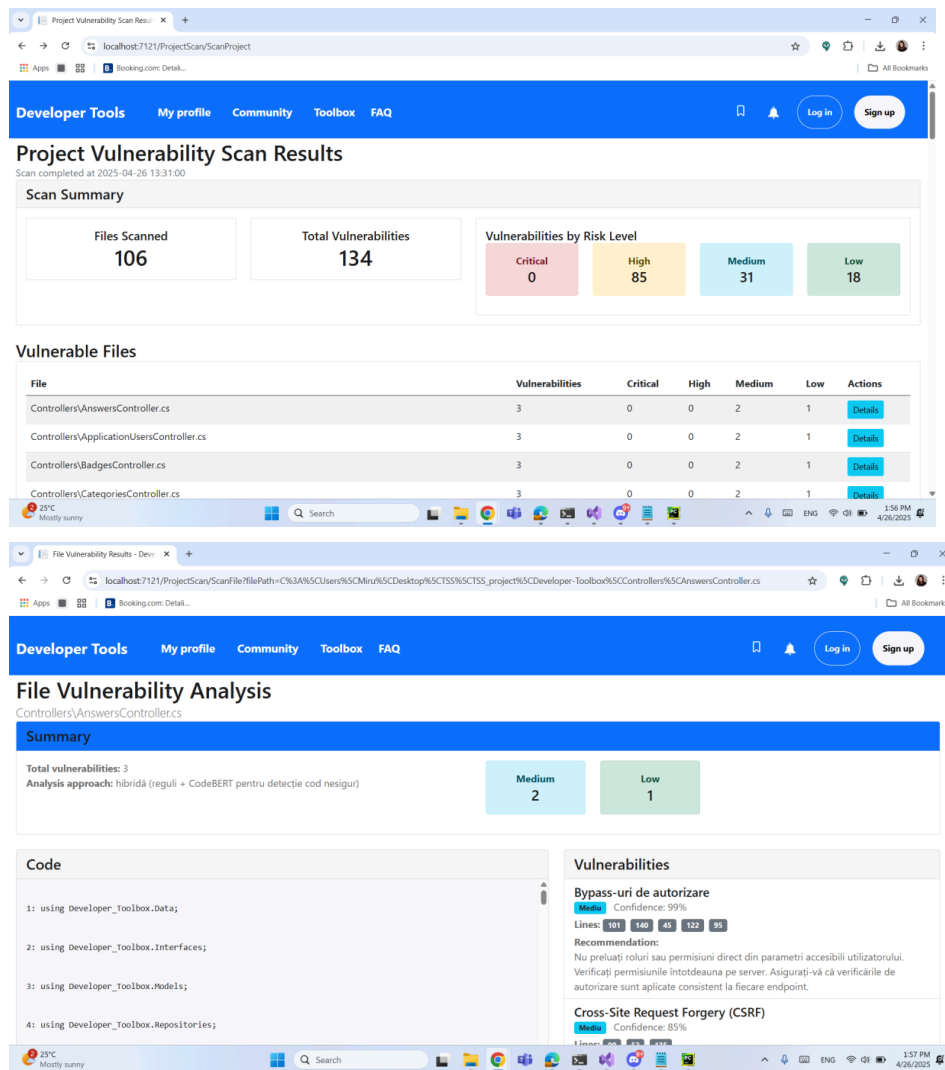
1. Utilizatorul accesează interfața web și inițiază o scanare de proiect
2. Controllerul delegă scanarea către CodeScannerService
3. CodeScannerService identifică fișierele relevante și le trimite pe rând către VulnerabilityService
4. VulnerabilityService face cereri POST către API-ul Flask cu conținutul fiecărui fișier
5. API-ul Flask analizează codul utilizând combinația de reguli și modelul ML
6. Rezultatele analizei sunt trimise înapoi la VulnerabilityService
7. CodeScannerService agregază rezultatele pentru toate fișierele
8. Controllerul prezintă rezultatele utilizatorului prin intermediul vizualizărilor

Această arhitectură modulară permite integrarea eficientă între aplicația .NET și API-ul Flask de analiză a vulnerabilităților, combinând avantajele ambelor tehnologii pentru a oferi o soluție completă de scanare a vulnerabilităților în codul sursă.

## Rezultatele analizei

Raportul generat include:

- Numărul total de vulnerabilități identificate
- Lista detaliată a vulnerabilităților cu:
  - Tipul vulnerabilității
  - Nivelul de încredere (confidence)
  - Nivelul de risc
  - Numerele liniilor afectate
  - Recomandări specifice pentru remediere
- Un rezumat statistic al vulnerabilităților pe niveluri de risc
- Mențiunea tipului de analiză utilizat (hibridă sau doar bazată pe reguli)



## Integrarea cu conceptul teoretic VulnRISKatcher

Implementarea noastră respectă principiile VulnRISKatcher prezentate în articolul științific:

1. **Utilizarea tehnicilor ML pentru verificarea codului** - Implementat prin integrarea modelului
2. **Suport pentru diverse limbaje de programare** - API-ul acceptă specificarea limbajului (deși regulile sunt optimizate pentru C#/.NET)
3. **Pipeline de procesare structurat:**
  - Furnizarea codului și limbajului de programare - prin endpoint-ul API
  - Preprocesarea datelor - implementată prin divizarea codului în linii și segmente

- Analiza pentru identificarea tiparelor - realizată prin reguli și ML
- Clasificarea vulnerabilităților - prin sistemul de tipuri și niveluri de risc
- Generarea unui raport detaliat - în format JSON structurat

4. **Identificarea și clasificarea riscurilor** - Realizată prin sistemul de evaluare cu 5 niveluri

## Îmbunătățiri viitoare

Pentru a extinde capabilitățile instrumentului, putem:

1. Antrena modele mai specializate pentru diferite limbaje de programare
2. Implementa detectarea bazată pe grafuri de dependențe pentru vulnerabilități mai complexe
3. Adăuga analiza fluxului de date pentru identificarea mai precisă a vulnerabilităților
4. Extinde baza de cunoștințe cu reguli pentru mai multe tipuri de vulnerabilități
5. Integra cu sisteme de CI/CD pentru verificarea automată a securității

În această implementare, am combinat cu succes abordarea bazată pe reguli cu tehnici de Machine Learning, reușind să construim un instrument practic care ilustrează conceptul teoretic de VulnRISKatcher prezentat în articolul științific.

## d. Implementarea celui de-al treilea modul – CodeAssert [10], [11]

### Scopul Modulului

Modulul CodeAssert, parte a framework-ului **TestLab**, vizează **generarea automată de teste unitare pentru clase model** pe baza analizei codului sursă (white-box testing). Acest PoC automatizează:

- extragerea de proprietăți din modele,
- detecția atributelor de validare,
- generarea fișierelor `.cs`` cu teste unitare folosind xUnit.

## Fluxul Fișierelor și Componentelor

### 1. `Program.cs` – *Coordonatorul*

- Definește directoarele sursă și output.
- Filtrează modelele relevante ignorând unele entități specifice.
- Pentru fiecare model:
  - extrage namespace-ul și proprietățile
  - generează fișierul de test corespunzător

Exemplu Output: `BadgeModel.cs` -> [BadgeModelUnitTest.cs](#)

### 2. `ModelAnalyzer.cs` – *Extractorul de Proprietati*

- Identifică proprietățile unei clase.
- Recunoaște colecții (`List<T>`, `IEnumerable<T>`, etc.).
- Generează valori implicite pentru fiecare tip.
- Detectează attribute de validare (`[Required]`, `[StringLength]`, etc.).

### 3. `CodeAnalysisService.cs` – *Namespace & Condiții*

- Extragerea namespace-ului.
- Extragerea condițiilor.

### 4. `TestTemplateGenerator.cs` – *Generatorul de Cod de Test*

- Creează clase `xUnit` cu:
  - test pentru constructor,
  - test pentru inițializarea proprietăților,
  - test pentru validări (`Required`, `StringLength`, etc.).

**Exemplu de test generat:**

```
[Fact]
public void Constructor_Test()
{
```

*csharp*

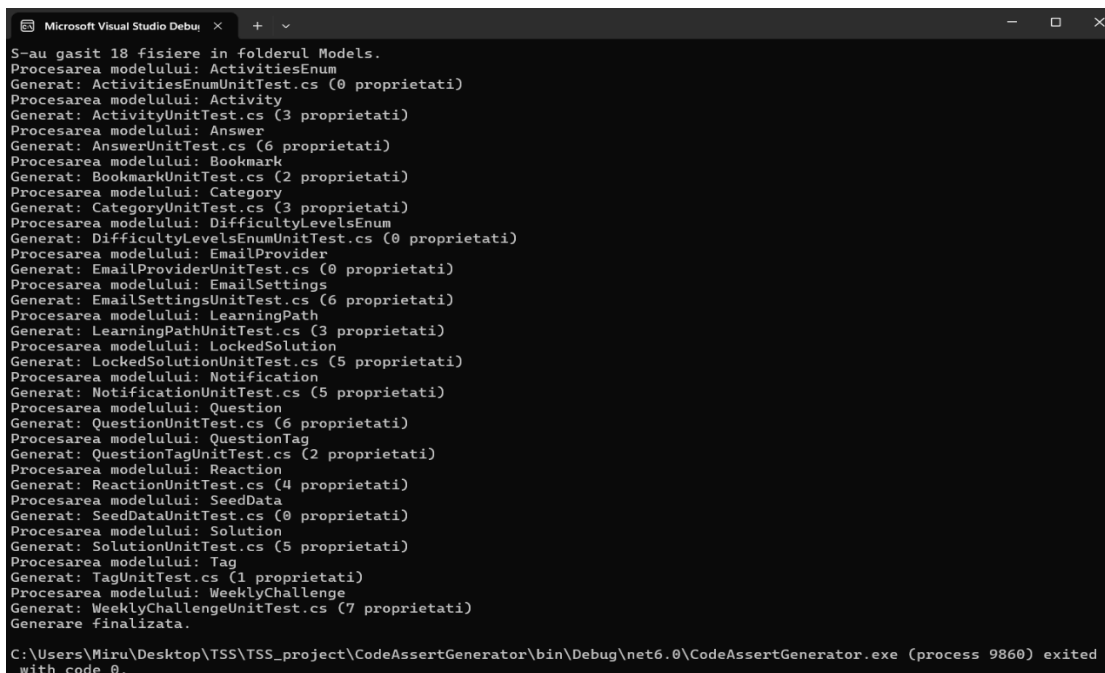
```
var user = new User();

Assert.NotNull(user);

Assert.IsType<User>(user);

}
```

Procesarea modelelor si generarea de teste, folosind `TestTemplateGenerator.cs`



```
Microsoft Visual Studio Debug
S-au gasit 18 fisiere in folderul Models.
Procesarea modelului: ActivitiesEnum
Generat: ActivitiesEnumUnitTest.cs (0 proprietati)
Procesarea modelului: Activity
Generat: ActivityUnitTest.cs (3 proprietati)
Procesarea modelului: Answer
Generat: AnswerUnitTest.cs (6 proprietati)
Procesarea modelului: Bookmark
Generat: BookmarkUnitTest.cs (2 proprietati)
Procesarea modelului: Category
Generat: CategoryUnitTest.cs (3 proprietati)
Procesarea modelului: DifficultyLevelsEnum
Generat: DifficultyLevelsEnumUnitTest.cs (0 proprietati)
Procesarea modelului: EmailProvider
Generat: EmailProviderUnitTest.cs (0 proprietati)
Procesarea modelului: EmailSettings
Generat: EmailSettingsUnitTest.cs (6 proprietati)
Procesarea modelului: LearningPath
Generat: LearningPathUnitTest.cs (3 proprietati)
Procesarea modelului: LockedSolution
Generat: LockedSolutionUnitTest.cs (5 proprietati)
Procesarea modelului: Notification
Generat: NotificationUnitTest.cs (5 proprietati)
Procesarea modelului: Question
Generat: QuestionUnitTest.cs (6 proprietati)
Procesarea modelului: QuestionTag
Generat: QuestionTagUnitTest.cs (2 proprietati)
Procesarea modelului: Reaction
Generat: ReactionUnitTest.cs (4 proprietati)
Procesarea modelului: SeedData
Generat: SeedDataUnitTest.cs (0 proprietati)
Procesarea modelului: Solution
Generat: SolutionUnitTest.cs (5 proprietati)
Procesarea modelului: Tag
Generat: TagUnitTest.cs (1 proprietati)
Procesarea modelului: WeeklyChallenge
Generat: WeeklyChallengeUnitTest.cs (7 proprietati)
Generare finalizata.
C:\Users\Miru\Desktop\TSS\TSS_project\CodeAssertGenerator\bin\Debug\net6.0\CodeAssertGenerator.exe (process 9860) exited
with code 0.
```

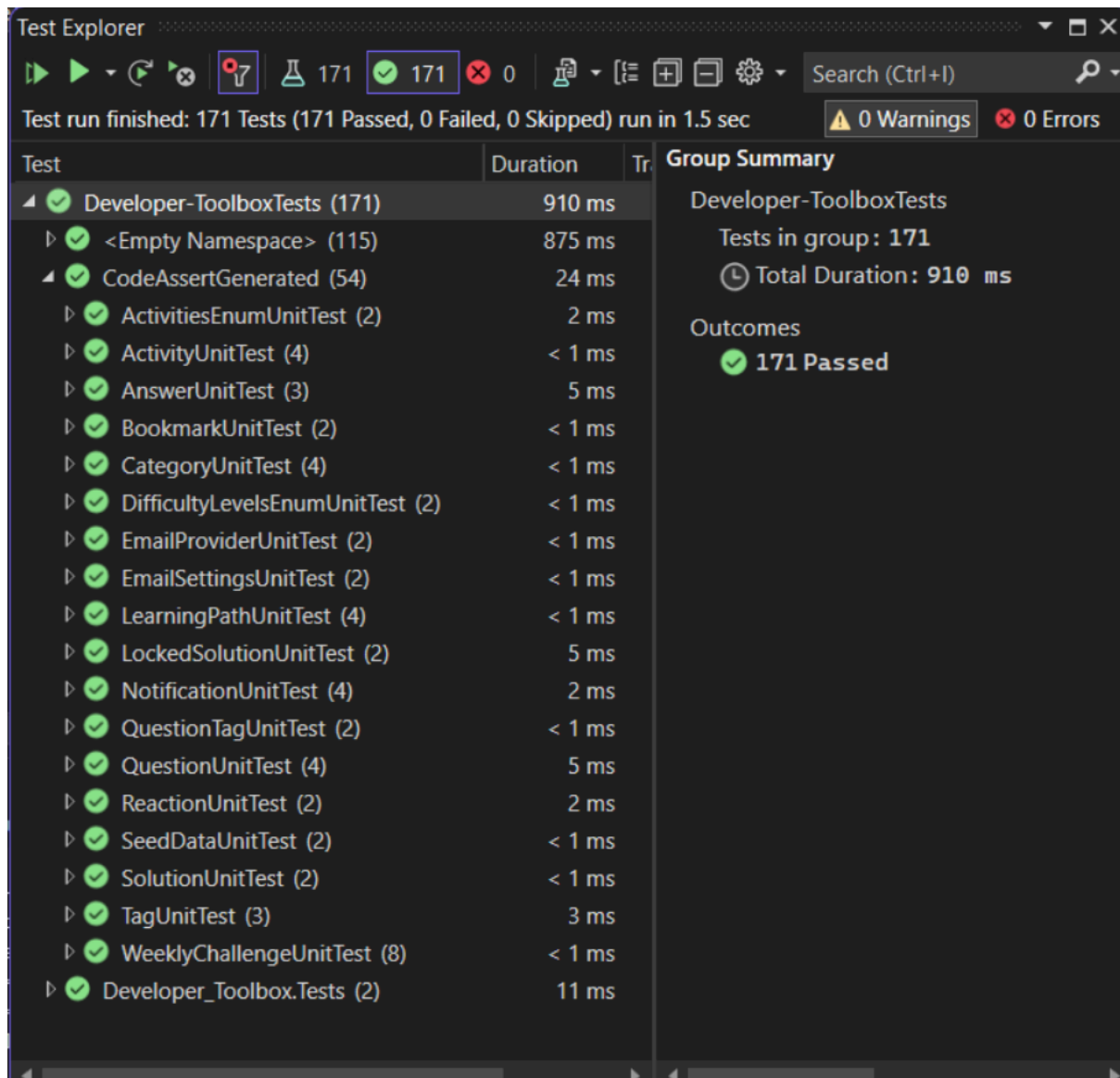
## Rezultate

După generarea automată a testelor, acestea sunt rulate pentru a valida integritatea procesului. Testele se execută rapid și confirmă acoperirea modelelor și a validărilor aferente.

### Toate testele au trecut cu succes

Imaginea de mai jos arată execuția a 171 de teste. Toate au trecut, fără erori sau skip-uri:





## Ce respecta implementarea modului, față de articol

- Automatizare completă pentru testele de bază.
- Ignorare fișiere irelevante -> reduce zgomotul.
- Tratament special pentru colecții.
- Suport pentru atribute de validare (inclusiv verificarea cu reflecție).
- Se integrează ușor în CI pentru generare continuă de teste.

## Îmbunătățiri Viitoare ale modului

1. Testarea pentru tipuri complexe
2. Integrarea cu Expected Values AI/NLP
3. Suport pentru alte frameworkuri de test (JUnit, MSTest)

## 5. Concluzii

TestLab propune o abordare integrată, inteligentă și automatizată pentru testarea software:

- Testare continuă
- Acoperire completă
- Detectarea timpurie a erorilor
- Automatizare extinsă → reducerea efortului uman

Prin realizarea acestui proiect, am conștientizat atât **importanța esențială a testării sistemelor software**, cât și **complexitatea reală a implementării ei eficiente în practică**.

Un instrument avansat, precum cel propus în articolul studiat, ar putea aduce un aport semnificativ în creșterea calității aplicațiilor. Totuși, **pe lângă eficiența sa tehnică, un aspect esențial îl constituie ușurința în utilizare** – astfel de unelte trebuie să fie **accesibile tuturor dezvoltatorilor**, indiferent de nivelul lor de experiență.

Deși am implementat modulele într-o formă simplificată, cu scop demonstrativ, **utilizarea lor concretă în cadrul aplicației ne-a evidențiat relevanța și necesitatea acestor mecanisme** în procesul de testare automatizată.

## 6. Anexe

În GitHub-ul aferent proiectului [7] avem următoarele documente încărcate care sunt completări/ anexe ale documentației realizare:

- ❖ Prezentare TestLab
- ❖ Demo [9]
- ❖ Raport despre folosirea unui tool de AI [5],[6]

## 7. Referințe

[1] TestLab: An Intelligent Automated Software Testing Framework, Tiago Dias, Arthur Batista, Eva Maia, Isabel Praça , <https://arxiv.org/abs/2306.03602>, Ultima accesare: 16 mai 2025

[2] „The prevalence of software systems has become an integral part of modern-day living.” / „Software usage has increased significantly, leading to its growth in both size and complexity.” / „Consequently, software development is becoming a more time-consuming process.” – din articolul TestLab

[3] „The integration of Artificial Intelligence (AI) in the software testing process is promising...” / „...no prior work has addressed the development of a comprehensive framework comprising multiple testing methods...” – din articolul TestLab

[4] Microsoft RESTler: Stateful REST API Fuzzing Tool,  
<https://www.microsoft.com/en-us/research/publication/restler-stateful-rest-api-fuzzing/>,  
Accesat: 15 mai 2025

[5] OpenAI, Chat GPT, <https://chatgpt.com/>, Data generării: 11 aprilie 2025

[6] Anthropic, Claude 3.7 Sonnet, <https://claude.ai/>, Data generării: 25 aprilie 2025

[7] Link GitHub proiect “TestLab: An Intelligent Automated Software Testing Framework”,  
[https://github.com/stoineamiruna/TSS\\_project?tab=readme-ov-file](https://github.com/stoineamiruna/TSS_project?tab=readme-ov-file)

[8] codebert-base-finetuned-detect-insecure-code,  
<https://huggingface.co/mrm8488/codebert-base-finetuned-detect-insecure-code>, Data ultimei  
accesari: 25 aprilie 2025

[9] Demo implementare proiect “TestLab: An Intelligent Automated Software Testing  
Framework”, <https://youtu.be/ppFEI5dOFXI>

[10] Roslyn: .NET Compiler Platform Microsoft. Roslyn - .NET Compiler Platform SDK.  
<https://github.com/dotnet/roslyn> , Data ultimei accesari: 12 mai 2025

[11] xUnit Testing Framework Brad Wilson et al. xUnit.net Documentation.  
<https://xunit.net/>, Data ultimei accesari: 12 mai 2025