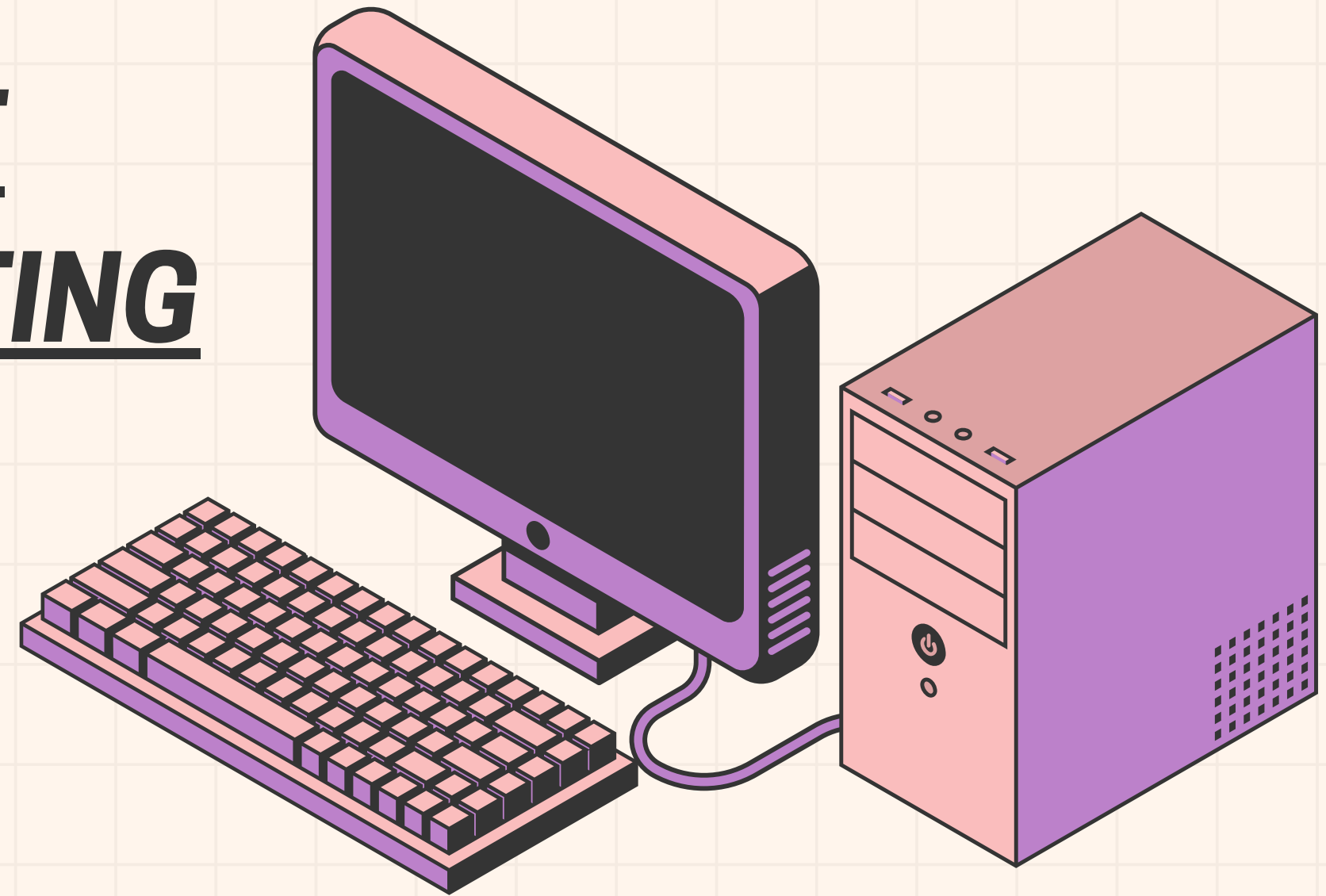


CONCEPTE ȘI METODE DIN ARTICOLUL TESTLAB: AN INTELLIGENT AUTOMATED SOFTWARE TESTING FRAMEWORK

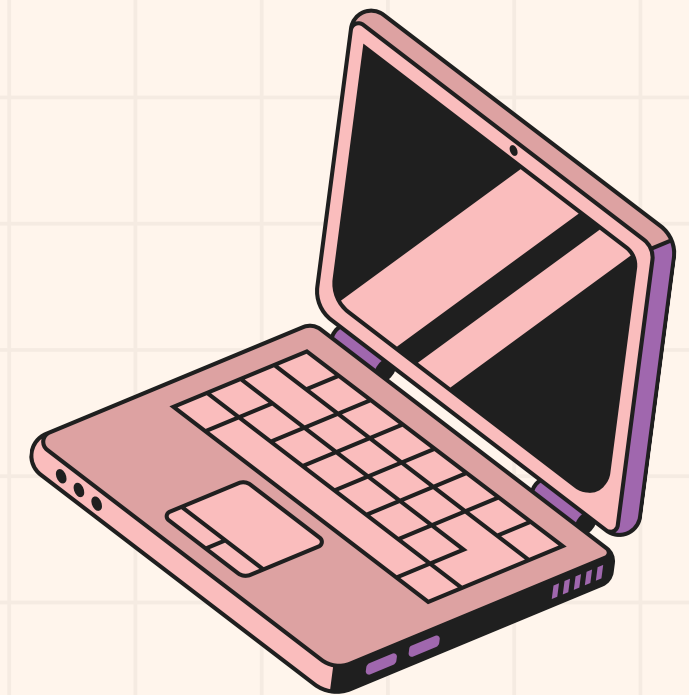
Antonescu Ionuț Andrei
Ciurescu Irina Alexandra
Nazare Elena
Stoinea Maria Miruna



INTRODUCERE

Într-o lume în care aplicațiile software devin tot mai complexe și omniprezente, calitatea codului nu mai este un lux, ci o necesitate. Cu mii de linii de cod, module interconectate și cerințe în continuă schimbare, testarea software a devenit o provocare majoră. Din dorința de a livra rapid, multe echipe reduc din timpul dedicat testării, ceea ce duce la erori costisitoare și vulnerabilități.

TestLab este soluția propusă pentru această problemă — un cadru automatizat, inteligent, care combină metode moderne de testare cu tehnici de Inteligență Artificială. Scopul este asigurarea unei testări eficiente și continue, pe toată durata dezvoltării aplicației și la toate nivelurile: de la testare unitară, până la validarea finală de către utilizator.



TestLab este compus din trei module principale:

FUZZTHEREST

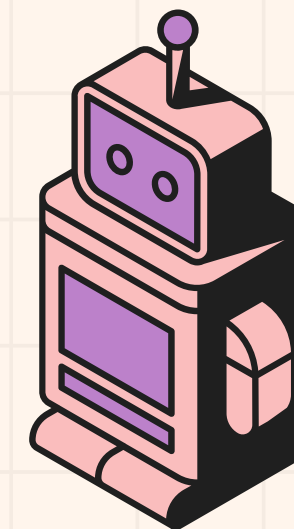
- Fuzzer inteligent pentru API-uri REST, bazat pe Reinforcement Learning
- Funcționează în mod black-box, generând input-uri deformate și folosind feedback-ul API-ului
- Poate fi extins la grey-box prin analiza execuției codului

VULNRISKATCHER

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

CODEASSERT

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



FUZZTHEREST

RESTler este un instrument dezvoltat de Microsoft pentru testarea automată a API-urilor de tip REST. Spre deosebire de alte tool-uri de testare, RESTler folosește o abordare inteligentă, bazată pe învățare prin întărire (RL), pentru a genera automat cereri (request-uri) către un API.

Pentru a funcționa, RESTler are nevoie de un fișier OpenAPI (numit adesea swagger.json) care descrie toate endpoint-urile disponibile în API. Pe baza acestei descrieri, tool-ul poate genera automat cereri valide, iar mai apoi poate începe să introducă variații și mutații în cereri pentru a observa cum răspunde sistemul testat.

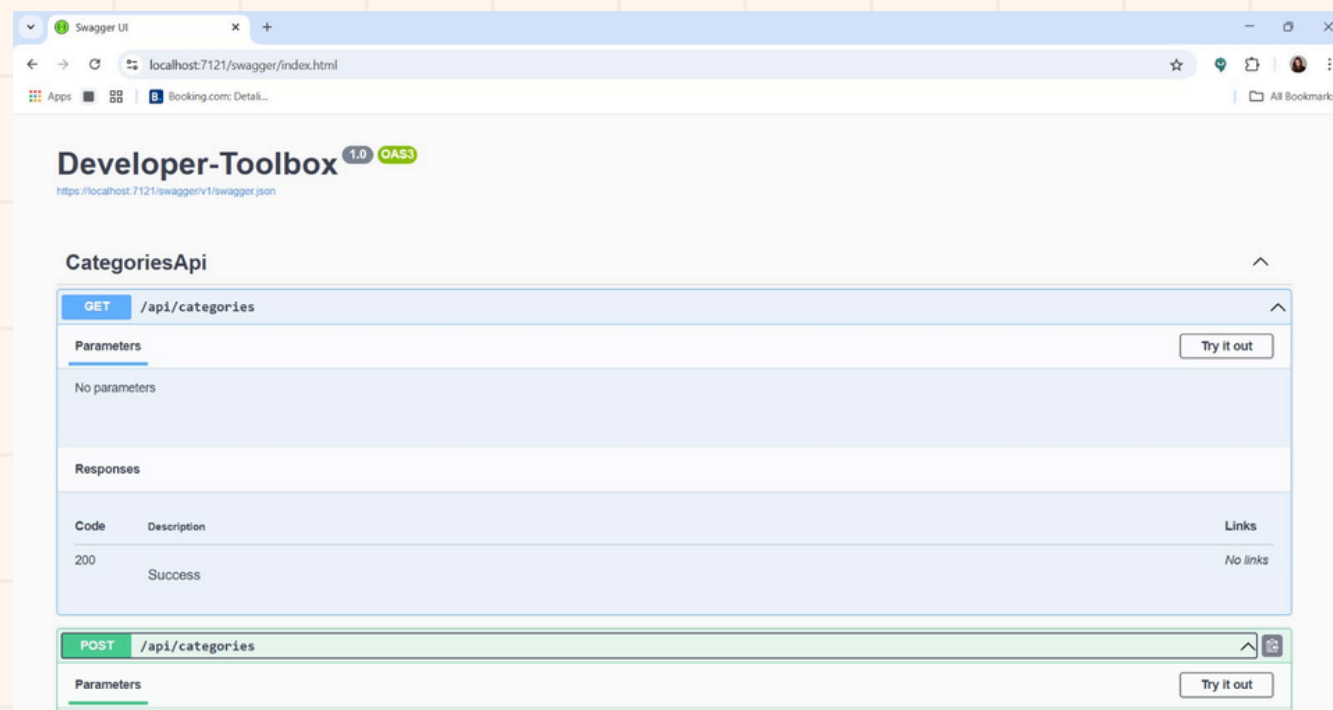
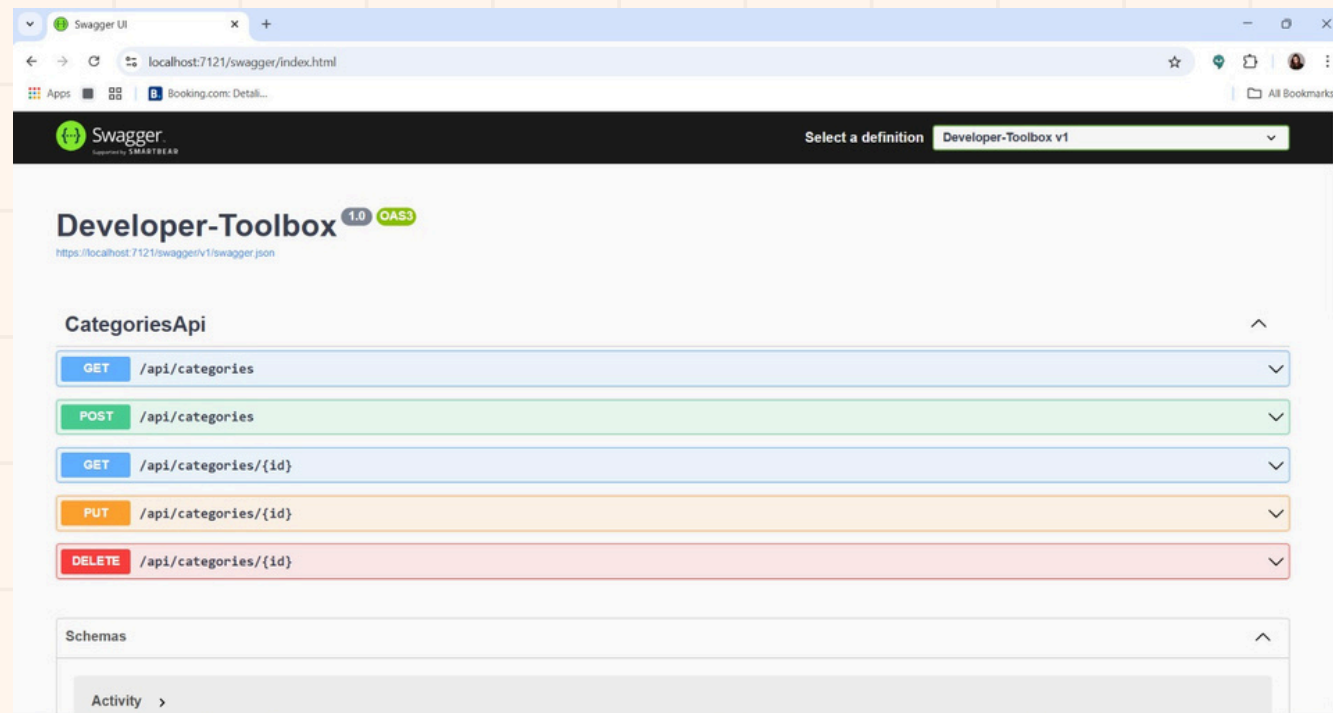
swagger.json →

```
{
  "openapi": "3.0.1",
  "info": {
    "title": "Developer-Toolbox",
    "version": "1.0"
  },
  "paths": {
    "/api/categories": {
      "get": {
        "tags": [
          "CategoriesApi"
        ],
        "responses": {
          "200": {
            "description": "Success"
          }
        }
      },
      "post": {
        "tags": [
          "CategoriesApi"
        ]
      }
    }
  }
}
```

Unul dintre aspectele cheie ale RESTler este faptul că ține cont de starea aplicației (stateful testing), învățând din răspunsurile anterioare ale API-ului pentru a construi secvențe de cereri din ce în ce mai complexe. Acest comportament îl face capabil să detecteze vulnerabilități și erori care apar doar în anumite condiții de utilizare.

FUZZTHEREST

Cum arată view-ul realizat de Swagger?



Cum folosim RESTler? Pași practici și scopul fiecăruia:

Compile:

- Se citește fișierul swagger.json și se generează automat:
 - grammar.py: definește toate cererile posibile
 - dict.json: valori implicite pentru parametri

Test:

- Se trimit cereri valide pentru a verifica dacă API-ul răspunde corect.
- Creează un „baseline” funcțional
- Confirmă corectitudinea structurii OpenAPI

Fuzz:

- Se trimit cereri modificate și deformate pentru a descoperi erori.
- Include mutații (valori lipsă, invalide, foarte mari etc.)
- Detectează crash-uri, răspunsuri HTTP anormale (ex: 500, 403)

VULNRISKATCHER

Tehnologiile și modelele pe care le-am folosit

- Am ales să utilizăm modele pre-antrenate de pe HuggingFace. Am optat pentru modelul *mrm8488/codebert-base-finetuned-detect-insecure-code*, care oferă clasificare binară a codului ca sigur (0) sau nesigur (1).

Sistemul nostru extins de reguli

- Am creat un sistem de reguli bazat pe expresii regulate pentru identificarea a zece categorii specifice de vulnerabilități: SQL Injection, Cross-Site Scripting (XSS), Probleme de autentificare, Referințe directe nesigure la obiecte, Expunere de date sensibile etc.

VulnRISKatcher este un instrument inovator care folosește Machine Learning pentru a identifica și clasifica vulnerabilitățile în codul sursă, oferind o alternativă mai eficientă față de instrumentele tradiționale de revizuire statică a codului.

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.

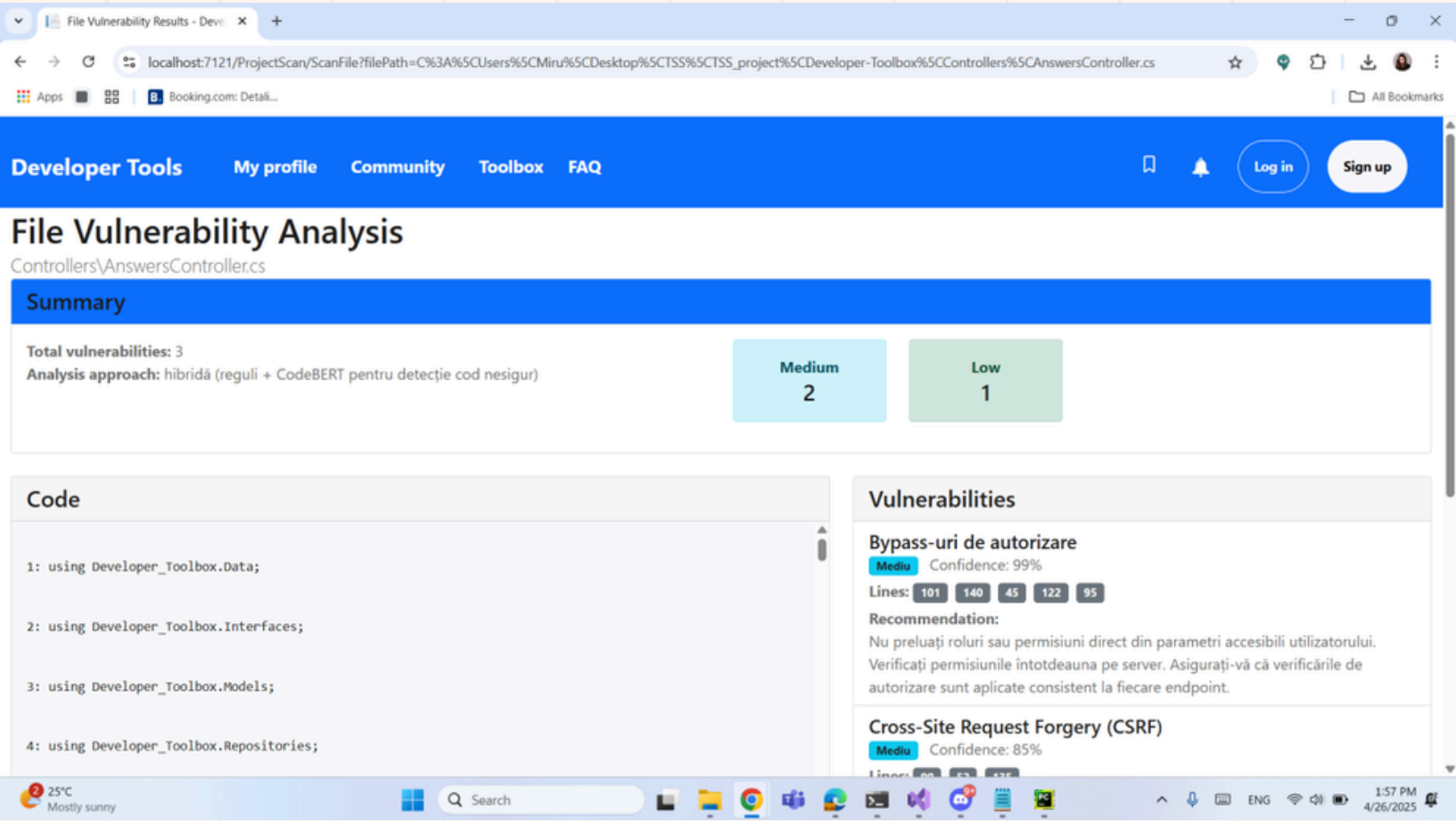
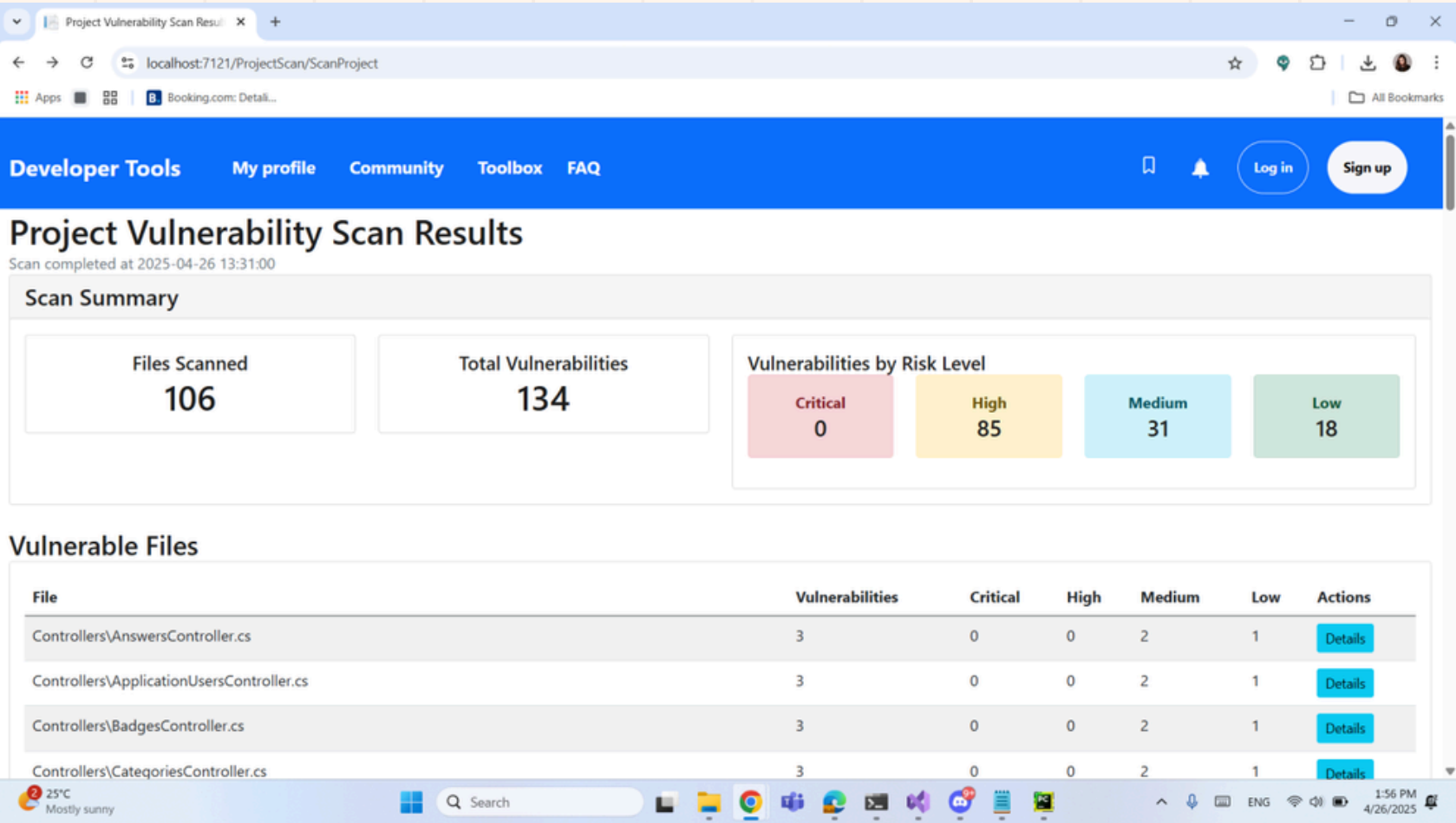
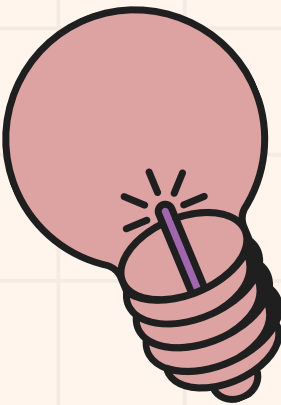
API-ul Flask oferă un endpoint REST (/analyze) care:

- 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

VULNRISKATCHER

Rezultatele analizei:

- Raport detaliat cu vulnerabilitățile identificate
- Nivel de încredere și risc pentru fiecare problemă
- Recomandări pentru remediere
- Statistici vizuale



CODEASSERT

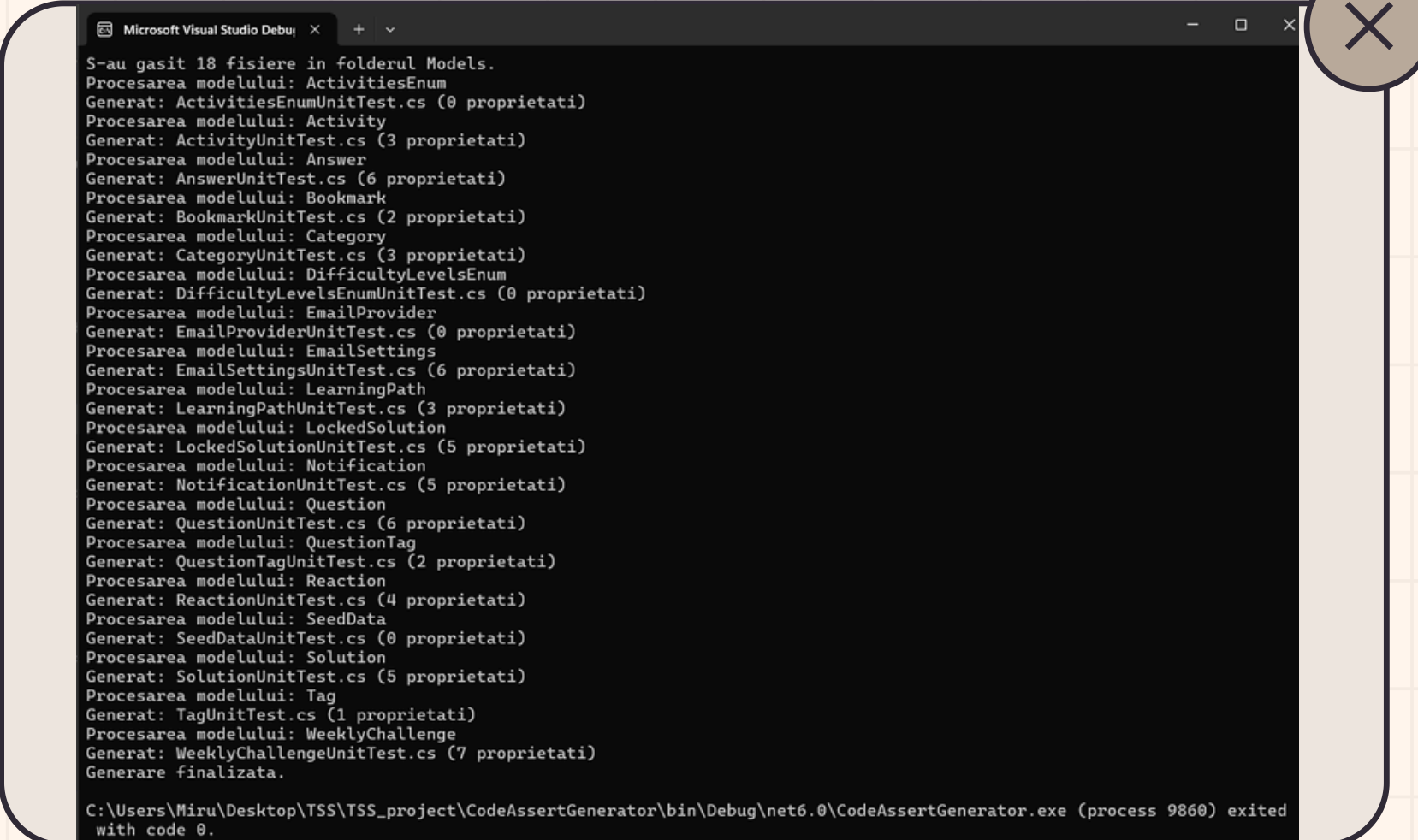
Modulul vizează generarea automată de teste unitare pentru clase model prin analiza white-box.

Funcționalități cheie:

- Extrage proprietăți & namespace-uri din modele C#
- Detectează attribute de validare ([Required], [StringLength], etc.)
- Generează teste xUnit (Constructor, Inițializare, Validări)
- Suport pentru colecții (List<T>, IEnumerable<T>)

Componente principale:

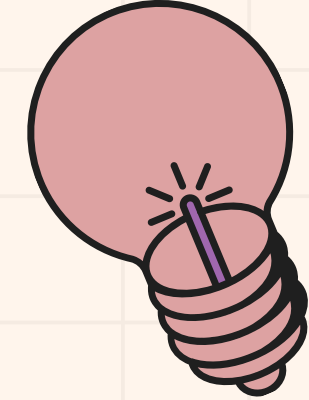
- Program.cs: Coordonator – selectează modele și declanșează procesarea
- ModelAnalyzer.cs: Identifică tipuri & attribute de validare
- CodeAnalysisService.cs: Extrage namespace & condiții
- TestTemplateGenerator.cs: Generează codul testelor xUnit



```
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.
```


CODEASSERT



Flux de procesare:

1. Selectare fișiere model
2. Analiză proprietăți & validări
3. Generare automată fișiere test .cs
 - Ex: *BadgeModel.cs* → *BadgeModelUnitTest.cs*

```
Exemplu Test Generat

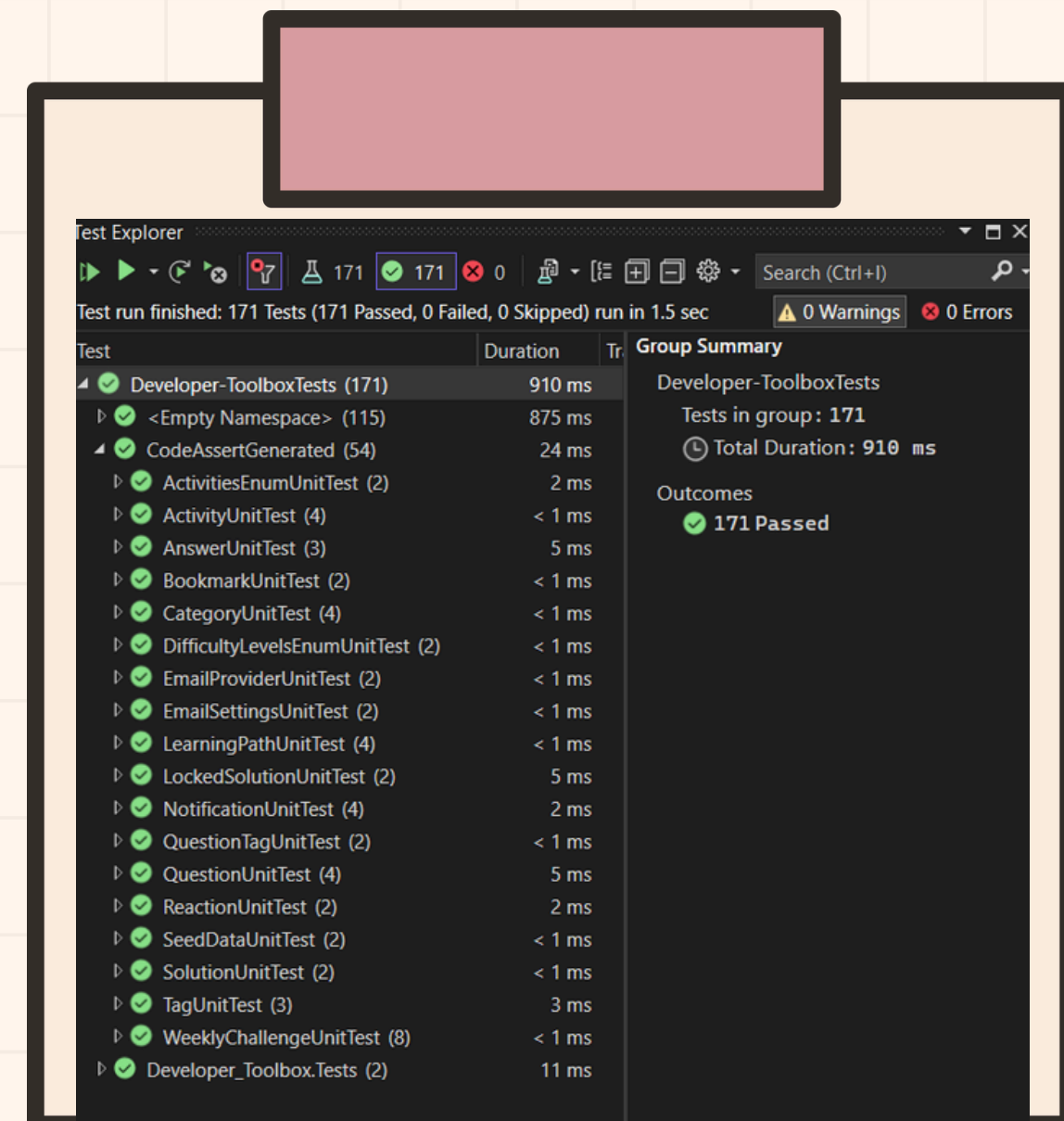
[Fact]
public void Constructor_Test() {
    var user = new User();
    Assert.NotNull(user);
    Assert.IsType<User>(user);
}
```

Avantaje:

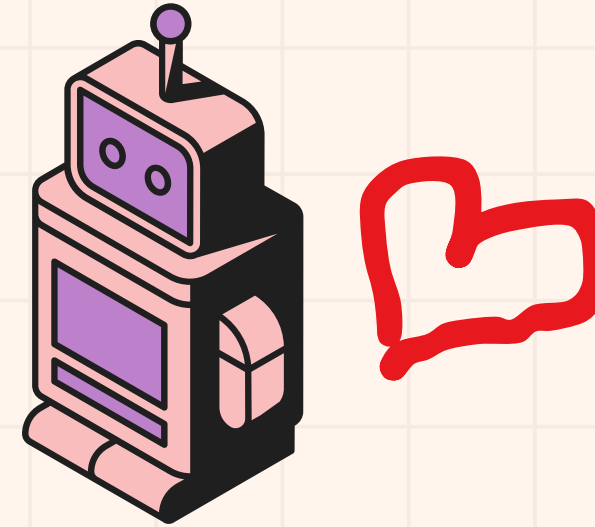
- Automatizare completă
- Reducere zgomot (filtrare fișiere nerelevante)
- Acoperire validări prin reflecție
- Eficiență crescută în asigurarea calității modelelor

Rezultate:

171 teste generate și executate cu succes



CONCLUZII



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.