# CUPRINS

[CUPRINS 1](#_Toc198062163)

[INTRODUCERE 3](#_Toc198062164)

[1 ANALIZA ȘI DEFINIREA DOMENIULUI DE STUDIU 4](#_Toc198062165)

[1.1 Analiza și cercetarea soluțiilor existente 5](#_Toc198062166)

[1.2 Definirea scopului și obiectivul proiectului 9](#_Toc198062167)

[2 PROIECTAREA ȘI ARHITECTURA SISTEMUL INFORMATIC 11](#_Toc198062168)

[2.1 Cerințele funcționale ale sistemului 11](#_Toc198062169)

[2.2 Cerințele non-funcționale ale sistemului 13](#_Toc198062170)

[2.3 Modelarea comportamentală a sistemului 14](#_Toc198062171)

[2.3.1 Generalitățile sistemului informatic 15](#_Toc198062172)

[2.3.2 Fluxul operațional al aplicației 18](#_Toc198062173)

[2.3.3 Modelarea comportamentului sistemului 20](#_Toc198062174)

[2.3.4 Descrierea fluxurilor de interacțiune utilizator–sistem 22](#_Toc198062175)

[2.3.5 Utilizarea diagramelor de comunicare pentru evidențierea fluxurilor informaționale 25](#_Toc198062176)

[2.3.6 Utilizarea diagramelor de clasă pentru modelarea componentelor sistemului 27](#_Toc198062177)

[2.3.7 Modelarea arhitecturii sistemului prin diagrame de componente 29](#_Toc198062178)

[2.3.8 Modelarea implementării aplicației 33](#_Toc198062179)

[3 ELABORAREA SISTEMULUI 35](#_Toc198062180)

[3.1 Funcționalitatea aplicației 36](#_Toc198062181)

[3.2 Detecția automată a algoritmului de hashing în HashSentinel 36](#_Toc198062182)

[3.3 Generarea automată de wordlist-uri 38](#_Toc198062183)

[BIBLIOGRAFIE 39](#_Toc198062184)

[Anexa A 41](#_Toc198062185)

# INTRODUCERE

În perioada curentă în care securitatea datelor este o preocupare majoră, strategiile de securizare a informațiilor și de recunoaștere a vulnerabilităților sunt cruciale pentru securitatea clienților și a organizațiilor. Odată cu dezvoltarea tehnologică atacurile cibernetice devin din ce în ce mai frecvente și periculoase, iar strategiile de asigurare necesită o îmbunătățire constantă. Printre tehnicile utilizate în securitatea cibernetică cea de brute force este utilizată atât în scopuri de protecție, cât și pentru testarea și analizarea vulnerabilităților din cadrele de date.

Această teză vizează investigarea în detaliu a procedurilor de brute force legate de spargerea hash-urilor, analizând viabilitatea și adecvarea acestor algoritmi în scenarii diferite. Hash-urile sunt o componentă crucială de asigurare a informațiilor, fiind utilizate pe scară largă pentru verificarea integrității fișierelor, autentificarea și protecția datelor. În cazul în care pentru hashing sunt folosiți algoritmi slabi sau dacă folosim parole compromise putem deveni victima unui astfel de atac. Prin intermediul acestei aplicații, utilizatorii vor putea testa astfel de atacuri și verifica nivelul de securitate al algoritmilor de hashing folosiți, precum și al credentialelor proprii. Pentru realizarea acestor simulări de atac, aplicația va oferi suport pentru diferiți algoritmi de hashing, printre care MD5, SHA-1 și SHA-256, unii dintre cei mai cunoscuți și care servesc drept fundație pentru alți algoritmii avansați.

Această lucrare nu se limitează doar la evidențierea riscurilor asociate atacurilor brute-force, ci își propune să ofere o soluție accesibilă și practică pentru testarea acestor tehnici. Prin dezvoltarea acestei aplicații, dorim să creăm un laborator interactiv, intuitiv și ușor de utilizat, conceput special pentru cei care nu sunt familiarizați cu aplicațiile de tip CLI (Command Line Interface). Astfel, utilizatorii vor putea explora și aplica conceptele de securitate cibernetică într-un mediu prietenos, care facilitează înțelegerea și experimentarea fără a necesita cunoștințe avansate de programare sau utilizare a terminalului.

# ANALIZA ȘI DEFINIREA DOMENIULUI DE STUDIU

În contextul securității cibernetice, hashing-ul reprezintă o funcție matematică unidirecțională care transformă datele într-un șir de caractere cu o lungime fixă, numit hash. Acest proces este ireversibil, ceea ce înseamnă că informația originală nu poate fi recuperată din hash-ul generat

Hashing-ul este utilizat în numeroase aplicații de securitate, incluzând stocarea parolelor, semnăturile digitale și verificarea integrității fișierelor și documentelor. Prin conversia conținutului în valori unice, hashing-ul asigură protecția datelor împotriva accesului neautorizat și a alterărilor accidentale sau intenționate. Un exemplu comun este utilizarea hashing-ului în autentificarea utilizatorilor, unde sistemele compară hash-ul parolei introduse cu cel stocat în baza de date pentru a permite sau bloca accesul. [1]

Pentru a obține un nivel ridicat de securitate, hashing-ul se bazează pe trei componente esențiale:

* Cheia de intrare (input key) – datele originale care urmează să fie procesate.
* Funcția hash – algoritmul matematic care convertește datele într-o valoare unică.
* Tabela de hash (hash table) – structură de date care stochează valorile hash pentru referință și căutare rapidă.

Unul dintre avantajele principale ale hashing-ului este capacitatea sa de a garanta integritatea și securitatea datelor. De exemplu, în procesul de validare a fișierelor descărcate, se creează un hash unic, numit checksum, care servește la verificarea autenticitații și integrității fișierului. Checksum-ul este o valoare numerică unică obținută din conținutul fișierului printr-un algoritm de hashing, iar orice modificare, chiar și un singur caracter, va genera un checksum diferit. La final, checksum-ul este recalculat și comparat cu valoarea inițială. Dacă valorile se potrivesc, acest lucru sugerează că fișierul a rămas neatins. O valoare diferită a checksum-ului final semnalează modificarea fișierului astfel, putem identifica o posibilă corupere sau o modificare, fie accidentală, fie deliberată, a datelor. [1]

Una dintre cele mai mari provocări este coliziunea – situația în care două valori de intrare diferite generează același hash. Pentru a minimiza acest risc, algoritmii moderni, cum ar fi SHA-256 și SHA-3, sunt proiectați pentru a reduce probabilitatea acestor coliziuni. [2] Ei folosesc diferse metode care minimizează probabilitatea întâlnirii acestui eveniment:

* Folosirea sării (Salting) - adăugarea unui șir aleatoriu la final înainte de face operațiunea de hash pentru a preveni coliziunile și atacurile de tip ”rainbow table”.
* Hashing iterativ (Key stretching) - aplicarea repetată a funcției de hash crește securitatea și reduce coliziunile în cazul parolelor.
* Verificare prin HMAC - pentru autentificare, utilizarea HMAC (Hash-based Message Authentication Code) ajută la evitarea coliziunilor și la protecția integrității mesajelor

În comparație cu criptarea hashing-ul este unidirecțional astfel textul, fișierul asupra căruia este aplicată operația de hashing nu poate fi decodificat în datele inițiale. Datorită acestui fapt hashing-ul este potrivit pentru a fi utilizat la autentificare și verificare. Comparativ cu criptarea care conține o cheie care poate decripta mesajul, o face perfectă spre folosire în protectiția și transmiterea datelor confidențiale.

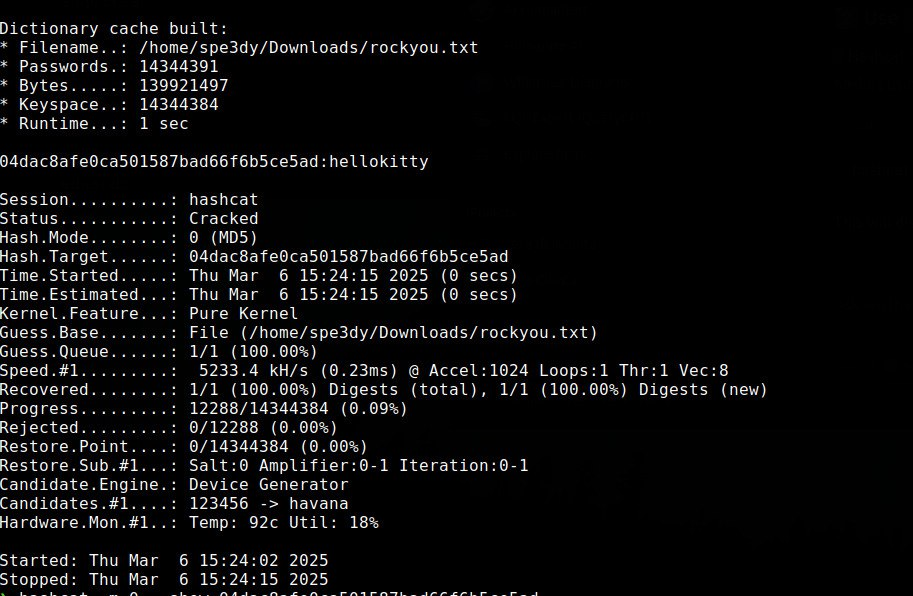
În concluzie, hashing-ul este o tehnologie esențială pentru securitatea cibernetică, fiind utilizat în gestionarea parolelor, autentificare, protecția datelor și verificarea integrității fișierelor. Alegerea unui algoritm de hashing adecvat este pentru a preveni atacurile cibernetice și pentru a asigura confidențialitatea și securitatea datelor digitale

## 1.1 Analiza și cercetarea soluțiilor existente

La moment platformele cele mai cunoscute care se ocupă de acest lucru sunt: “HashCat”, “Jhon the Ripper”, “Cain & Abel”, “OphCrack” .

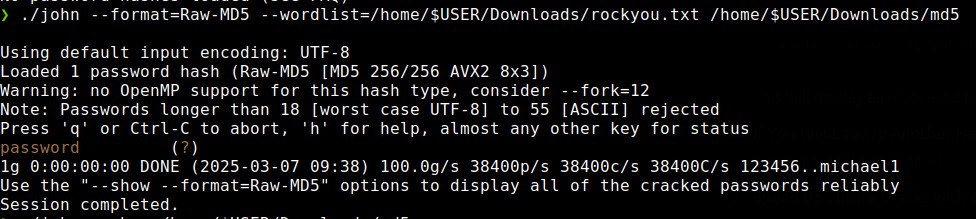
**Hashcat** este un instrument open-source și este unul dintre cele mai puternice și eficiente pentru atacuri brute-force și dictionary attack asupra diverselor tipuri de hash-uri. Acesta este optimizat pentru procesare GPU, permițând efectuarea unor atacuri rapide și eficiente asupra unor baze mari de date de hash-uri, datorită vitezei procesoarelor grafice. Hashcat suportă multiple moduri de atac, inclusiv atacuri de dicționar, combinare, hibrid și chiar atacuri bazate pe reguli personalizabile. Hashcat dispune de următoarele funcții. [3]

* poate fi utilizat pe mai multe sisteme de operare
* multi-platform (suport OpenCL și CUDA)
* peste 150 algoritmi implementați
* utilizare redusă a resurselor
* sistem de benchmarking incorporat
* poate efectua dictionary attacks la peste 30 de protocoale



**Figura 1.1 –** Aplicația “Hashcat”

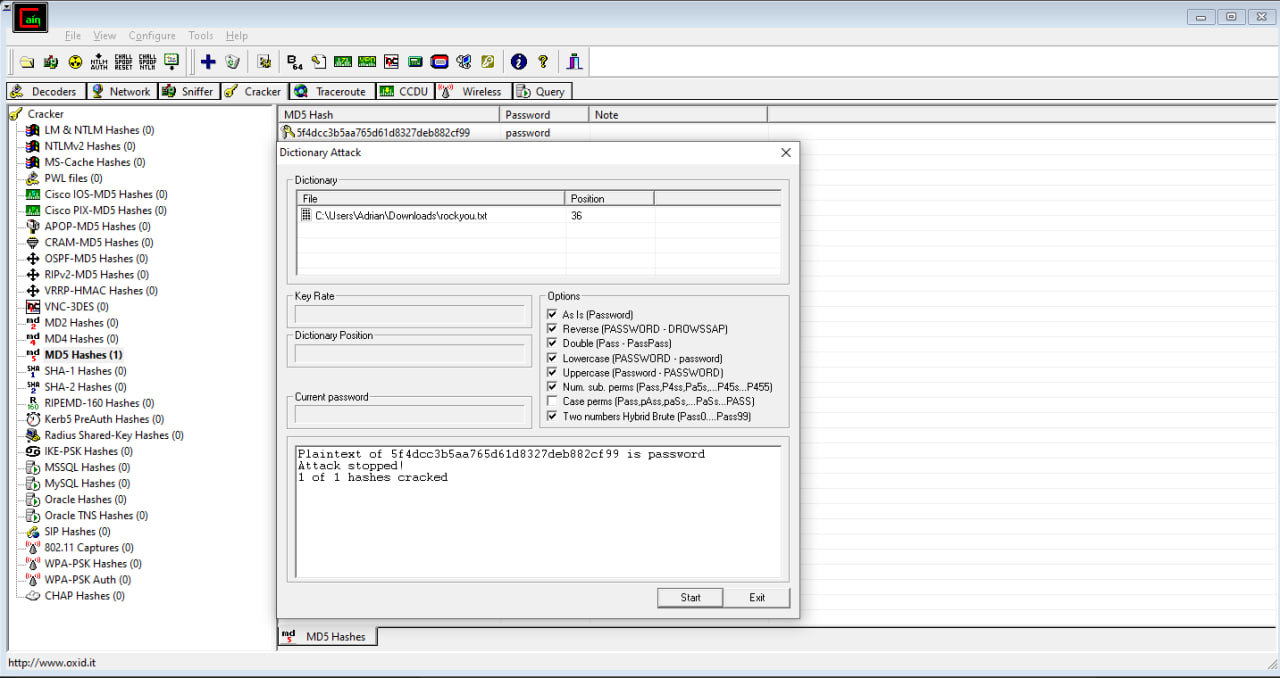
**John the Ripper** este un instrument versatil destinat testării parolelor și hash-urilor. Acesta poate funcționa atât în mod single-user, unde încearcă să ghicească parolele bazându-se pe modele comune, cât și în mod dictionary attack sau brute-force. John the Ripper este popular datorită capacității sale de a lucra pe multiple platforme și de a suporta mai multe formate de hash, inclusiv NTLM, LM, SHA, MD5 și multe altele. În plus, oferă posibilitatea de a personaliza atacurile prin reguli complexe. John the Ripper este optimizat pentru a utiliza eficient resursele sistemului, inclusiv suport pentru accelerare GPU și SIMD, ceea ce îmbunătățește semnificativ viteza de procesare.[4]



**Figura 1.2 –** Aplicația “John The Ripper”

**Cain & Abel** este un instrument multifuncțional de securitate ofensivă, utilizat nu doar pentru cracking-ul hash-urilor, ci și pentru sniffing-ul traficului de rețea, analiza protocolului VoIP și recuperarea parolelor ascunse sub asteriscuri în aplicațiile Windows. Cain & Abel este util în scenarii de testare a securității unde se analizează vulnerabilitățile din rețele și parolele utilizate în medii nesecurizate. Este instrumentul cel mai bun pentru atacurile de tip MITM, dar problema este că poate fi folosit doar pe windows și poate fi mai complicat pentru utilizatorii încapători.[3] Funcțiile de care dispune acest instrument sunt.

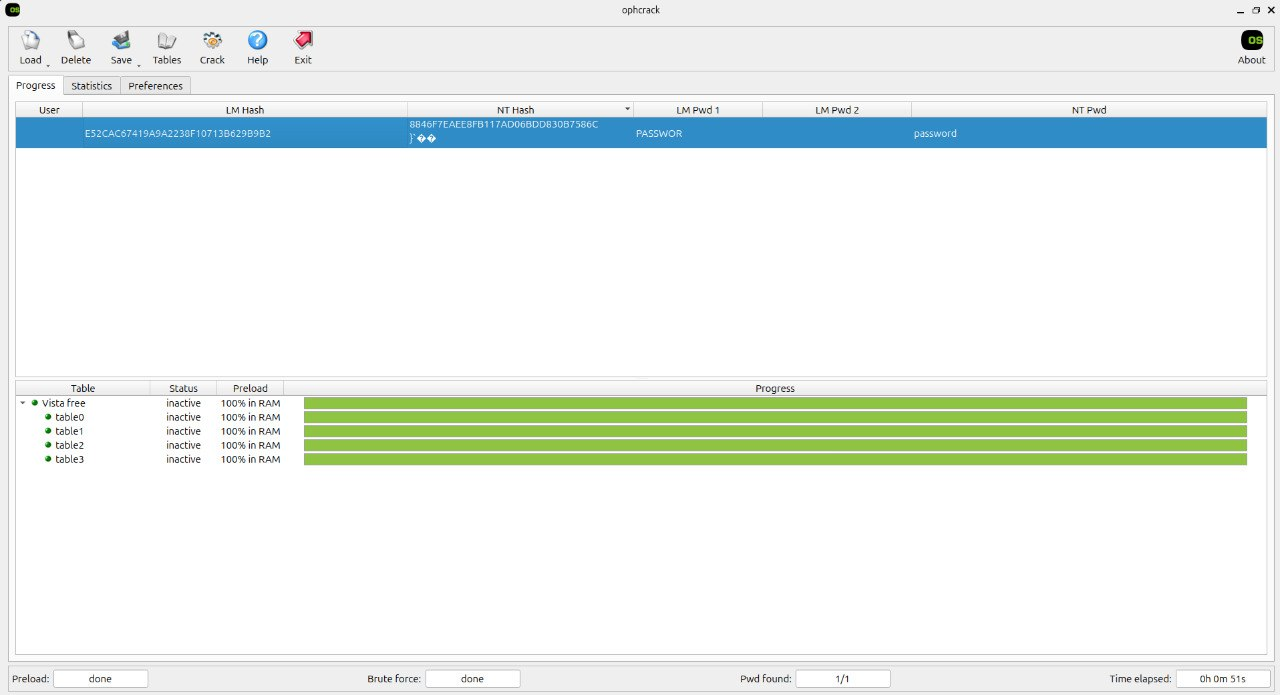
* folosit pentru web-cracking
* ARP spoofing
* posibilitatea de înregistra conversații VoIP
* acesta poate sparge diverse forme de hașuri LM și NT, hash-uri IOS și PIX hash-uri RADIUS, parole RDP, MD2, MD4, MD5, SHA-1, SHA-2, RIPEMD-160, Kerberos 5, MSSQL, MySQL, Oracle și SIP



**Figura 1.3 – Aplicația “Cain & Abel”**

**Ophcrack** este un software specializat în cracking-ul hash-urilor LM și NTLM utilizate de sistemele de operare Windows. Acesta folosește o metodă eficientă bazată pe rainbow tables, ceea ce permite spargerea rapidă a parolelor slab securizate. Ophcrack este preferat pentru recuperarea parolelor din sistemele Windows, deoarece poate funcționa independent de sistemul de operare și poate descifra parole chiar și fără acces direct la sistemul țintă. Acest instrument este suficient de rapid și ușor pentru un utilizator care sparge parole pentru prima dată cu cunoștințe de bază de Windows și poate sparge majoritatea parole în câteva minute, pe majoritatea computerelor.[3] Acesta are următoarele funcții.

* disponibil pe mai multe sisteme de operare ca Windows, Linux/Unix, Mac OS
* poate sparge hashuri de tim LM și NTLM
* atac combinatoric
* atac brute-force
* atac direct



**Figura 1.4 – Aplicația “Ophcrack”**

## 1.2 Definirea scopului și obiectivul proiectului

Scopul acestui proiect este de a dezvolta o aplicație interactivă destinată testării și analizei vulnerabilităților algoritmilor de hashing prin atacuri brute-force. Aplicația este concepută pentru utilizatorilor interesați de securitatea cibernetică și are ca obiectiv principal oferirea unui mediu de testare și învățare accesibil, care combină atât aspectele teoretice, cât și aplicațiile practice ale atacurilor asupra algoritmilor de hashing.

Din cauza unor posibile limitări hardware, aplicația va rula exclusiv pe procesor (CPU), fără suport pentru accelerare GPU. Această restricție poate afecta viteza și eficiența atacurilor brute-force, deoarece GPU-urile sunt semnificativ mai rapide în procesarea masivă a datelor. Ca urmare, timpul necesar pentru spargerea unui hash va fi mai mare comparativ cu soluțiile care utilizează accelerare hardware. Totuși, aplicația va optimiza execuția pe CPU pentru a maximiza performanța în limitele resurselor disponibile.

Obiectivele principale ale aplicației includ:

* Realizarea unei interfețe intuitive pentru a oferi o experiență bună utilizatorilor
* Identificarea tipului de hash utilizat.
* Generarea hash-urilor pentru diverse parole și texte introduse de utilizator.
* Verificarea unui hash în wordlist-uri compromise, care conțin posibile credențiale expuse, facilitând astfel testarea securității acestora.
* Încărcarea wordlist-urilor suplimentare pentru a oferi utilizatorilor o gamă mai largă de credențiale și date suplimentare.
* Efectuarea atacurilor de tip brute-force, încercând descifrarea unui hash prin testarea sistematică a combinațiilor posibile.
* Realizarea unui raport după finisarea atacului brute-force cu referire la timpul total de execuție, numărul de combinații testate, rata medie de testare pe secundă (hashes per second) și rezultatul final (dacă hash-ul a fost spart sau nu)
* Vizionarea în timp real a hash-ului testat și a rezultatului curent, oferind utilizatorilor posibilitatea de a monitoriza parolele încercate și de a analiza dinamica procesului de cracking.
* Posibilitatea de a opri, pune pe pauză și relua atacul fără a pierde progresul testării, permițând utilizatorilor să gestioneze eficient resursele hardware.

# 2 PROIECTAREA ȘI ARHITECTURA SISTEMUL INFORMATIC

În timpul proiectării arhitecturii unui sistem informatic se definesc structuri fundamentale, soluții software și scalabile. Proiectarea și arhitectura sistemului destinat aplicației de brute-force pentru cracking-ul hash-urilor va facilita înțelegerea cerințelor funcționale și non-funcționale. Pentru modelarea sistemului informatic se vor utiliza tehnici avansate care vor simula entitățile și relațiile dintre ele, vor reprezenta fluxul datelor și vor defini structurile software necesare pentru o implementare optimizată și eficientă. Proiectarea unui sistem informatic constă în arhitectura sistemului, care cuprinde entitățile, modulele, interfețele

Pentru proiectarea unui sistem informatic sunt disponibile mai multe opțiuni

* Metodele structurate, ele implică o abordare sistematică, divizând sistemul în module independente pentru a facilita înțelegerea și întreținerea.[5]
* Metode orientate pe obiect, ele concentrează pe modelarea sistemului în jurul obiectelor, care combină datele și comportamentul, promovând reutilizarea codului și întreținerea eficientă.[5]
* Metodele iterative și incrementale, ele presupun dezvoltarea sistemului în etape succesive, permițând testarea și evaluarea constantă a componentelor, reducând riscurile și identificând problemele mai devreme în procesul de dezvoltare.[5]
* Metode funcționale, ele se axează pe utilizarea funcțiilor matematice pure pentru a modela comportamentul sistemului, eliminând efectele secundare și mutabilitatea stărilor, ceea ce crește predictibilitatea și fiabilitatea sistemului.[5]

În plus, utilizarea unui limbaj de modelare precum UML (Unified Modeling Language) permite o reprezentare clară și standardizată a structurii și comportamentului sistemului. Diagramele UML contribuie la o mai bună comunicare între dezvoltatori, utilizatori și alte părți interesate, facilitând atât înțelegerea, cât și implementarea soluției.

## 2.1 Cerințele funcționale ale sistemului

Pentru a putea realiza un sistem informatic, este necesară stabilirea inițială a cerințelor funcționale. Acestea definesc acțiunile pe care sistemul trebuie să le îndeplinească pentru a asigura funcționalitatea dorită. Printre cerințele funcționale ale sistemului se numără: introducerea unui hash și selectarea metodei de brute-force, detectarea automată a tipului de hash introdus, posibilitatea de a încărca un wordlist sau un rainbow table pentru optimizarea procesului, vizualizarea progresului în timp real și salvarea unui raport detaliat la finalizarea sau oprirea procesului.

Fiecare dintre aceste funcționalități a fost detaliată, subliniind modul în care utilizatorii vor interacționa cu sistemul și cum se va realiza procesarea datelor. Introducerea unui hash este punctul de start al aplicației, permițând utilizatorilor să introducă manual un hash pe care doresc să-l spargă. În ceea ce privește determinarea tipului de hash, aplicația va analiza formatul și lungimea acestuia pentru a identifica algoritmul utilizat la generarea sa. Acest proces ajută utilizatorii să aleagă cea mai eficientă metodă de atac, optimizând timpul de procesare.

Un aspect principal al sistemului este posibilitatea de a încărca wordlist-uri sau rainbow tables. Această funcționalitate permite utilizatorilor să utilizeze seturi de date preexistente pentru a accelera procesul de spargere a hash-urilor, în loc să genereze combinații complet aleatorii. De asemenea, aplicația trebuie să suporte atât wordlist-uri standard, cât și personalizate, oferind utilizatorilor flexibilitate în strategia de atac.

Un alt element important este vizualizarea progresului în timp real. Utilizatorii trebuie să poată urmări informații despre numărul de combinații testate, rata de procesare (hash-uri/secundă), timpul estimat pentru finalizarea procesului și rezultatele intermediare. Aceste date vor fi afișate într-o interfață grafică intuitivă, facilitând monitorizarea și ajustarea parametrilor de atac, dacă este necesar.

După finalizarea sau oprirea procesului, utilizatorii vor putea generea un raport detaliat. Acest raport trebuie să includă hash-ul inițial, parola descoperită (dacă a fost identificată), metoda utilizată, durata totală a procesului și alte statistici relevante. În cazul în care procesul a fost întrerupt, raportul trebuie să reflecte progresul realizat până în acel moment, astfel încât utilizatorii să poată relua atacul de unde a fost întrerupt.

Pentru a asigura o experiență de utilizare optimă, toate aceste funcționalități vor fi integrate într-o interfață de utilizator (UI) intuitivă și ușor de navigat. O interfață bine concepută, bazată pe principii de design centrate pe utilizator, facilitează accesul rapid la informații și îmbunătățește satisfacția utilizatorilor. Pentru a atinge acest obiectiv, este important să se acorde atenție detaliilor și să se pună în practică principii de design bine stabilite.[6] Astfel, utilizatorii vor putea interacționa eficient cu sistemul, beneficiind de o experiență plăcută și productivă.

## 2.2 Cerințele non-funcționale ale sistemului

Cerințele nefuncționale se referă la atributele calitative ale unui sistem software, influențând aspecte precum performanța, securitatea, scalabilitatea și experiența generală a utilizatorului. Aceste cerințe, deși nu descriu funcționalități specifice, sunt necesare pentru asigurarea eficienței și fiabilității sistemului astfel ele pot impune anumite restricții sau contrângeri asupra sistemului.[7]

Pentru ca sistemul să fie performant trebuie să fie capabil să proceseze hash-urile într-un mod eficient, pentru acest lucru e nevoie de optimizarea algoritmilor pentru a minimiza timpul de procesare. Optimizarea algoritmilor eficientizează și gestionarea resurselor hardware pentru a nu supraîncărca sistemul utilizatorului. De asemenea, scalabilitatea este crucială pentru a permite sistemului să se adapteze la volume mari de date, cum ar fi wordlist-urile sau tabelele rainbow de dimensiuni considerabile. Aceste aspecte sunt fundamentale în proiectarea și implementarea unui sistem informatic robust și eficient.

Pentru a asigura fiabilitatea și disponibilitatea sistemului, este necesar ca acesta să funcționeze continuu și fără erori, menținând o disponibilitate ridicată. Implementarea mecanismelor de backup și recuperare este crucială pentru prevenirea pierderii datelor și pentru asigurarea continuității operațiunilor. În plus, sistemul trebuie să fie rezilient la diverse tipuri de atacuri sau defecțiuni, menținându-și funcționalitatea chiar și în condiții adverse.

Sistemul trebuie să asigure un nivel ridicat de fiabilitate și disponibilitate, garantând o funcționare continuă și stabilă în mare parte din timp, fără erori critice care să afecteze utilizatorii. Pentru a preveni pierderea datelor în cazul unor întreruperi neașteptate, aplicația va include un mecanism de salvare automată a progresului utilizatorului, permițând reluarea activităților din punctul în care au fost întrerupte, asigurând astfel o experiență fluidă și neîntreruptă.

Aplicația trebuie să fie proiectată astfel încât să permită integrarea unor noi algoritmi de brute-force, oferind flexibilitate în extinderea funcționalităților fără modificări majore în structura existentă. De asemenea, codul va fi organizat modular, asigurând o arhitectură clară și bine definită, ceea ce va facilita atât întreținerea, cât și actualizarea sistemului pe termen lung.

Pentru a asigura corectitudinea și funcționalitatea algoritmilor implementați, aplicația va include teste unitare și de integrare, verificând astfel atât componentele individuale, cât și interacțiunea dintre ele. De asemenea, testarea va fi realizată în diferite scenarii, simulând diverse condiții de utilizare, pentru a evalua eficiența tehnicilor brute-force și impactul acestora asupra performanței sistemului. Această abordare va permite identificarea și remedierea eventualelor erori înainte de implementarea finală, garantând o funcționare stabilă și optimizată.

Aplicația va fi dezvoltată și utilizată cu respectarea principiilor etice, evitând orice utilizare abuzivă sau ilegală a tehnicilor brute-force implementate. Scopul principal al sistemului este de a contribui la cercetare și securitate cibernetică, nu de fi folosită pentru posibile activități malițioase, iar utilizatorii vor fi informați cu privire la utilizarea corectă și responsabilă a software-ului.

## 2.3 Modelarea comportamentală a sistemului

Descrierea comportamentală a sistemului, conform standardului UML, se realizează prin diferite tipuri de diagrame care pun accentul pe aspecte specifice ale interacțiunii dintre componentele sale. Aceste diagrame oferă o perspectivă transparentă asupra modului în care funcționează aplicația, permițând echipei de dezvoltare să înțeleagă procesele interne și modul în care utilizatorii interacționează cu sistemul.

Pentru aplicația de spargere a hash-urilor prin forța brută, următoarele diagrame UML sunt utilizate pentru modelarea comportamentală:

* Diagrama cazurilor de utilizare (Use case diagram) - prezintă o imagine de ansamblu a sistemului, subliniind actorii și funcționalitățile principale.
* Diagrama de activități (Activity diagram) - descrie fluxurile de date și etapele necesare pentru finalizarea unui proces.
* Diagrama de stări (Statechart diagram) - prezintă tranzițiile și schimbările de stare ale componentelor sistemului.
* Diagrama de secvență (Sequence diagram) - reprezintă interacțiunile dintre componente într-o anumită secvență temporală.
* Diagrama de colaborare (Collaboration diagram) - ilustrează fluxurile de mesaje și conexiunile dintre componentele sistemului.

Aceste diagrame oferă o reprezentare cuprinzătoare și intuitivă a comportamentului sistemului, facilitând identificarea și optimizarea proceselor principale pentru funcționalitatea aplicației. Prin intermediul diagramelor comportamentale, dezvoltatorii pot vizualiza și analiza secvențele de acțiuni, interacțiunile între obiecte și evoluția stărilor acestora în timp.

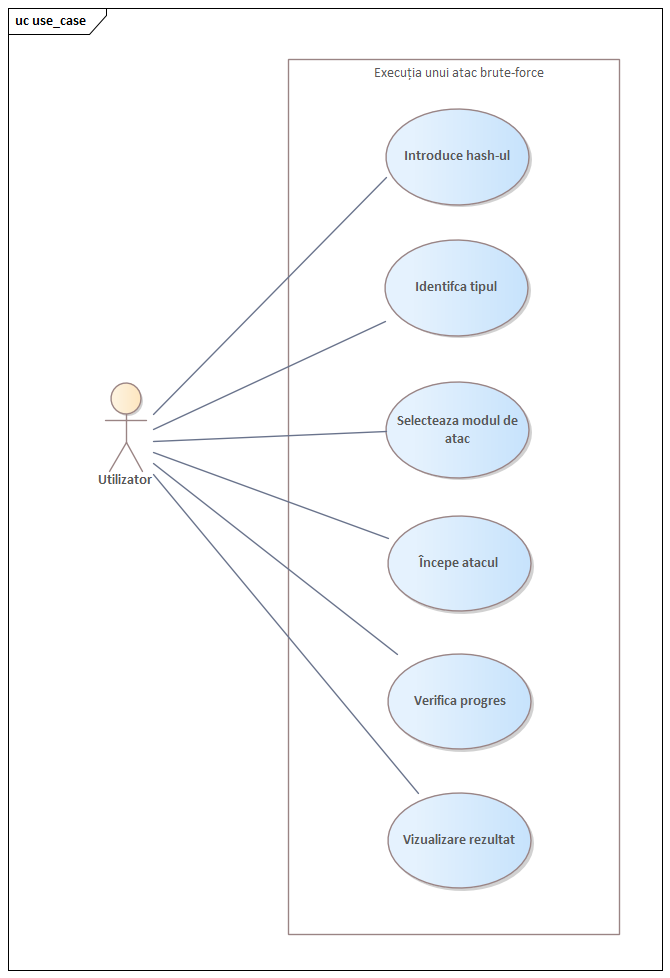
## 2.3.1 Generalitățile sistemului informatic

Sistemul informatic dezvoltat pentru analiza și implementarea tehnicilor brute-force în procesul de cracking al hash-urilor urmărește să ofere o soluție eficientă și intuitivă pentru utilizatori. Acesta este structurat modular, incluzând o interfață grafică pentru gestionarea procesului, un motor de procesare dezvoltat în Golang pentru efectuarea atacurilor brute-force, precum și componente pentru administrarea listelor de parole și generarea rapoartelor.

Pentru a asigura o proiectare clară și o înțelegere detaliată a interacțiunii dintre utilizatori și sistem, utilizăm diagramele Use Case. Ea oferă o reprezentare vizuală a tuturor scenariilor în care utilizatorul poate interacționa cu sistemul, evidențiind acțiunile principale și fluxurile de execuție.

Diagramele Use Case sunt utile deoarece:

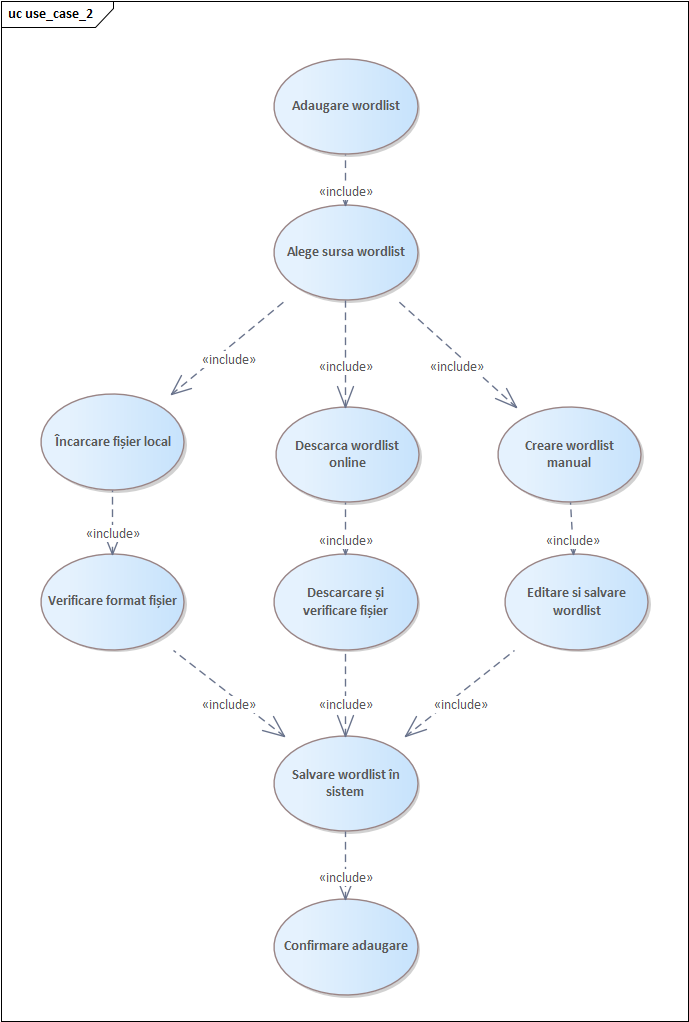
* Clarifică cerințele funcționale ale aplicației.
* Evidențiază interacțiunile dintre utilizatori și sistem, ajutând la definirea comportamentului general al aplicației.
* Simplifică procesul de comunicare între echipa de dezvoltare și utilizatorii finali, facilitând înțelegerea cerințelor și a funcționalităților.
* Ajută la testarea și validarea sistemului, oferind o bază pentru verificarea scenariilor de utilizare.



**Figura 2.1 –** Funcționalități generale ale sistemului

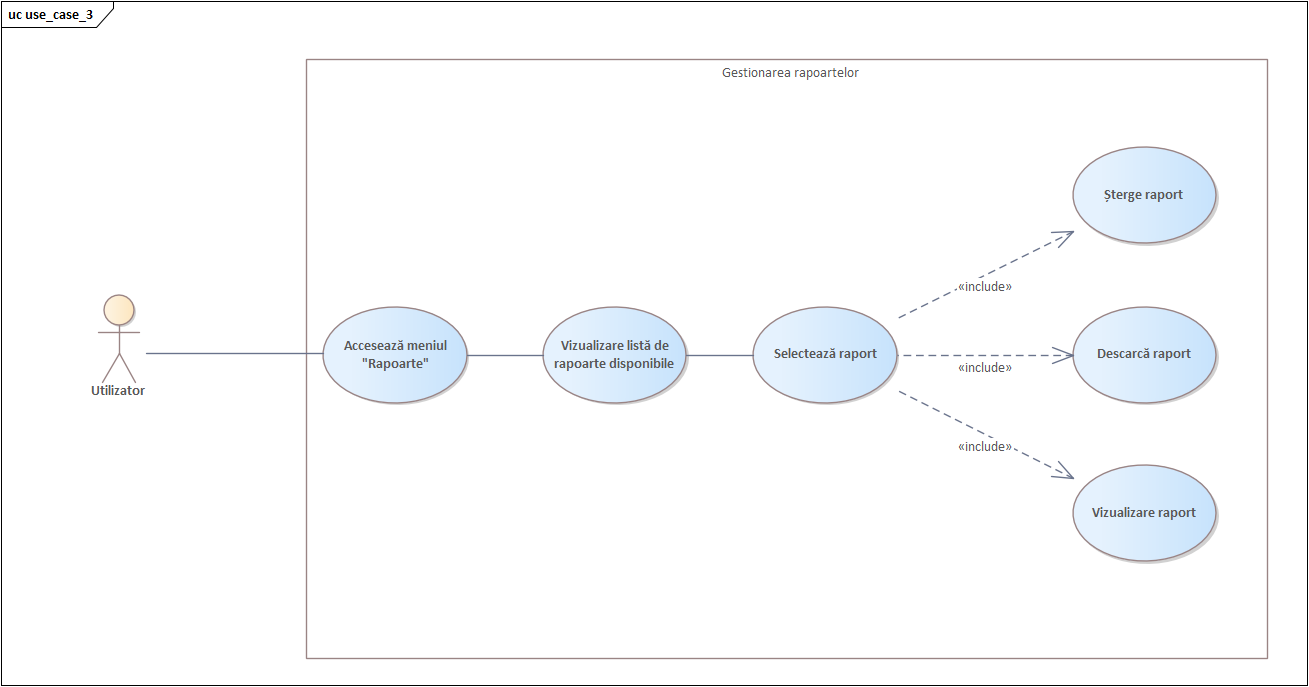
Diagrama din figura 2.1 reprezintă cazurile pentru funcționalitățile generale ale aplicației de cracking hash prin atac brute-force. Aceasta oferă o viziune de ansamblu asupra procesului prin care utilizatorul poate executa un astfel de atac.

Utilizatorul poate introduce un hash pe care dorește să-l spargă, iar sistemul identifică automat tipul acestuia pentru a selecta metoda adecvată. După identificarea tipului de hash, utilizatorul poate alege modul de atac, optând între diferite strategii, cum ar fi forța brută pură sau metode bazate pe dicționar. Odată selectată metoda, utilizatorul inițiază atacul, iar sistemul începe procesul de spargere a hash-ului. Pe parcurs, utilizatorul are posibilitatea de a verifica progresul atacului în timp real, urmărind eventualele rezultate parțiale. La finalul procesului, aplicația afișează rezultatul atacului, indicând dacă hash-ul a fost spart sau dacă încercarea a eșuat.



**Figura 2.2 -** Adăugarea unui wordlist

Diagrama din figura 2.2 reprezintă procesul de adăugare a unui wordlist în aplicație. Utilizatorul va avea avea 3 metode de adăugare a unui wordlist, ele sunt următoarele: încărcarea unui fișier local, descărcarea unui wordlist online sau crearea manuală a unuia nou. În cazul încărcării unui fișier sau descărcării unui wordlist, sistemul verifică automat formatul acestuia pentru a se asigura că este compatibil. Dacă fișierul este valid, acesta este salvat și poate fi folosit în atacurile brute-force. În cazul în care utilizatorul optează pentru crearea manuală a unui wordlist, acesta poate edita și salva conținutul direct în aplicație. La final, sistemul confirmă adăugarea wordlist-ului, iar acesta devine utilizabil în procesul de spargere a hash-urilor.



**Figura 2.3 –** Interacțiunea utilizatorului cu modulul de rapoarte

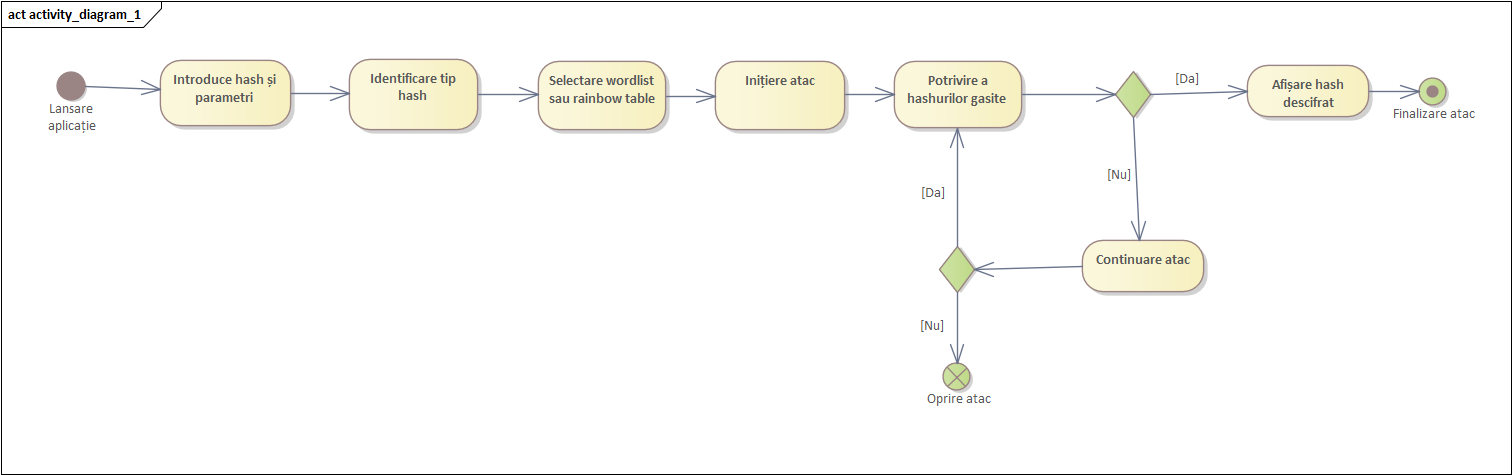
În diagrama 2.3 este ilustrată interacțiunea utilizatorului cu modulul de gestionare a rapoartelor din cadrul aplicației. Utilizatorul accesează secțiunea dedicată rapoartelor generate în sesiunile anterioare de cracking, moment în care sistemul afișează o listă cu documentele disponibile. Asupra fiecărui raport selectat, utilizatorul poate efectua una dintre cele trei acțiuni: „Vizualizare raport”, „Descarcă raport” sau „Șterge raport”.

## 2.3.2 Fluxul operațional al aplicației

În timpul dezvoltării aplicației, diagramele de activitate sunt importante pentru modelarea și înțelegerea fluxului operațional al sistemului. Aceste diagrame oferă o reprezentare vizuală a succesiunii de acțiuni și decizii din cadrul proceselor aplicației, facilitând o perspectivă clară asupra funcționării acesteia.​

Importanța diagramelor de activitate în cadrul aplicației:

* Clarificarea fluxului de procese: Diagramele de activitate permit identificarea și reprezentarea secvențială a activităților, evidențiind tranzițiile și condițiile dintre acestea. ​
* Detectarea potențialelor blocaje sau ineficiențe: Prin analiza diagramelor de activitate, se pot identifica zonele susceptibile la blocaje sau ineficiențe în fluxul operațional, oferind oportunități de optimizare.​
* Facilitarea comunicării între echipă: O reprezentare vizuală clară a proceselor ajută la colaborarea membrilor echipei de dezvoltare și la asigurarea unei înțelegeri comune a funcționalităților sistemului.

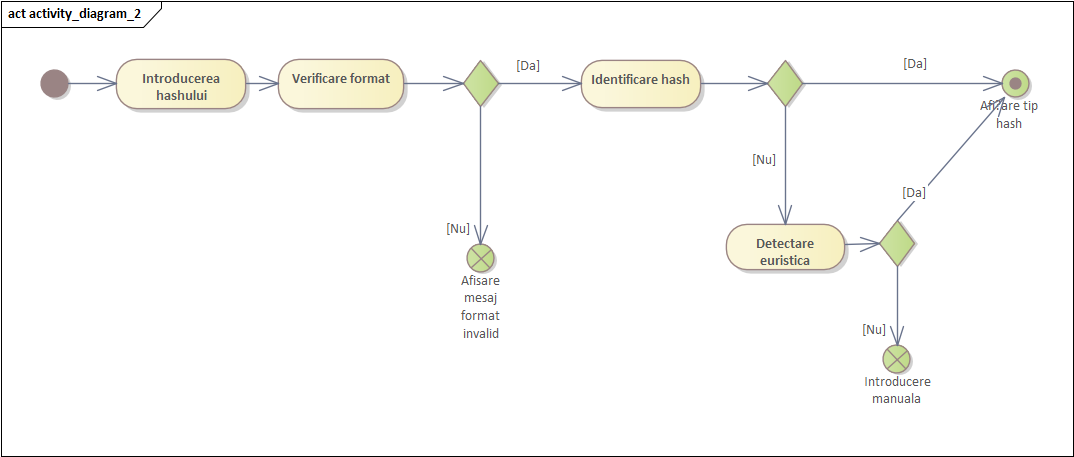


**Figura 2.4 –** Fluxul procesului de cracking

În figura 2.4 este reprezentat fluxul procesului pe care îl urmează aplicația pentru a face brute-force unui hash. Utilizatorul introduce un hash, iar sistemul identifică automat tipul de hash sau utilizatorul poate specifica tipul manual în parametri, acești pași sunt necesari pentru alegerea metodei de atac.

După selecția strategiei (wordlist sau rainbow table), aplicația inițiază atacul și compară hash-urile generate cu cel introdus. Dacă se găsește o potrivire, parola este descifrată și afișată. În caz contrar, utilizatorul poate continua atacul sau opri procesul.

Fluxul se finalizează fie prin afișarea parolei descifrate, fie prin oprirea atacului în lipsa unui rezultat, oferind utilizatorului flexibilitate în alegerea metodei potrivite.



**Figura 2.5 –** Procesul de identifcare al unui hash

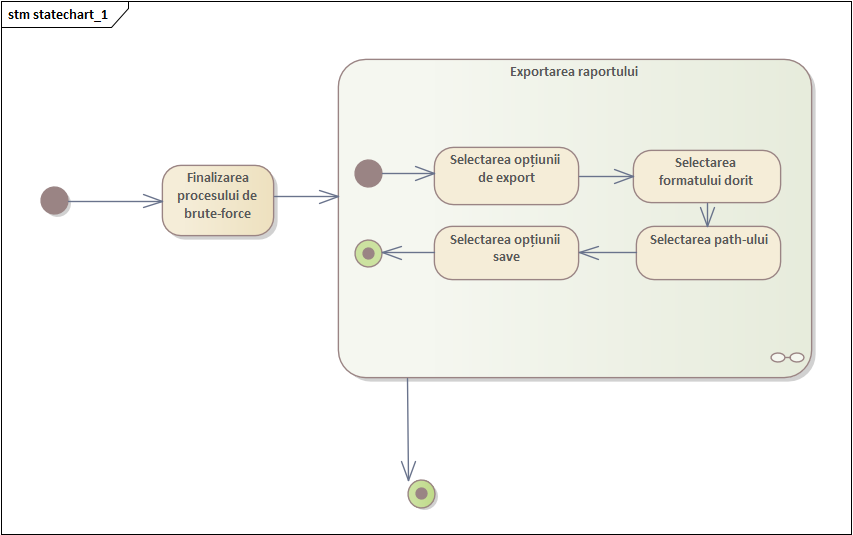
În figura 2.5 este ilustrat fluxul procesului de identificare a unui hash. Aplicația verifică automat dacă formatul hash-ului este valid, permițând continuarea procesului doar în cazul în care acesta este recunoscut. Dacă tipul hash-ului este identificat, utilizatorului îi este afișată informația corespunzătoare. În situația în care identificarea automată eșuează, sistemul recurge la o metodă de detectare euristică. Dacă nici această metodă nu oferă un rezultat clar, utilizatorul are posibilitatea de a introduce manual tipul hash-ului.

## 2.3.3 Modelarea comportamentului sistemului

În timpul dezvoltării aplicației, diagramele de stare sunt fundamentale în modelarea și înțelegerea comportamentului dinamic al sistemului. Aceste diagrame oferă o reprezentare vizuală a diferitelor stări prin care poate trece o componentă a sistemului și modul în care tranzițiile între aceste stări sunt declanșate de evenimente specifice.

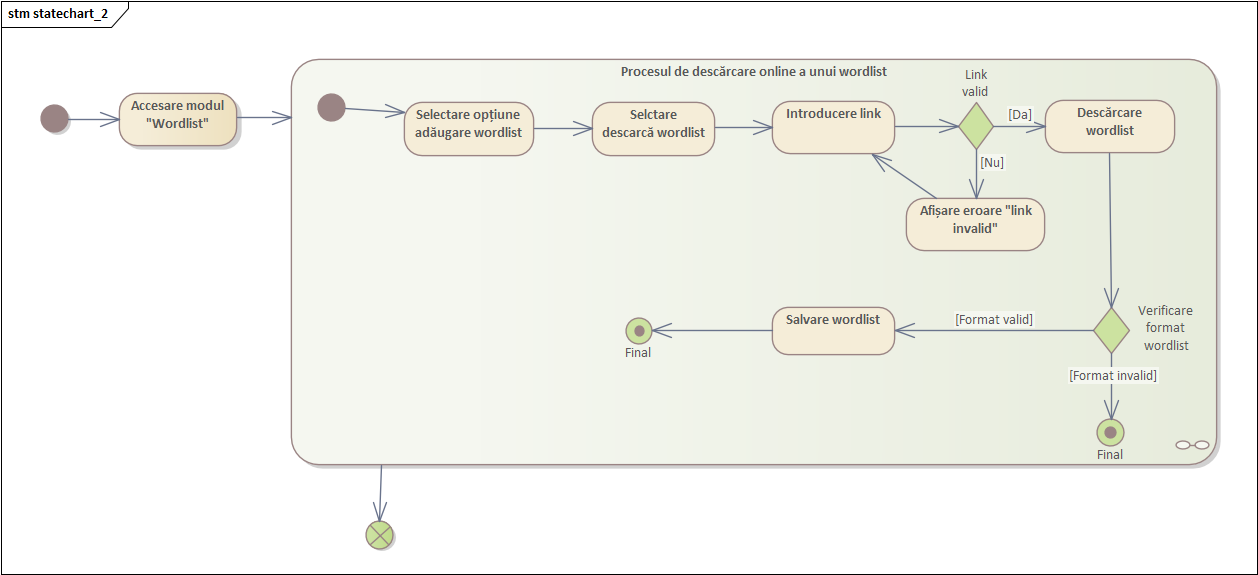
Importanța diagramelor de stare în cadrul aplicației:

* Clarificarea comportamentului sistemului: Diagramele de stare permit identificarea și reprezentarea secvențială a stărilor și tranzițiilor dintre acestea, evidențiind condițiile și evenimentele care determină schimbările de stare. Aceasta ajută la o înțelegere profundă a modului în care sistemul reacționează la diferite stimuli și situații. ​
* Identificarea comportamentelor complexe și a scenariilor de eroare: Prin analiza diagramelor de stare, se pot identifica comportamente complexe, bucle infinite sau stări neacoperite, oferind oportunități de optimizare și asigurând o funcționare robustă a sistemului.
* Facilitarea comunicării între membrii echipei: O reprezentare vizuală clară a comportamentului sistemului ajută la alinierea membrilor echipei de dezvoltare și la asigurarea unei înțelegeri comune a funcționalităților și reacțiilor sistemului în diverse situații. ​



**Figura 2.6 –** Exportarea raportului

Diagrama din figura 2.6 modelează comportamentul sistemului în etapa de exportare a raportului după finalizarea unui atac brute-force. După încheierea procesului de cracking, utilizatorul selectează opțiunea de export și alege formatul dorit, urmat de specificarea locației unde fișierul va fi salvat. Diagrama evidențiază succesiunea stărilor interne ale sistemului și condițiile logice necesare pentru a ajunge la finalizarea operației. Totodată, subliniază modul secvențial în care se parcurg pașii de configurare a exportului.



**Figura 2.7 –** Descărcarea și validarea unui wordlist din sursă online

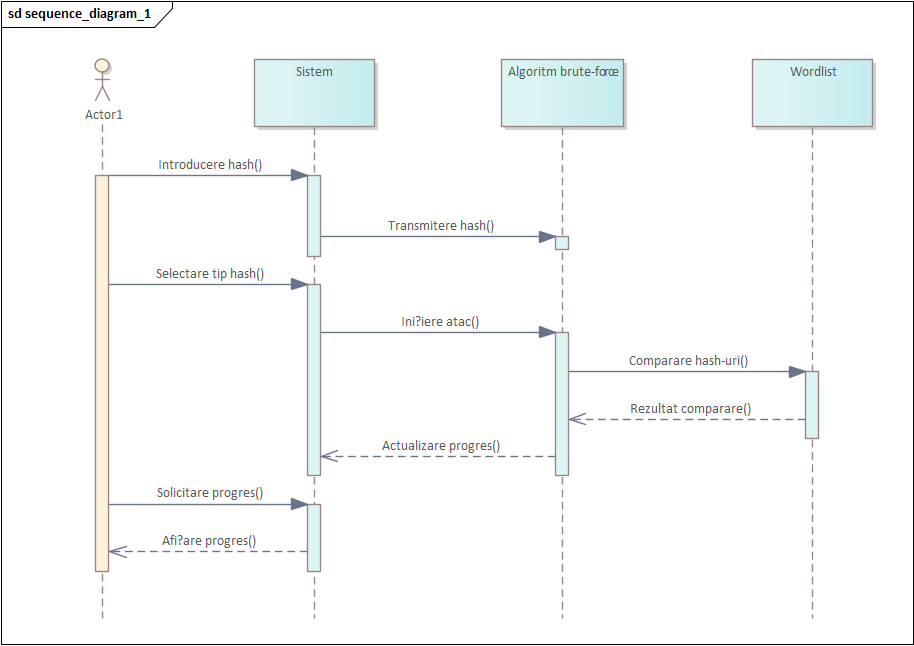
Diagrama din figura 2.7 simulează fluxul de stări prin care trece aplicația atunci când utilizatorul dorește să adauge un wordlist dintr-o sursă online. După introducerea linkului, sistemul verifică validitatea acestuia, permițând continuarea doar dacă link-ul este valid. După descărcare, fișierul este verificat din punct de vedere al formatului, iar dacă este valid, este salvat în sistem. Diagrama evidențiază atât traseul ideal, cât și gestionarea cazurilor de eroare (link invalid, format greșit).

## 2.3.4 Descrierea fluxurilor de interacțiune utilizator–sistem

Diagramele de secvență oferă o reprezentare clară și detaliată a modului în care componentele unui sistem software interacționează în timp pentru a îndeplini anumite funcționalități. Acestea evidențiază ordinea mesajelor schimbate între obiecte, subliniind fluxurile de control și de date care apar între utilizator și subsistemele aplicației. Prin utilizarea diagramelor de secvență, se poate înțelege exact comportamentul sistemului în diverse scenarii de utilizare, contribuind astfel la validarea logicii de implementare și la facilitarea comunicării între membrii echipei de dezvoltare.[8] În plus, ele servesc ca documentație valoroasă pe termen lung, oferind o bază solidă pentru mentenanța și extinderea ulterioară a aplicației.[9]

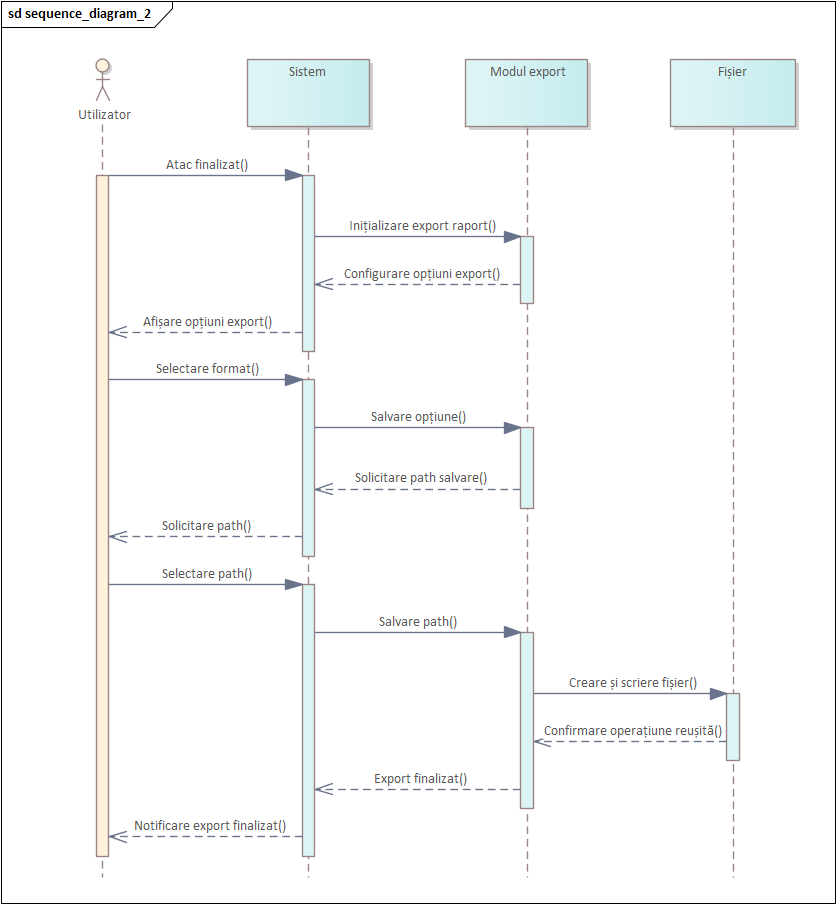
În cadrul lucrării, diagramele de secvență sunt aplicate pentru a reprezenta procese precum inițierea și execuția unui atac de tip brute-force, exportul raportului după finalizarea atacului și identificarea automată a tipului de hash. Prin aceste reprezentări vizuale, se evidențiază succesiunea mesajelor transmise între obiecte și fluxul de control asociat fiecărui scenariu, contribuind la clarificarea cerințelor funcționale și la detectarea timpurie a eventualelor probleme de proiectare.​

Astfel, utilizarea diagramelor de secvență în această teză nu doar că sprijină procesul de dezvoltare și documentare a aplicației, dar și îmbunătățește comunicarea în cadrul echipei de proiect, asigurând o înțelegere comună și detaliată a mecanismelor interne ale sistemului.



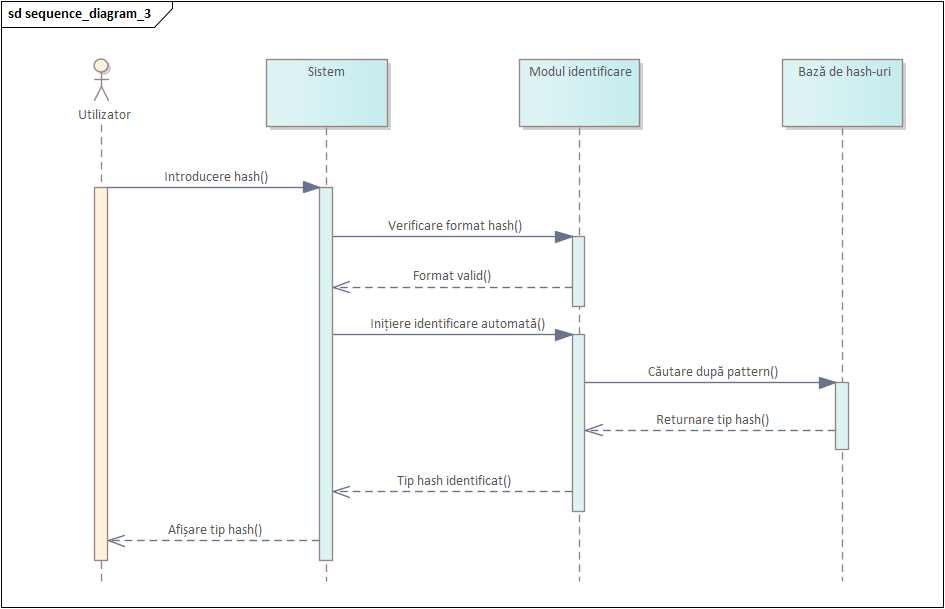
**Figura 2.8 -** Inițierea și execuția unui atac de tip brute-force

În figura 2.8 este simulat procesul de inițiere și execuție a unui atac de tip brute-force. Utilizatorul introduce hash-ul și selectează tipul acestuia, interacționând cu sistemul central care transmite datele către modulul de algoritm. Acesta realizează procesul de comparare folosind un wordlist și actualizează progresul, care este transmis înapoi spre sistem și afișat utilizatorului în timp real. Această diagramă evidențiază clar succesiunea pașilor implicați în atacul brute-force, subliniind interacțiunile dintre componentele sistemului și utilizator.



**Figura 2.9 -** Exportul raportului după finalizarea atacului

Figura 2.9 descrie secvența asociată exportului raportului după finalizarea atacului. Sistemul inițiază procesul de export prin comunicarea cu modulul dedicat, care, la rândul său, configurează opțiunile de export pe baza alegerii utilizatorului. După selectarea formatului și a locației de salvare, fișierul este generat, scris și confirmat ca fiind salvat cu succes, iar utilizatorul este notificat. Această diagramă subliniază importanța interacțiunii dintre utilizator și sistem în configurarea și finalizarea cu succes a procesului de export al raportului.



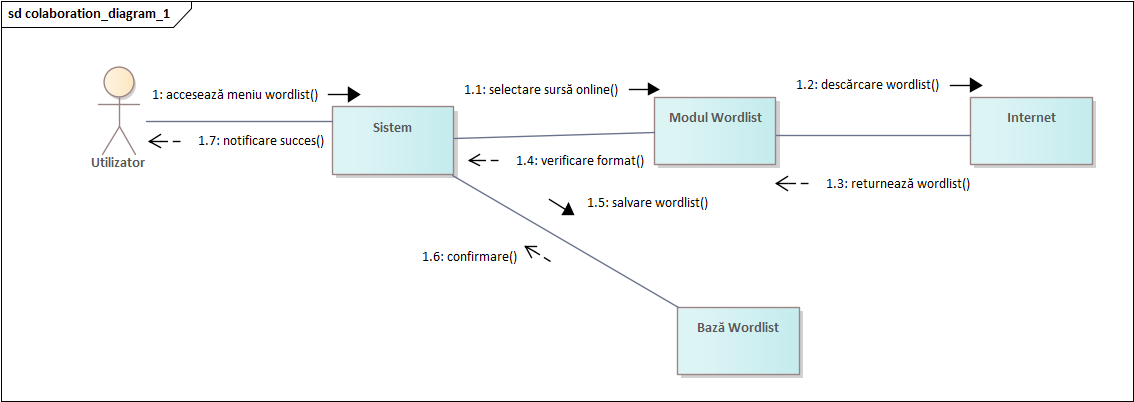
**Figura 2.10 –** Identificarea algorimtului de hash

Figura 2.10 prezintă interacțiunea dintre utilizator și sistem în procesul de identificare automată a tipului de hash. După introducerea valorii hash, sistemul apelează modulul de identificare care verifică formatul și, dacă este valid, inițiază o căutare euristică în baza de hash-uri pentru a determina tipul acestuia. În funcție de rezultat, sistemul comunică utilizatorului tipul identificat. Această diagramă evidențiază eficiența sistemului în automatizarea procesului de identificare a tipului de hash, reducând astfel efortul utilizatorului și minimizând posibilitatea erorilor umane.

## 2.3.5 Utilizarea diagramelor de comunicare pentru evidențierea fluxurilor informaționale

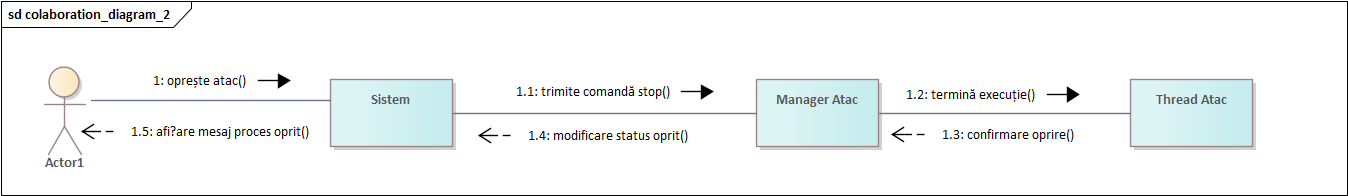
Diagramele de comunicare (cunoscute și ca diagrame de colaborare) reprezintă un tip de diagramă UML utilizat pentru a evidenția interacțiunile dintre obiecte în cadrul unui sistem, punând accent pe schimbul de mesaje într-un context structural. Aceste diagrame sunt utile în analiza și proiectarea sistemelor software, deoarece permit înțelegerea clară a modului în care entitățile colaborează pentru a îndeplini o funcționalitate specifică [10].

În cadrul proiectului, au fost realizate două astfel de diagrame, fiecare reflectând un scenariu distinct de interacțiune între componentele sistemului dezvoltat. Aceste diagrame oferă o imagine clară a succesiunii mesajelor și a responsabilităților fiecărei entități, asigurând astfel o bază solidă pentru implementarea logicii aplicației.



**Figura 2.11 –** Descărcarea și salvarea unui wordlist

În cadrul figurii 2.11 este prezintată diagrama de comunicare corespunzătoare procesului de selectare și salvare a unei liste de cuvinte (wordlist) dintr-o sursă online. Interacțiunea debutează cu inițiativa utilizatorului de a accesa meniul dedicat gestionării wordlist-urilor, moment în care sistemul declanșează o secvență de mesaje care implică Modulul Wordlist și o resursă externă din Internet. După ce lista este descărcată, aceasta este verificată din punct de vedere al formatului, urmând ca, în cazul în care este validă, să fie salvată în baza de date. Procesul se încheie cu confirmarea operațiunii și notificarea utilizatorului cu privire la succesul acțiunii.



**Figura 2.12 –** Oprirea unui atac în execuție

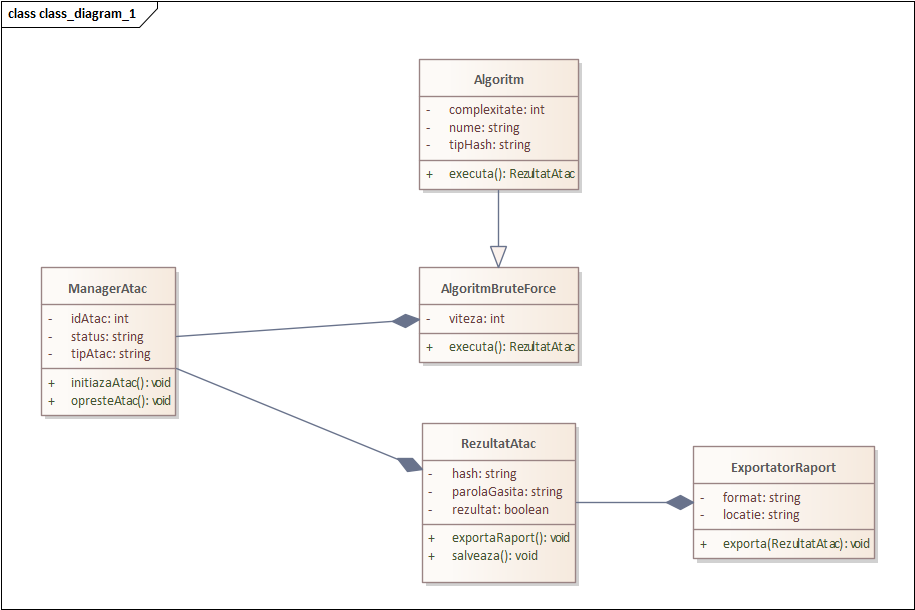
Figura 2.12 ilustrează un scenariu distinct, cel al opririi unui atac activ. Actorul principal al sistemului inițiază oprirea printr-un mesaj transmis către sistem, care, la rândul său, propagă comanda de oprire către Managerul de Atac. Acesta comunică cu firul de execuție dedicat atacului, solicitând finalizarea acestuia. După confirmarea opririi, sistemul actualizează starea internă corespunzătoare și notifică utilizatorul cu un mesaj care atestă oprirea completă a procesului. Diagrama evidențiază succesiunea logică și clară a pașilor parcurși pentru întreruperea controlată a unui proces critic.

## 2.3.6 Utilizarea diagramelor de clasă pentru modelarea componentelor sistemului

Diagramele de clasă reprezintă un instrument fundamental în ingineria software, utilizat pentru a modela structura statică a unui sistem. Aceste diagrame ilustrează clasele componente ale sistemului, atributele și metodele acestora, precum și relațiile dintre ele, oferind o perspectivă clară asupra arhitecturii și interacțiunilor interne ale aplicației. O clasă într-o diagramă UML reprezintă o categorie de entități cu atribute și comportamente comune, servind drept șablon pentru crearea obiectelor cu caracteristici și funcționalități specifice[11]. Diagramele de clasă sunt utilizate pentru a modela structura (viziunea statică asupra) unui sistem, incluzând clase, interfețe, obiecte și relațiile dintre acestea[12].

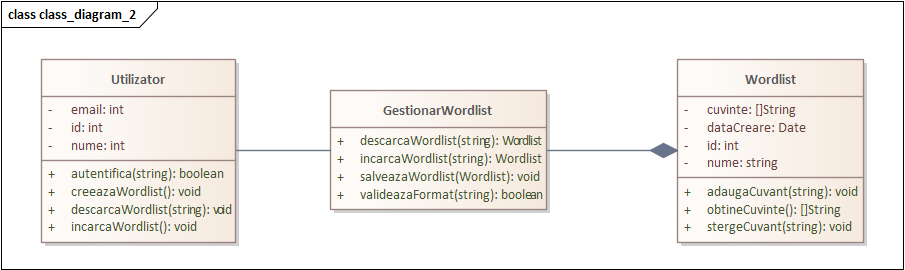
În procesul de dezvoltare al sistemului, diagramele de clasă au fost importante în modelarea componentelor cheie, precum procesele de atac brute-force, mecanismele de export al rapoartelor și gestionarea listelor de cuvinte (wordlist-uri, rainbow tables). Aceste reprezentări au eficientizat înțelegerea structurii și comportamentului fiecărei componente, simplificând etapele de proiectare și implementare.​

În continuare, vom examina în detaliu diagramele de clasă relevante pentru sistemul dezvoltat, subliniind modul în care acestea reflectă arhitectura și funcționalitățile aplicației.​



**Figura 2.13 -** Diagrama de clasă pentru procesul de atac de tip brute-force

Diagrama de clasă din figura 2.13 este specifică procesului de atac brute-force care evidențiază structura statică a componentelor implicate și relațiile dintre acestea. Managerul de atac coordonează execuția atacului, gestionând inițierea, oprirea și obținerea rezultatelor, menținând atribute precum identificatorul atacului, tipul și starea curentă. Algoritmul reprezintă o entitate abstractă ce definește structura generală a unui atac, specificând numele, tipul de hash și complexitatea asociată. Algoritmul de tip brute-force extinde această entitate abstractă, adăugând atributul de viteză și implementând metoda de execuție specifică acestui tip de atac. Rezultatul atacului stochează informațiile obținute, inclusiv hash-ul, parola identificată și succesul operațiunii, oferind funcționalități pentru salvarea și exportul raportului. Exportatorul de raport gestionează procesul de export al rezultatelor într-un format și la o locație specificate, facilitând documentarea și analiza ulterioară a datelor.

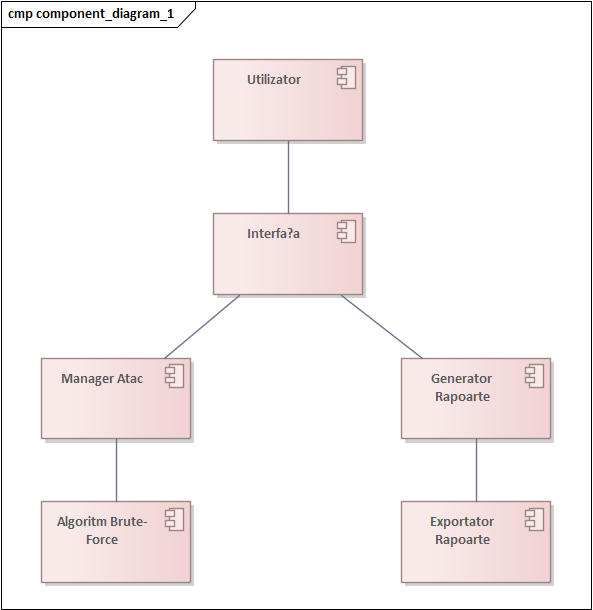


**Figura 2.14 -** Diagrama de clasă pentru procesul de export al raportului

În cadrul figurii 2.14 este reprezentată diagrama de clasă asociată procesului de export al raportului. Componentele și interacțiunile reprezentate în diagramă sunt necesare pentru generarea și salvarea rapoartelor rezultate în urma atacurilor. Utilizatorul inițiază procesul de export și vizualizează raportul generat, interacționând cu managerul de export, care este responsabil de întregul proces, inclusiv inițierea și verificarea stării exportului. Raportul conține detaliile relevante, cum ar fi conținutul și data creării, fiind generat de un component dedicat, generatorul de raport, care colectează informațiile din rezultatul atacului. Exportatorul de fișier gestionează salvarea raportului într-un la o locație specificată, asigurând accesibilitatea și integritatea datelor. Interacțiunile dintre aceste componente asigură un flux coerent și eficient al procesului de export, de la inițierea de către utilizator până la obținerea raportului final, gata pentru analiză și arhivare.

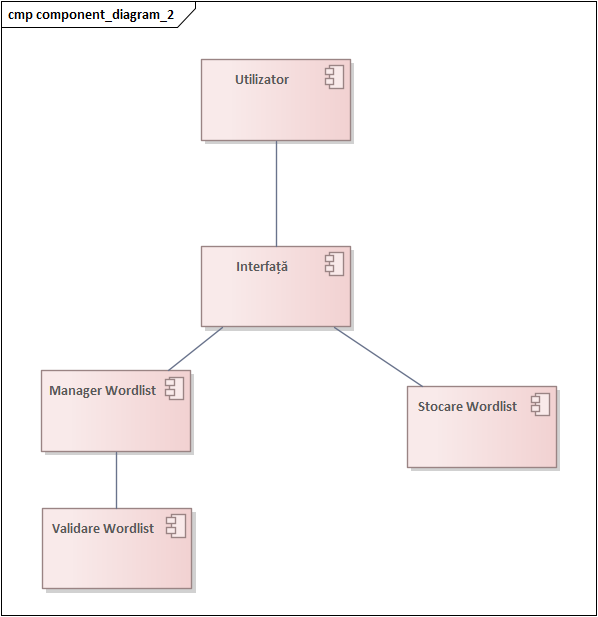
## 2.3.7 Modelarea arhitecturii sistemului prin diagrame de componente

Diagramele de componente oferă numeroase avantaje pentru gestionarea eficientă a proiectelor. În primul rând, acestea asigură o claritate structurală prin reprezentarea vizuală a componentelor și a relațiilor dintre ele, facilitând astfel înțelegerea arhitecturii sistemului de către toți membrii echipei de proiect. În al doilea rând, diagramele de componente permit identificarea și gestionarea dependențelor dintre module, contribuind la detectarea potențialelor probleme legate de interdependențe și promovând un design modular robust. De asemenea, prin delimitarea clară a responsabilităților fiecărei componente, se încurajează reutilizarea acestora în diverse contexte, ceea ce sporește eficiența procesului de dezvoltare. Nu în ultimul rând, aceste diagrame servesc ca instrumente eficiente de comunicare între dezvoltatori, arhitecți și alte părți interesate, asigurând o înțelegere comună și coerentă a soluției propuse.



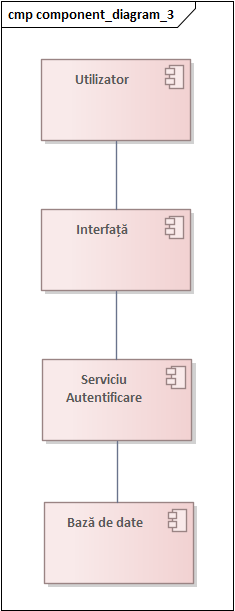
**Figura 2.15 –** Arhitectura sistemului de brute-force

Diagrama de componente din figura 2.15 este pentru procesul de atac brute-force și evidențiază interacțiunile dintre componentele sistemului. Utilizatorul pornește atacul prin intermediul Interfeței Utilizator, care transmite parametrii necesari către Managerul Atac. Acesta coordonează execuția Algoritmului Brute-Force, responsabil pentru generarea și testarea combinațiilor de parole. Rezultatele obținute sunt procesate de Generatorul de Rapoarte și exportate de Exportatorul de Rapoarte, oferind utilizatorului acces la raportul final. Această diagramă clarifică arhitectura modulară a sistemului și relațiile dintre componente, facilitând înțelegerea și optimizarea procesului de atac.



**Figura 2.16 –** Arhitectura pentru validarea wordlist-ului

Diagrama de componente prezentată în figura 2.16 prezintă procesul cd încărcare sau creare a unui wordlist, evidențiind relațiile și dependențele dintre principalele module implicate. Inițial, utilizatorul utilizează interfața pentru a introduce sau selecta un wordlist. Interfața transmite această informație către managerul de wordlist-uri, care e responsabilă de procesul de gestionare. Managerul trimite wordlist-ul către modulul de validare pentru a fi verificat. În cazul în care wordlist-ul este valid, managerul îl stochează în modulul de stocare a wordlist-urilor. Această diagramă subliniază fluxul de date și interacțiunile necesare pentru asigurarea unui management eficient și sigur al wordlist-urilor.​



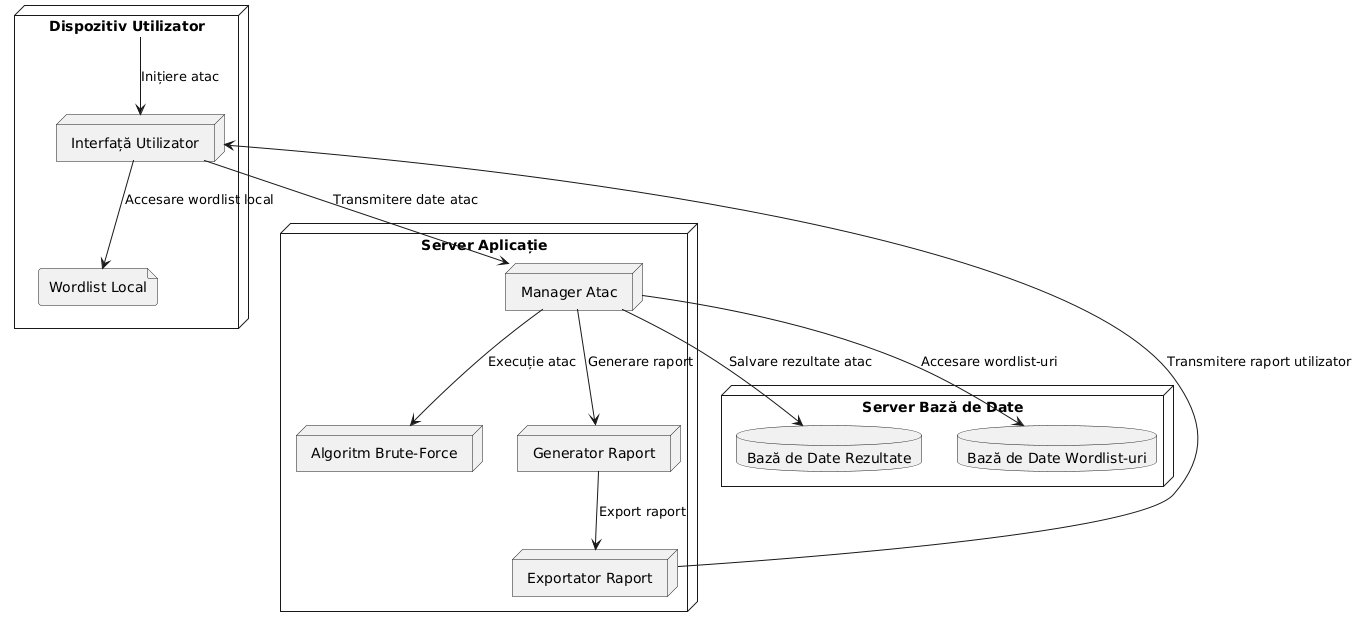
**Figura 2.17 –** Arhitectura sistemului de autentificare

În diagrama din figura 2.17 utilizatorul inițiază procesul de autentificare prin intermediul Interfeței Utilizator (UI), unde introduce credențialele necesare, precum numele de utilizator și parola. Interfața Utilizator transmite aceste informații către serviciul de autentificare, componenta centrală responsabilă cu procesarea cererii de autentificare.​

Serviciul de autentificare validează datele primite după care accesează Baza de Date a Utilizatorilor pentru a compara credențialele furnizate cu cele stocate. Dacă informațiile corespund, Managerul de Autentificare generează un token de sesiune și transmite confirmarea către Interfața Utilizator, semnalând succesul autentificării. În caz contrar, utilizatorul este notificat cu privire la eșecul autentificării și i se oferă opțiunea de a reîncerca.

## 2.3.8 Modelarea implementării aplicației

Diagramele de plasare facilitează înțelegerea și comunicarea modului în care componentele software sunt distribuite și interacționează în cadrul infrastructurii hardware a unui sistem.   
Aceste diagrame pot fi folosite pentru a vizualiza topologia hardware a unui sistem, pentru a descrie componentele hardware utilizate pentru implementare și pentru a arăta relațiile dintre acestea.  
 Unul dintre principalele avantaje ale diagramelor de plasare este de a evidenția interacțiunile dintre componentele software și hardware, oferind o perspectivă clară asupra arhitecturii sistemului. Această claritate contribuie la identificarea și rezolvarea potențialelor probleme legate de performanță, scalabilitate și securitate încă din fazele incipiente ale dezvoltării.



**Figura 2.18 -** Diagrama de plasare a sistemului de atac brute-force

În diagrama din figura 2.18, Dispozitivul Utilizatorului inlcude Interfața Utilizator și poate stoca un Wordlist Local. Utilizatorul inițiază atacul și interacționează cu interfața pentru a configura și lansa procesul. Serverul Aplicație conține componentele principale ale sistemului: Managerul Atac, Algoritmul Brute-Force, Generatorul Raport și Exportatorul Raport, care lucrează împreună pentru a executa atacul și a genera rapoartele corespunzătoare. Serverul Bază de Date stochează Baza de Date Rezultate, unde sunt salvate rezultatele atacurilor, și Baza de Date Wordlist-uri, care conține listele de parole utilizate în procesul de atac. Interacțiunile dintre aceste componente sunt reprezentate prin săgeți, indicând fluxul de date și procesele din cadrul sistemului.

# 3 ELABORAREA SISTEMULUI

Acest capitol descrie modul în care a fost implementată aplicația HashSentinel, concepută pentru analizarea și spargerea valorilor hash folosind metode brute-force și wordlist-uri. Dezvoltarea sistemului s-a bazat pe o combinație de tehnologii selectate în funcție de performanță, simplitate în utilizare și flexibilitate în extindere.

Interfața grafică a fost realizată în Python, folosind biblioteca standard Tkinter. Aceasta oferă o soluție rapidă și eficientă pentru construirea aplicațiilor desktop, fiind potrivită pentru dezvoltarea unei interfețe clare, ușor de folosit, fără dependențe externe complexe. Cu ajutorul Tkinter, utilizatorul poate introduce datele necesare, selecta fișiere, porni și opri procesele, precum și vizualiza rezultatele.

Motorul de procesare a fost implementat în Go (Golang), un limbaj compilat care oferă performanțe ridicate și suport nativ pentru programare concurentă prin goroutine-uri. Acest motor este responsabil de logica principală a aplicației: detectarea algoritmului hash, executarea atacurilor brute-force, gestionarea modului de lucru (secuențial sau din wordlist) și raportarea stării procesului. Limbajul Go a fost ales datorită vitezei sale de execuție, eficienței în utilizarea resurselor și structurii clare a codului.

Comunicarea dintre cele două componente nu implică schimb de date prin fișiere intermediare, ci se realizează direct prin comenzi în linia de comandă. Interfața Python lansează executabilul Go cu parametrii necesari, preluați din opțiunile selectate de utilizator. Această metodă de comunicare permite menținerea modularității și oferă un control clar asupra fluxului de date, fără a introduce timp suplimentar cauzat de salvarea și citirea fișierelor.

Această arhitectură duală cu separarea clară între interfață și procesare face ca aplicația HashSentinel să fie ușor de întreținut, portabilă și pregătită pentru extinderea ulterioară, de exemplu prin integrarea de noi algoritmi sau funcționalități suplimentare de raportare.

## 3.1 Funcționalitatea aplicației

Aplicația HashSentinel oferă o interfață grafică intuitivă, dezvoltată în Python cu ajutorul bibliotecii Tkinter, care permite utilizatorului să configureze și să controleze procesul de spargere a valorilor hash. Interfața permite introducerea valorii hash țintă, selectarea tipului de algoritm de hash (ex. MD5, SHA1, SHA256, SHA512), alegerea modului de atac (brute-force sau wordlist), specificarea setului de caractere, stabilirea lungimii maxime a parolelor, definirea numărului de threaduri și adăugarea de prefixe sau sufixe de tip salt.

În modul de lucru bazat pe wordlist, utilizatorul poate selecta un fișier existent sau genera unul nou pe baza unor reguli personalizabile (transformări de caz, adăugări de prefixe/sufixe, substituții de caractere și combinări). Aceste opțiuni conferă flexibilitate în configurarea atacurilor și permit adaptarea acestora la diverse scenarii de testare.

Controlul execuției este facilitat prin butoane dedicate pentru inițierea, pauzarea, reluarea și oprirea atacului. Comenzile sunt transmise către motorul de procesare Go prin intermediul canalului standard de intrare, asigurând o comunicare eficientă și în timp real între interfață și procesul executabil.

La finalizarea procesului, aplicația generează automat un fișier JSON care conține toate datele relevante ale sesiunii: valoarea hash testată, parola identificată (dacă a fost găsită), metoda de atac, numărul de combinații testate, durata totală, rata medie de procesare și valorile salt utilizate. Pe baza acestui fișier, aplicația creează un raport grafic interactiv care evidențiază vizual progresul și performanța atacului. Graficul include evoluția ratei de procesare în timp și alte statistici utile pentru analiza eficienței procesului de spargere.

Prin aceste facilități, HashSentinel oferă o soluție completă și accesibilă pentru testarea algoritmilor de hashing, combinând o interfață grafică ușor de utilizat cu un motor de procesare performant și flexibil.

## 3.2 Detecția automată a algoritmului de hashing în HashSentinel

În aplicația HashSentinel, această operațiune este implementată printr-o metodă euristică, fundamentată pe potrivirea expresiilor regulate. Astfel, sistemul compară valoarea hash introdusă de utilizator cu o serie de reguli predefinite, descrise într-un fișier extern de configurare rules.json. Această abordare permite extensibilitate, menținere facilă și adaptare rapidă la noi tipuri de hash, fără recompilarea codului sursă.

Fișierul rules.json conține o colecție de obiecte JSON, fiecare corespunzând unui algoritm de hash, fiind definit prin două câmpuri principale: name – care reprezintă denumirea algoritmului și regex – expresia regulată asociată formatului tipic al valorii hash respective.

{ "name": "MD5", "regex": "^[a-fA-F0-9]{32}$" }

La nivel de implementare, regulile sunt modelate în cadrul aplicației printr-o structură de tip HashRule, care include câmpurile Name, Regex și Pattern, ultimul reprezentând versiunea compilată a expresiei regulate. Aceste reguli sunt încărcate la rulare prin funcția loadRulesFromJSON, care parcurge fișierul JSON, decodează obiectele și compilează expresiile regulate pentru eficiență în timpul execuției.

type HashRule struct {

Name string `json:"name"`

Regex string `json:"regex"`

Pattern \*regexp.Regexp

}

După încărcarea regulilor, identificarea propriu-zisă a algoritmului este realizată de funcția DetectHash (diponibilă în Anexa A.1). Pentru fiecare regulă care se potrivește, se calculează un scor pe baza a trei factori: potrivirea expresiei regulate, corespunderea lungimii hash-ului cu lungimea estimată definită în regex și prezența unor prefixe specifice (precum caracterul $, des întâlnit la algoritmi precum SHA512-CRYPT). Fiecare potrivire este asociată unei structuri de tip ScoredMatch, care reține numele algoritmului și scorul atribuit. Lista rezultată este ulterior sortată descrescător în funcție de scor, astfel încât algoritmii cei mai plauzibili sunt returnați primii.

Mecanismul de scor implementat în funcția scoreRule (disponibil în Anexa A.2), care acordă punctaje fixe fiecărui criteriu îndeplinit: 10 puncte pentru potrivirea expresiei regulate, 5 puncte pentru o lungime corectă și 2 puncte pentru prezența unui prefix distinctiv. Estimarea lungimii hash-ului se face prin analiza expresiei regulate a fiecărei reguli, folosind expresii regulate suplimentare care extrag valorile dintre acolade.

De exemplu, pentru un hash precum c94cd960e9a8c9a1c4f8f3b4817fd38e, aplicația va compara acest șir cu expresiile definite în rules.json. Regula MD5 se va potrivi atât ca expresie regulată, cât și ca lungime (32 de caractere). În absența unui prefix special, scorul calculat va fi de 15 puncte. Dacă și alte reguli se potrivesc (de exemplu NTLM, care are un format identic), scorul va fi același, iar ordinea finală este determinată de poziția în listă sau de diferențele subtile dintre regex-uri

## 3.3 Generarea automată de wordlist-uri

Generarea unui fișier wordlist personalizat este altă opțiune a aplicației HashSentinel, oferindu-i utilizatorului posibilitatea de a construi seturi de string-uri adaptate exact nevoilor sale de testare. În loc să se bazeze pe liste predefinite, acesta poate defini propriul set de caractere – litere, cifre sau simboluri – și poate stabili lungimea dorită a combinațiilor, precum și limita maximă de spațiu ocupat pe disc.

Acest proces este accesibil printr-o fereastră integrată în interfața grafică, în care utilizatorul introduce parametrii de generare. Generarea rulează în fundal, astfel încât aplicația rămâne utilizabilă în continuare, iar procesul poate fi oprit oricând, dacă este nevoie. Acest sistem oferă un control asupra conținutului și dimensiunii wordlist-ului, fiind ideal atât pentru experimente rapide, cât și pentru scenarii mai complexe de spargere a parolelor.

Utilizatorul poate specifica intervale de caractere în formate precum a-z sau 0-9, dar și caractere individuale separate prin virgulă (ex: a,b,c,f). La nivel de implementare, aplicația parcurge fiecare segment definit de utilizator și verifică dacă acesta corespunde unui interval (identificabil prin prezența caracterului -). Dacă un segment este de forma x-y, sistemul extrage codurile ASCII corespunzătoare capetelor intervalului folosind funcția ord() și generează toate caracterele din interval prin intermediul funcției range() și chr().

for part in raw\_letters.split(","):

part = part.strip()

if "-" in part:

start, end = part.split("-")

charset += "".join(chr(c) for c in range(ord(start), ord(end) + 1))

elif part:

charset += part

Această metodă asigură generarea corectă și completă a secvențelor, indiferent dacă acestea includ doar o parte din alfabet sau din setul numeric. De exemplu, un input precum a-c,z va produce setul de caractere ['a', 'b', 'c', 'z']. Prin această strategie, sistemul oferă o abordare declarativă, dar expresivă, de definire a spațiului de căutare pentru atacurile brute-force.

După generarea setului de caractere, aplicația construiește combinații de lungimi variabile (în funcție de valorile minime și maxime introduse). Aceste combinații sunt folosite pentru a forma parole compuse în jurul unui cuvânt central introdus – core\_word. În funcție de opțiunile bifate în interfață, fiecare combinație generată poate fi adăugată în poziție de prefix (combinație + core\_word), sufix (core\_word + combinație) sau în ambele poziții simultan (combinație + core\_word + combinație). Astfel sunt construite parole, oferind o eficiență mai mare în scenarii de testare sau spargere controlată.

if use\_prefix and use\_suffix:

line = f"{word}{core\_word}{word}\\n"

elif use\_prefix:

line = f"{word}{core\_word}\\n"

elif use\_suffix:

line = f"{core\_word}{word}\\n"

După generarea fiecărei parole, aplicația nu o scrie imediat pe disc, ci implementează un sistem de bufferizare care optimizează considerabil utilizarea resurselor. Parolele sunt acumulate temporar într-o listă (buffer), iar scrierea fizică se produce doar după atingerea unui prag predefinit de linii – în implementarea curentă, acest prag este de 5000. Astfel, se reduce frecvența operațiilor de intrare/ieșire, se evită fragmentarea la nivel de fișier și se accelerează procesul de generare. Această logică este redată prin următorul fragment:

buffer.append(line)

if len(buffer) >= BUFFER\_SIZE:

current\_file.writelines(buffer)

buffer.clear()

În paralel, aplicația utilizează un sistem de fragmentare automată a fișierelor generate în unități de dimensiune fixă, denumite „chunk-uri”. Fiecare fișier este limitat la maximum 1 GB, iar atunci când această valoare este atinsă, aplicația închide fișierul curent și inițiază automat un fișier nou, incremental. Această strategie este ideală pentru gestionarea unor volume mari de date, facilitând ulterior transportul sau procesarea modulară a wordlist-ului. Codul relevant este:

if chunk\_bytes + line\_size > CHUNK\_MAX\_BYTES:

if buffer:

current\_file.writelines(buffer)

buffer.clear()

current\_file.close()

chunk\_index += 1

current\_chunk\_path = os.path.join(output\_dir, f"chunk\_{chunk\_index:03}.txt")

current\_file = open(current\_chunk\_path, "w", encoding="utf-8")

chunk\_bytes = 0

În paralel cu aceste procese, aplicația monitorizează și volumul total scris, comparându-l constant cu limita maximă setată de utilizator. În cazul în care dimensiunea stabilită este depășită, generarea este oprită automat.

## 3.4 Arhitectura și optimizarea algoritmilor de spargere hash

În cadrul aplicației HashSentinel, algoritmii destinați spargerii hash-urilor sunt implementați în limbajul Go, o alegere justificată prin eficiența execuției, suportului nativ pentru programare concurentă. Arhitectura acestor algoritmi a fost proiectată pentru a fi modulară și extensibilă, iar logica de procesare are trei metode distincte: atacul brute-force, atacul pe bază de wordlist accesat secvențial și atacul pe bază de wordlist încărcat integral în memorie.

La nivel funcțional, toate cele trei metode urmează un model similar: citirea valorii hash introduse, aplicarea unui algoritm de hashing pe o combinație de test, compararea rezultatului și returnarea parolei dacă o potrivire este detectată. Diferențele apar în momentul generarării și testarea combinațiilor, în modul de gestiune a memoriei și în strategia de paralelizare.

Parametrii pe care algoritmii îi primesc pentru a sparge un hash sunt următorii:

* valoarea hash care trebuie spartă;
* tipul algoritmului de hash (ex: MD5, SHA256, SHA512);
* numărul de threaduri care pot fi folosite în paralel;
* opțional, un salt (text adăugat înainte sau după parola testată);
* modul de atac (brute-force sau wordlist);
* alte detalii specifice fiecărui mod, precum charset-ul pentru brute-force sau calea către fișierul wordlist.

Atacul de tip brute-force este unul dintre cele mai simple din punct de vedere conceptual, dar și dintre cele mai costisitoare din punct de vedere al timpului de execuție. El presupune generarea și testarea tuturor combinațiilor posibile de caractere până la o anumită lungime maximă. Pentru fiecare combinație, algoritmul aplică funcția de hashing corespunzătoare și compară rezultatul cu valoarea hash introdusă de utilizator.

Pentru a evita testarea secvențială, care ar co„nsuma mult timp, algoritmul a fost proiectat să ruleze în paralel. Astfel, combinațiile sunt împărțite automat între mai multe fire de execuție, în funcție de numărul de threaduri alocat. Aceste fire rulează în mod concurent și transmit rezultatele către un mecanism central care urmărește dacă a fost găsită o potrivire. Dacă parola corectă este detectată, toate celelalte fire sunt oprite, iar rezultatul este returnat imediat. Această execuție paralelă asigură o scalabilitate bună și o utilizare eficientă a resurselor disponibile pe sistem.

Pentru scenariile în care utilizatorul folosește un fișier care conține posibile parole, aplicația permite folosirea unui atac bazat pe wordlist. În această metodă, fișierul este parcurs linie cu linie, iar fiecare parolă este convertită într-un hash și comparată cu valoarea țintă.

Avantajul principal al acestei metode este acela că nu solicită multă memorie, deoarece datele sunt procesate pe măsură ce sunt citite, fără a fi încărcate integral în RAM. Acest lucru o face potrivită pentru fișiere foarte mari sau pentru utilizatori care folosesc sisteme cu resurse limitate.

Chiar dacă parolele sunt procesate secvențial, verificările sunt totuși distribuite între mai multe fire de execuție, ceea ce îmbunătățește semnificativ viteza de analiză. Algoritmul poate fi o alegere excelentă atunci când fișierul conține parole frecvent întâlnite sau generate pe baza unor reguli relevante pentru contextul de testare.

## 3.5 Mecanismul de control al sesiunii de cracking

Pentru a crește flexibilitatea aplicației HashSentinel și pentru a oferi utilizatorului control asupra gestionării procesului de spargere a hash-ului, au fost implementate funcționalități care permit pauzarea temporară a execuției, reluarea acesteia, oprirea imediată și continuarea unui atac de la punctul în care a fost întrerupt. Aceste opțiuni sunt folositoare în scenarii în care procesul poate dura mai mult timp sau trebuie adaptat la condițiile de rulare ale sistemului.

Atunci când utilizatorul dorește să întrerupă temporar execuția, aplicația intră într-o stare de așteptare activă, în care toate firele de execuție sunt suspendate pentru o perioadă scurtă, dar repetată. Această metodă de pauză nu blochează aplicația și permite o reluare imediată fără pierderea progresului:

func WaitIfPaused() {

for IsPaused() && !IsStopped() {

time.Sleep(200 \* time.Millisecond)

}

}

Această funcție nu întrerupe procesul complet și nu consumă excesiv din resursele sistemului. În schimb, fiecare fir de execuție verifică dacă pauseFlag este activat. Dacă da, atunci acesta „doarme” temporar timp de 200 de milisecunde, după care verificarea se reia. Acest ciclu se repetă până când flagul este dezactivat — adică atunci când utilizatorul reia procesul.

Reluarea efectivă a execuției se face foarte simplu: flagul pauseFlag este setat înapoi pe false, fie prin comanda --resume în linia de comandă, fie prin apăsarea butonului „PAUZĂ” (care devine „RESUME”) în interfața grafică.

Pentru a evita reluarea de la zero a unui atac lung sau complex, aplicația HashSentinel oferă posibilitatea salvării și restaurării sesiunii, permițând reluarea execuției exact din punctul în care a fost întreruptă. Acest lucru este util atunci când procesul a fost oprit voluntar, din lipsă de resurse, sau din alte motive neprevăzute.

Atunci când procesul de cracking este pus manual pe pauză aplicația salvează progresul în format JSON în directoriul restore cu ajutorul structurii de mai jos:

type Session struct {

SessionName string `json:"session\_name"`

FilePath string `json:"file\_path"`

ResumeIndex int `json:"resume\_index"`

Hash string `json:"hash"`

HashType string `json:"hash\_type"`

SaltPrefix string `json:"salt\_prefix"`

SaltSuffix string `json:"salt\_suffix"`

LastUpdated string `json:"last\_updated"`

}

Prin această structură sunt salvate toate informațiile necesare pentru continuarea execuției: calea către fișierul wordlist, indexul până la care s-a ajuns (ResumeIndex), hash-ul țintă, tipul acestuia, precum și eventualele valori de salt.

Fișierul este generat prin apelul funcției SaveSession(), care se ocupă de serializarea structurilor și salvarea într-un fișier corespunzător:

func SaveSession(s Session) {

sessionLock.Lock()

defer sessionLock.Unlock()

s.LastUpdated = time.Now().Format("2006-01-02 15:04:05")

path := getSessionFile(s.SessionName)

data, \_ := json.MarshalIndent(s, "", " ")

\_ = os.MkdirAll("restore", os.ModePerm)

\_ = os.WriteFile(path, data, 0644)

}

Atunci când se dorește reluarea procesului este apelată funcția LoadSession care citește datele din restore.json și setează parametri de reluare a atacului precum ResumeIndex, Salt, Algoritm, Wordlist Path și parametri tehnici de rulare.

Pentru reluarea corectă a unei sesiuni aplicția se asigură că fiecare fir de execuție reia procesul de la punctul exact unde a fost întrerupt. În HashSentinel, acest lucru este realizat printr-un mecanism de incrementare atomică numit resumeIndex. Acesta este implementat cu ajutorul unei variabile atomic.Int32, ceea ce permite accesul sincronizat din mai multe goroutine-uri fără riscul de conflicte sau suprascrieri.

De menționat că ResumeIndex permite algoritmului de cracking să facă skip până la indexul salvat, astfel nu se aplică operații suplimentare asupra cuvintelor și revenirea se face fără pierderi de timp, deoarece asupra cuvintelor până la ResumeIndex nu se aplică operația de hash.

## BIBLIOGRAFIE

[1] “What Is Hashing in Cybersecurity? | CrowdStrike,” CrowdStrike.com. Accessed: Mar. 04, 2025. [Online]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/data-protection/data-hashing/

[2] “https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.” Accessed: Mar. 04, 2025. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[3] V. L. Yisa, M. Baba, and E. T. Olaniyi, “A Review of Top Open Source Password Cracking Tools,” 2016.

[4] “John the Ripper | Hackviser.” Accessed: Mar. 05, 2025. [Online]. Available: https://hackviser.com/tactics/tools/john-the-ripper

[5] “Proiectarea Sistemelor Informatice - CUPRINS CAPITOLUL I 1 Proiectarea sistemelor - Studocu.” Accessed: Mar. 07, 2025. [Online]. Available: https://www.studocu.com/ro/document/universitatea-romano-americana-din-bucuresti/proiectarea-sistemelor-informatice/proiectarea-sistemelor-informatice/3631008

[6] “UX (User Experience) – tot ce trebuie să știi despre proiectarea experienței utilizatorului - Copymate.” Accessed: Mar. 12, 2025. [Online]. Available: https://copymate.app/ro/blog/multi/ux-user-experience-tot-ce-trebuie-sa-stii-despre-proiectarea-experientei-utilizatorului/

[7] “Ce este cerința nefuncțională în ingineria software?” Accessed: Mar. 13, 2025. [Online]. Available: https://www.guru99.com/ro/non-functional-requirement-type-example.html

[8] “Diagrame de interacțiune, colaborare și secvență cu exemple.” Accessed: Mar. 24, 2025. [Online]. Available: https://www.guru99.com/ro/interaction-collaboration-sequence-diagrams-examples.html

[9] D. L. Jisa, “Evaluarea si compararea instrumentelor CASE bazate pe UML,” 2001.

[10] “Communication Diagram | Enterprise Architect User Guide.” Accessed: Mar. 24, 2025. [Online]. Available: https://sparxsystems.com/enterprise\_architect\_user\_guide/17.0/modeling\_languages/communicationdiagram.html

[11] “Tutorial diagramă de clasă UML: Clasă abstractă cu exemple.” Accessed: Mar. 25, 2025. [Online]. Available: https://www.guru99.com/ro/uml-class-diagram.html

[12] “https://cadredidactice.ub.ro/sorinpopa/files/2018/10/L2\_diagrama\_de\_clase.pdf” Accessed: Mar. 25, 2025. [Online]. Available: https://cadredidactice.ub.ro/sorinpopa/files/2018/10/L2\_diagrama\_de\_clase.pdf

# Anexa A

func DetectHash(input string) []ScoredMatch {

if len(rules) == 0 {

jsonPath := filepath.Join(filepath.Dir(os.Args[0]), "rules.json")

\_ = loadRulesFromJSON(jsonPath)

}

matches := []ScoredMatch{}

for \_, rule := range rules {

if rule.Pattern.MatchString(input) {

s := scoreRule(input, rule)

matches = append(matches, ScoredMatch{Name: rule.Name, Score: s})

}

}

sort.SliceStable(matches, func(i, j int) bool {

return matches[i].Score > matches[j].Score

})

return matches

}

**Anexa A.1 –** funcția de detectare a hash-ului

func scoreRule(input string, rule HashRule) int {

score := 0

if rule.Pattern.MatchString(input) {

score += 10

}

if len(input) == expectedLength(rule.Regex) {

score += 5

}

if input[:1] == "$" {

score += 2

}

return score

}

**Anexa A.2 –** funcția de a oferi puncte hash-ului