UNIVERSITATEA DE MEDICINĂ, FARMACIE, ȘTIINȚE ȘI TEHNOLOGIE "GEORGE EMIL PALADE" DIN TÂRGU-MUREŞ

FACULTATEA DE INGINERIE ȘI TEHNOLOGIA INFORMAȚIEI

Specializarea: Informatică Viza facultăţii

Aplicație Web de gestionare a adeverințelor medicale

studențești bazate pe semnături digitale

Coordonator științific Lect. univ. dr. ing. Bogdan Crainicu

Candida: Bălan Sebastian Mihai

Anul absolvirii: 2023

Tema lucrării de licență:

Aplicație Web de gestionare a adeverințelor medicale studențești bazate pe semnături digitale

Problemele principale care vor fi tratate:

* Criptografia asimetrică
* Semnarea digitală a documentelor
* Certificate digitale

Bibliografie recomandată:

* Johannes A. Buchmann. Introduction to Public Key Infrastructures DOI:10.1007/978-3-642-40657-7 ISBN: 978-3-642-40656-0
* John F. Dooley. A Brief History of Cryptology and Cryptographic Algorithms. DOI: 10.1007/978-3-319-01628-3 ISBN: 978-3-319-01628-3
* Henk C. A. van Tilborg, Sushil Jajodia. Encyclopedia of Cryptography and Security. DOI: 10.1007/978-1-4419-5906-5 ISBN: 978-1-4419-5905-8

Termene obligatorii de consultații: Locul practicii:

Primit la data de: Termen de predare:

Semnătura director departament Semnătura conducătorului

Semnătura candidatulu

**UNIVERSITATEA DE MEDICINĂ, FARMACIE, ȘTIINȚE ȘI TEHNOLOGIE “GEORGE EMIL PALADE” DIN TÂRGU-MUREŞ**

**FACULTATEA DE INGINERIE ȘI TEHNOLOGIA INFORMAȚIEI SPECIALIZAREA: INFORMATICĂ**

PROIECT DE DIPLOMĂ

*Aplicație Web de gestionare a adeverințelor*

*medicale studențești bazate pe*

*semnături digitale*

Îndrumător științific: Lect. univ. dr. ing. Bogdan Crainicu

Absolvent: Bălan Sebastian Mihai

Iulie 2023

Contents

[ABSTRACT 5](#_Toc139377427)

[CAPITOLUL 1 6](#_Toc139377428)

[INTRODUCERE 6](#_Toc139377429)

[1.1 Introducere în tematica lucrării 7](#_Toc139377430)

[1.2.Contribuții Personale 8](#_Toc139377431)

[CAPITOLUL 2 9](#_Toc139377432)

[TEHNICI ȘI ALGORITMI DE CRIPTARE 9](#_Toc139377433)

[2.1 Criptografia simetrică 10](#_Toc139377434)

[2.1.1 AES 11](#_Toc139377435)

[2.2.Criptografia asimetrică 14](#_Toc139377436)

[2.2.1 RSA 16](#_Toc139377437)

[CAPITOLUL 3 20](#_Toc139377438)

[INFRASTRUCTURI DE CHEI PUBLICE 20](#_Toc139377439)

[3.1 Schimbul de chei Diffie-Hellman 20](#_Toc139377440)

[3.2 Funcții Hash criptografice 22](#_Toc139377441)

[3.2.1 SHA-2 22](#_Toc139377442)

[3.2.2 MD5 23](#_Toc139377443)

[3.2.3 Salting 24](#_Toc139377444)

[3.3 Semnături digitale 25](#_Toc139377445)

[3.4 Autoritatea de certificare 26](#_Toc139377446)

[3.5 Certificate X.509 28](#_Toc139377447)

[3.6 ASP.NET Core MVC 31](#_Toc139377448)

[3.7 Entity Framework Core 34](#_Toc139377449)

[3.8 ASP.NET Identity 36](#_Toc139377450)

[3.9 BouncyCastle API 37](#_Toc139377451)

[3.10 Modelul *Repository* 37](#_Toc139377452)

[CAPITOLUL 4 39](#_Toc139377453)

[PROIECTAREA APLICAȚIEI 39](#_Toc139377454)

[4.1 Cerințe 39](#_Toc139377455)

[4.1.1 Înregistrarea securizată 39](#_Toc139377456)

[4.1.2 Autentificarea securizată 40](#_Toc139377457)

[4.1.3 Invitarea utilizatorilor 40](#_Toc139377458)

[4.1.4 Separarea utilizatorilor folosind roluri 41](#_Toc139377459)

[4.1.5 Gestionarea situațiilor școlare 42](#_Toc139377460)

[4.1.6 Fluxul adeverințelor 42](#_Toc139377461)

[4.1.7 Implementarea funcțiilor criptografice 43](#_Toc139377462)

[4.2 Proiectarea bazei de date 43](#_Toc139377463)

[CAPITOLUL 5 45](#_Toc139377464)

[IMPLEMENTAREA APLICAȚIEI 45](#_Toc139377465)

[5.1 Implementarea înregistrării și autentificării securizate 45](#_Toc139377466)

[5.2 Implementarea invitării utilizatorilor 47](#_Toc139377467)

[5.3 Implementarea rolurilor 49](#_Toc139377468)

[5.4 Implementarea gestionării situațiilor școlare 50](#_Toc139377469)

[5.5 Implementarea fluxului adeverințelor 52](#_Toc139377470)

[5.6 Implementarea clasei *CertificateAuthority* 58](#_Toc139377471)

[5.6.1 Generarea cererilor de semnare a certificatului (CSR) 58](#_Toc139377472)

[5.6.2 Generarea certificatului folosind CSR 59](#_Toc139377473)

[5.6.3 Semnarea digitală a datelor 61](#_Toc139377474)

[5.6.4 Verificarea semnăturilor digitale 62](#_Toc139377475)

[5.6.5 Arhivarea cheii private 63](#_Toc139377476)

[5.7 Implementarea clasei *MailManager* 63](#_Toc139377477)

[5.8 Implementarea modelului *Repository* 64](#_Toc139377478)

[5.9 Implementarea bazei de date 66](#_Toc139377479)

[CAPITOLUL 6 68](#_Toc139377480)

[CONCLUZII 68](#_Toc139377481)

[REFERINȚE 69](#_Toc139377482)

# ABSTRACT

This paper presents the development of a secure platform for issuing and managing medical certificates in a university environment. The platform caters to four types of users: students, professors, system administrators and doctors from the university’s medical clinic. The aim is to provide a seamless, easy-to-use system through which medical personnel can issue medical certificates, digitally signed using their private key, which is obtained after completing the registration process. After issuing said certificates, they are to be approved by a system administrator and the student in question, after which they are made available to professors, which can choose whether to honor the request or not.

The University and, by default, the professors have at their disposal a reliable way to verifythe authenticity and integrity of a digital medical certificate; in addition, the administration of student attendance, affected by personal health problems, can be seamlessly integrated within the University management platform of student activity.

The implementation of such a platform addresses the challenges of privacy, data protection, and secure file transfer within a university environment. It contributes to streamlining the process of issuing medical certificates, significantly reducing administrative burden, and enhancing efficiency. The security of the solution presented in this paper is assured by the implementation of several cryptographic processes including, but not limited to, the digital signing of documents and the verification of such signatures.

Overall, the development of this platform contributes to the modernization and improvement of existing processes within a university, providing a robust and user-friendly way for issuing and managing digital medical certificates, alongside a secure and strict policy for approving said certificates.

# CAPITOLUL 1

## INTRODUCERE

Criptografia este o ramură a matematicii și informaticii care se ocupă cu securizarea comunicațiilor dintre două sau mai multe entități prin transformarea informațiilor într-o formă ilizibilă pentru oricine, în afară de destinatarul dorit. Scopul criptografiei este protejarea datelor cu caracter confidențial și prevenirea accesului terțelor neautorizate la aceste informații. Având în vedere faptul că mai mult de 5 miliarde de persoane utilizează internetul, criptografia este un aspect extrem de important în securizarea comunicațiilor.

Criptografia poate fi împărțită în două categorii principale: criptografie simetrică și criptografie asimetrică (sau bazată pe chei publice). Criptografia simetrică se referă la utilizarea aceleiași chei atât pentru criptarea datelor, cât si pentru decriptarea acestora. Pe de altă parte, criptografia asimetrică utilizează două chei diferite – o cheie publică pentru criptare, si o cheie privată pentru decriptare sau vice-versa. În contextul criptografiei asimetrice, este posibilă stabilirea unui canal securizat de comunicare dintre entități care nu au avut vreun contact anterior. De asemenea, criptografia asimetrica încapsulează concepte vitale securității cibernetice din ziua de azi, cum ar fi semnăturile digitale, fără de care confidențialitatea și autenticitatea informațiilor transmise prin intermediul Internetului nu ar exista, cel puțin nu în forma complexă si avansată de care dispunem în prezent.

În cadrul criptografiei bazată pe chei publice, cheia privată nu trebuie împărtășită cu nimeni – aceasta, așa cum sugerează și numele, trebuie să rămână exclusiv în posesia entității care a inițiat generarea. Prin urmare, administrarea acestora este de mare importanță. Pe de altă parte, cheia publică, deși are o legătură matematică cu cea privată, poate fi împărtășită cu oricine, deoarece cunoașterea acesteia nu poate duce, într-un timp rezonabil, la determinarea cheii private asociată cu aceasta. Cheile private trebuie păstrate într-o locație securizată, de exemplu pe un modul de securitate hardware (HSM), iar accesul la acestea trebuie sa fie monitorizat. În cazul în care o cheie privata este compromisă, trebuie revocată imediat și este necesară generarea unei noi perechi de chei. Pentru înțelegerea mai profundă a sistemelor informatice ce incorporează criptografia asimetric, lucrarea va aborda implementarea acestor concepte într-un mod ușor de înțeles, într-un context real.

Un aspect important ce trebuie luat în considerare în vederea construirii unei soluții ce are la bază concepte de criptografie asimetrică este verificarea autenticității cheilor publice, care poate fi realizată prin certificatele digitale de autenticitate. Aceste certificate sunt folosite pentru a autentifica identitatea unei entități și conțin date care duc la identificarea deținătorului certificatului, cum ar fi numele, adresa si cheia publica. Aceste informații sunt verificate de o terță de încredere, cunoscută sub numele de autoritate de certificare, înainte ca certificatul să fie emis. În cele mai multe cazuri, certificatele sunt semnate de o autoritate, însă există cazuri în care entitatea care generează certificatul o să fie si utilizatorul acestuia.

### 1.1 Introducere în tematica lucrării

O parte esențială într-o infrastructură bazată pe chei publice este asigurarea autenticității cheilor publice si ale datelor transmise. Acest lucru poate fi realizat cu ajutorul certificatelor digitale, numite si certificate de autenticitate.

Certificatele digitale sunt niște documente electronice care ajută la verificarea identității unei entități. Acestea sunt folosite pentru schimbul de date în mod securizat, în mediul online. Încrederea într-un astfel de sistem este dată de încrederea în autoritatea care emite aceste certificate reprezentând o legătura dintre cheile publice și entitățile de care aparțin. Autoritatea emițătoare se numește Autoritate de Certificare (*Certificate Authority*) și are rolul de a verifica identitatea entităților care solicită un certificat, după care, în cazul în care identitatea a fost validată și individul este eligibil pentru un certificat, acesta este emis și transmis către beneficiar. În continuare, vom prezenta niște elemente esențiale în compunerea unui certificat. [1]

1. Numele și informațiile deținătorului: numele persoanei, organizației, sau dispozitivul căruia i se eliberează certificatul, precum și informațiile de contact aferente.

2. Cheia publică: cheie criptografică unică, utilizată pentru criptarea si decriptarea datelor. Aceasta este inclusă în certificat și este utilizată de alte părți pentru a cripta datele trimise deținătorului certificatului, sau pentru a verifica o semnătură digitală.

3. Perioada de validitate: perioada în care certificatul este considerat valid, exprimata printr-o dată a emiterii si o dată a expirării.

4. Numele entității care a emis certificatul: identificator folosit pentru recunoașterea autorității emițătoare.

5. Număr de serie (*Serial Number*): Identificator unic atribuit de către autoritatea de certificare. Nu există două certificate semnate de aceeași autoritate cu același număr de serie.

6. Algoritmul criptografic folosit.

Acesta este doar un exemplu de componente minime ce trebuie să fie prezente în cadrul unui certificat, însă pot exista mai multe elemente ce însumează câmpurile acestuia. De obicei, emițătorul unui astfel de certificat este altul decât cel care îl va folosi, însă există excepții în cazul cărora emițătorul este același cu utilizatorul, iar acestea se numesc *S*elf-*Signed Certificates*. Acestea sunt deseori utilizate pentru testarea și dezvoltarea de software, eficientizând procesul deoarece nu este nevoie de eliberarea unui certificat nou de către o autoritate abilă în acest sens de fiecare data când este nevoie de unul. Prin urmare, utilizarea acestora nu este fiabila în cadrul soluțiilor aflate în producție, deoarece nu sunt considerate sigur, nefiind semnate de o terță considerată de încredere, atât de utilizatori cât si de *Web Browsers* sau sisteme de operare.

### 1.2.Contribuții Personale

Lucrarea prezintă o soluție implementabilă în cadrul unei instituții de învățământ, care are în vedere facilitarea motivării absențelor studenților într-un mod rapid și sigur. În acest context, certificatele digitale pot fi folosite pentru semnarea digitală a adeverințelor de către un medic din cadrul cabinetului universitar, astfel garantând autenticitatea si integritatea acestora.

Scopul acestei lucrări este transmiterea unei adeverințe către profesori în vederea motivării absențelor. Aceasta este semnată digital de către un medic aparținând cabinetului universitar și trebuie aprobată de către un administrator de sistem și studentul beneficiar. Dacă a fost aprobată de ambele părți, adeverința ajunge la profesori, care pot decide dacă motivează sau nu absențele.

# CAPITOLUL 2

# TEHNICI ȘI ALGORITMI DE CRIPTARE

Criptografia este o ramură a matematicii a cărei implementări (într-un mod relativ primitiv) datează în jurul anilor 1900 înainte de Hristos, în mormântul lui Khnumhotep II, în Egipt. Scribul a folosit niște hieroglife neobișnuite în locul celor comune. Scopul nu a fost ascunderea mesajului, ci modificarea acestuia într-o formă care s-ar prezenta ca fiind impunătoare. [7] O formă mai recentă și cunoscută în zilele noastre este cifrul Caesar care, după cum sugerează numele, a fost folosit de către Julius Caesar al Romei în jurul anilor 100 înainte de Hristos. Acesta este un simplu cifru de substituție în care fiecărei litere din alfabet îi corespunde altă literă din alfabet, aflată la o distanță definită de entitatea care realizează implementarea acestuia.

Totuși, în contextul actual, avem nevoie de algoritmi de criptare mai puternici și care urmăresc anumite principii bine definite. În continuare, vor fi prezentate câteva principii vitale în construcția unui algoritm criptografic sigur.

1. Autentificare: Originea mesajului poate fi verificată, astfel receptorul poate valida identitatea emițătorului. În consecință, furtul de identitate este prevenit.

2. Integritate: Datele primite de către receptor trebuie să fie identice cu cele trimise de către emițător, adică mesajul trebuie să fie imun la alterare pe tot parcursul comunicației. Acest lucru poate fi asigurat de către funcțiile criptografice Hash.

3. Non Repudiere: Emițătorul unui mesaj nu poate nega trimiterea acestuia. Acest lucru poate fi asigurat prin utilizarea semnăturilor digitale.

4. Confidențialitate: Informația trebuie să rămână privată și securizată. Tehnici de criptografie, precum criptarea, pot fi folosite în acest sens pentru alterarea informațiilor, făcându-le ilizibile pentru oricine fără cheia potrivita pentru decriptarea acestora. [8]

5. *Key Management*: Se referă la procesul de generare, distribuire si gestionarea cheilor criptografice. Implementarea unui sistem de gestionare potrivit este esențială pentru securitatea unui sistem criptografic, deoarece securitate unei infrastructuri de acest tip depinde de discreția cheii. [8]

Criptografia are două mari ramuri, criptografia simetrică și cea asimetrică. În continuare vom dezbate diferența dintre cele două și proprietățile relevante lucrării.

## 2.1 Criptografia simetrică

Criptografia simetrică este o ramură a criptografiei care folosește o cheie secretă atât pentru criptarea datelor cât și pentru decriptarea cifrului aferent. În cadrul unei infrastructuri ce are la bază criptografia simetrică, entitățile ce vor să stabilească un canal de comunicație securizat trebuie să aleagă o cheie comună, pe care o vor păstra secretă.

În contextul criptografiei simetrice, emițătorul criptează mesajul folosind cheia secretă înainte de a-l trimite, iar receptorul folosește aceeași cheie secretă pentru decriptarea datelor primite. Astfel, informațiile pot fi citite doar de către entitățile care au acces la cheia secretă folosită în cadrul criptării. (Fig. 1)

A diagram of keys connected to a document

Description automatically generated with low confidence

Fig. 1 – Ciclu de operații în criptografia simetrică

În consecință, păstrarea cheii secrete în siguranță este un aspect fundamental, deoarece dacă un atacator are acces la aceasta, descifrarea comunicațiilor se dovedește a fi trivială, iar în consecință comunicațiile viitoare și cele anterioare nu mai pot fi considerate sigure. În cazul în care cheia secretă este compromisă, trebuie generată una nouă și împărtășită în mod securizat.

Criptografia simetrică este o metodă sigură de comunicare cât timp cheia secretă este păstrată în siguranță. În contrast cu criptografia asimetrică, aceasta este mai rapidă, deci și viabilă în cadrul transmiterii unui volum mare de date. Dezavantajul, totuși, este că un atacator poate intercepta cheia secretă în tranzit, în cazul în care mijlocul de comunicare este compromis, lucru ce nu se aplică în cadrul criptografiei asimetrice. [9]

Există multe aplicări ale criptografiei simetrice, precum în cadrul comunicării client-server HTTPS pentru criptarea datelor transmise. De asemenea, criptarea parolelor stocate într-o bază de date este deseori realizată folosind o cheie secretă, iar când utilizatorul realizează o autentificare, parola introdusă de către acesta este criptată cu cheia secretă și comparată cu valoarea stocată în baza de date. Două importante sisteme criptografice ce folosesc criptografia simetrică sunt:

1. DES (*Data Encryption Standard*): Nu este un algoritm sigur, deoarece cheia de 56 de biți este considerată prea mică. Totuși, a fost un algoritm ce a avut o mare influență în progresul criptografiei. [5]
2. AES (*Advanced Encryption Standard*): Cunoscut și sub numele Rijndael, este considerat un algoritm sigur, deci momentan nu s-au găsit vulnerabilități în cadrul acestuia.

### 2.1.1 AES

Datorită faptului că DES nu era considerat sigur, dat fiind lungimea redusă a cheii, 3DES (*Triple Data Encryption Standard*) a fost proiectat pentru a rezolva această problemă, dar dezavantajul a fost faptul că era considerat prea lent. AES este cel mai popular algoritm simetric de criptare. Este mai rapid și mai puternic decât 3DES și este considerat un algoritm sigur. Algoritmul are o mărime a blocului de date de 128 de biți și o mărime a cheii de 128, 192 sau 256 de biți.[1] AES folosește o rețea de substituție-permutare (*substitution-permutation network*) cu mai multe runde. O cheie de 128 de biți impune 10 runde, o cheie de 192 de biți 12 runde, iar una de 256 de biți 14 runde. Fiecare rundă necesită o cheie de rundă (*Round Key*), dar dat fiind faptul că o singură cheie este dată algoritmului, aceasta trebuie să fie expandată (*Key Expansion*) pentru a obține câte o cheie pentru fiecare rundă, inclusiv runda 0. Fiecare rundă constă în 4 pași.[10]

**Pașii din cadrul rundelor AES:**

1. Substituirea de biți: Primul pas este înlocuirea biților din blocul de date bazată pe reguli dictate de S-boxes (*Substitution Boxes*) predefinite. (Fig. 2)

Diagram

Description automatically generated

Fig. 2 Substituirea de biți[10]

1. Deplasarea rândurilor (*Shift Rows*): Fiecare rând, în afară de primul, este deplasat cu o poziție, așa cum este prezentat în (Fig. 3)

Diagram

Description automatically generated

Fig. 3 Deplasarea rândurilor[10]

1. Amestecarea coloanelor (*Mix Columns*): Este folosit cifrul Hill pentru a amesteca mesajul și mai mult folosind coloanele blocului. (Fig. 4)

Diagram

Description automatically generated

Fig. 4 Amestecarea Coloanelor[10]

1. Diagram

   Description automatically generatedAdăugarea cheii de rundă: Se face XOR (sau exclusiv) între mesaj și cheia de rundă. (Fig. 5)

Fig. 5 Adăugarea cheii de rundă[10]

În cadrul criptării AES, în ultima rundă se omite pasul adăugării cheii de rundă.

## 2.2.Criptografia asimetrică

Criptografia asimetrică, cunoscută și sub denumirea de criptografie bazată pe chei publice, este o ramură a criptografiei care, spre deosebire de criptografia simetrică, folosește două chei diferite pentru criptare și decriptare. În cadrul unui sistem specializat în acest sens, fiecare utilizator are acces la o pereche de chei: o cheie publică și una privată. Cheia publică, după cum implică și numele, poate fi împărtășită cu oricine, și este folosită pentru criptarea mesajelor destinate deținătorului cheii private. Pe de altă parte, cheia privată trebuie să rămână secretă și este folosită pentru decriptarea mesajelor care au fost criptate cu cheia publică corespunzătoare. Un astfel de sistem este fezabil deoarece se poate iniția un canal de comunicație securizat între două entități aflate la distanță, deoarece distribuirea cheilor publice într-un mediu mai puțin sigur nu reprezintă un impediment, iar decriptarea informațiilor necesită cheia privată a destinatarului, deci în consecință datele transmise, în cazul în care sunt interceptate, nu sunt inteligibile de către atacator.

Securitatea unei astfel de infrastructuri este bazată pe dificultatea computațională impusă pentru derivarea cheii private din cheia publică. Acest lucru înseamnă că, în cazul în care un atacator are acces la cheia publică, calcularea cheii private este imposibilă. În cazul în care cheia privată a oricărei părți a fost descoperită, este necesară generarea unei noi perechi de chei.

Criptografia asimetrică are o varietate de aplicații în cadrul soluțiilor ce necesită comunicare sigură, autentificare, sau pur și simplu securitatea datelor. De exemplu, acest tip de criptografie este folosită în cadrul aplicațiilor de tip VPN (*Virtual Private Network*), fiind utilizată pentru crearea unor conexiuni securizate între utilizatori și servere, protejând datele confidențiale de interceptare și acces neautorizat.

În general, se consideră mai fezabilă folosirea cheilor simetrice, deoarece se reduce utilizarea de resurse computaționale. Se recomandă folosirea sistemelor bazate pe chei publice atunci când datele transmise au o mărime relativ mică sau în cazul în care este necesar un canal securizat de comunicație intre entitățile participante într-un schimb de date. [9]

**Exemplu de comunicare folosind criptarea asimetrica:**

Alice vrea să îi transmită un mesaj lui Bob. Bob împărtășește cheia sa publică cu Alice, după care Alice folosește această cheie pentru a cripta mesajul dorit. Alice trimite mesajul către Bob, iar acesta îl decriptează folosind cheia sa privată. Astfel, Alice a transmis către Bob un mesaj în mod confidențial. (Fig. 6)

A picture containing screenshot, diagram, design

Description automatically generated

Fig. 6 – Ciclu de operații în criptografia asimetrică

În continuare vom prezenta câteva sisteme criptografice care folosesc la bază perechi de chei.

1. RSA (Rivest-Shamir-Adleman): Cheia publică este generată bazat pe două numere prime mari, împreună cu o valoare auxiliară arbitrară. Numerele prime sunt păstrate secrete. Mesajele pot fi criptate de oricine, dar pot fi decriptate doar de către entitatea care cunoaște numerele prime. [4]
2. DSA (*Digital Signature Algorithm*): Se bazează pe conceptul matematic de exponențiere modulară și problema logaritmului discret. DSA este o variantă a schemelor de semnare Schnorr și ElGamal. [1]
3. ECC (*Eliptic-curve cryptography*): Se bazează pe structura algebrică a curbelor eliptice pe câmpuri finite. În cadrul ECC se permite o mărime mai redusă a cheii comparabil cu algoritmii care nu sunt bazați pe curbe eliptice pentru o securitate echivalentă. [4]

### 2.2.1 RSA

Implementarea RSA are la bază utilizarea aritmeticii modulare, teoremei lui Euler și a funcției Totient a lui Euler. În RSA cheia publică este generată prin înmulțirea a două numere prime mari. Pentru un sistem informatic, calcularea produsului a două numere prime mari este trivială, însă factorizarea în numere prime a unui număr mare este foarte complicată, necesitând o putere computațională imensă și un timp îndelungat. Cu cât numerele alese sunt mai mari, cu atât factorizarea produsului este mai dificilă. [11]

În continuare, vom exemplifica modul de funcționare teoretic al algoritmului RSA.

1. Alegem două numere prime mari (***p*** și ***q***). Acestea trebuie să fie suficient de mari încât determinarea lor să fie aproape imposibilă, dar generarea lor se dovedește a fi eficientă folosind testul de primariate. Numerele alese trebuie să fie similare în magnitudine, dar diferite în lungime pentru a oferi o securitate sporită. Securitatea algoritmului stă în siguranța numerelor prime alese, în consecință este imperativă păstrarea acestora într-un mediu securizat, deoarece aflarea acestora îi oferă unui atacator posibilitatea de a intercepta și înțelege informațiile criptate folosind cheia aferentă. Pe de altă parte, în cazul în care numerele prime nu sunt aflate, singura opțiune la dispoziția atacatorului este căutarea exhaustivă de numere prime, lucru care, în cazul în care regulile de alegere a numerelor prime sunt respectate, este nefezabil din punct de vedere computațional. De dragul exemplului, ne vom abate de la aceste reguli și vom alege niște numere prime mai mici decât ar trebui și având aceeași lungime (***p*** = 60, ***q*** = 50). Calculăm ***n* = *p*\**q*** și obținem valoarea pentru ***n*** = 3000. În cadrul acestui algoritm, valoarea **n** este folosită ca modul pentru cheia publică și cea privată. Lungimea acesteia este deseori exprimată în biți și este numită **lungimea cheii**. valoarea lui **n** este parte din cheia publică și este împărtășită în domeniul public. [13]

2. Pasul următor este să calculăm funcția totient a lui Carmichael (*Carmichael’s totient function*) notată ***λ(n)****.* Dat fiind faptul că ***n***= ***p***\****q***, ***λ(n)*** = lcm(***λ(p)*,** ***λ(q)***). lcm reprezintă cel mai mic multiplu comun (*lowest common multiple*) și se poate calcula folosind algoritmul lui Euclid. Având în vedere că **p** și **q** sunt valori prime, ***λ(p)*** = ***φ(p)*** = ***p-1***, iar analog ***λ(q)*** = ***φ(q)*** = ***q-1***, deci prin urmare ***λ(n)*** = LCM(***p-1***, ***q-1***). În cadrul ecuației, ***φ*** reprezintă **funcția totient a lui Euler** (*Euler’s totient function*). ***λ(n)*** este o valoare ce trebuie să rămână secretă.

3. Acum, este necesar să alegem o valoare întreagă ***e***, astfel încât 2 < ***e*** < ***λ(n)*** și gcd(***e***, ***λ(n)***) = 1 (gcd – *greatest common divisor* /cel mai mare divizor comun) . Pe scurt, ***e*** și ***λ(n)*** sunt co-prime. Dacă valoarea lui ***e*** are o lungime a biților scăzută, rezultă o criptare mai eficientă. Cea mai comună valoare aleasă pentru ***e*** este ***216 + 1 = 65,537***. Cea mai mică (și eficientă) valoare posibilă atribuibilă lui ***e*** este ***3***, dar o valoare atât de mică s-a dovedit a fi mai puțin sigură în anumite circumstanțe. [12]

4. În continuare, este necesară valoarea multiplicativei modulare inverse (*modular multiplicative inverse*)a lui ***e modulo λ(n)***, notată și ***d*** atunci când ***d ≡ e-1(mod(λ(n)***. Practic, este necesară rezolvarea ecuației ***d\*e = 1(mod(λ(n))***. Valoarea lui ***d*** poate fi calculată folosind varianta extinsă a algoritmului lui Euclid deoarece, datorită valorilor co-prime ale lui ***e*** si ***λ(n),*** ecuația este o formă a identității lui Bézout (lema lui Bézout), unde ***d*** este unul dintre coeficienți. (Fig. 7) Valoarea lui ***d*** este păstrată secretă și reprezintă exponentul cheii private. [6]

Graphical user interface, text, application

Description automatically generated

Fig. 7 Lema lui Bézout[14]

Astfel, urmărind pașii anteriori, putem genera o pereche de chei ce pot fi folosite în cadrul algoritmului RSA. Distribuirea cheilor publice necesită un canal sigur, dar nu neapărat secret, deoarece cheia privată nu poate fi, în mod fezabil, derivată din cheia publică împărtășită. Să presupunem că Bob dorește să îi transmită niște informații lui Alice în mod sigur. Dacă aceștia decid să folosească RSA, Bob trebuie să cunoască cheia publică a lui Alice pentru a cripta mesajul, iar Alice trebuie să folosească cheia privată aferentă pentru a decripta mesajul primit. (Fig. 8)

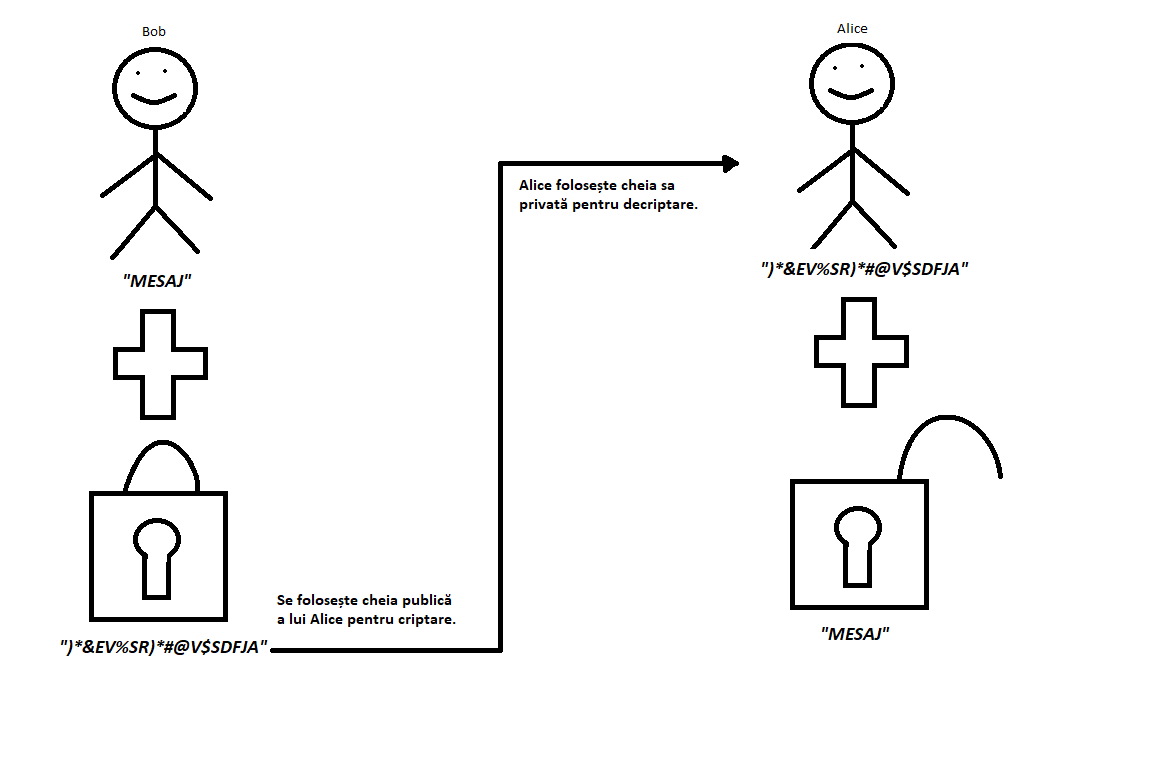


Fig. 8 Reprezentare ciclu criptare și decriptare

De obicei, pentru criptarea mesajelor se utilizează valoarea ASCII (*American Standard Code for Information Interchange*) aferentă fiecărui caracter din mesajul inițial. Criptarea este realizată folosind formula ***mesaj\_criptat = caracter\_ASCII^e modulo n*** (caracter\_ASCII se referă la fiecare caracter din cadrul mesajului inițial, nealterat, convertit în ASCII. Valorile ***e*** și ***n*** sunt valorile calculate în momentul generării cheilor pereche). Valorile rezultate din ecuația reprezentată anterior sunt trimise către recipientul ales. Mesajul criptat mai poartă și denumirea de *ciphertext*.

Decriptarea se realizează folosind cheia privată a receptorului și mesajul primit în format criptat. Se bazează pe formula ***mesaj\_decriptat = mesaj\_criptat\*d mod n***. mesaj\_decriptat reprezintă valoarea ASCII ce reprezintă caracterul în format decriptat. mesaj\_criptat este mesajul primit, criptat folosind cheia publică. ***d*** și ***n*** sunt valori calculate anterior, în momentul generării cheilor. Valoarea rezultată din rezolvarea ecuației este mesajul decriptat și mai poartă denumirea de *plaintext*.

Pentru a-i permite lui Bob să transmită mesaje criptate, Alice îi transmite acestuia cheia sa publică (cheia formată din valorile aflate anterior, ***n*** și ***e***) printr-un mediu sigur. [6] Astfel, având acces la cheia publică generată de Alice, Bob poate cripta mesajul cu aceasta și îl poate trimite către Alice printr-un mediu public, fără riscul de a expune informații confidențiale unui atacator. Alice, după primirea mesajului criptat, folosind cheia privată legată matematic de cea utilizată în momentul criptării, poate decripta mesajul în forma sa originală.

În continuare, vom reprezenta un exemplu de aplicare al algoritmului RSA, folosind valori prime nepotrivite în cazul unei aplicări reale, dar viabile în sensul exemplificării modului de funcționare.

1. Alegem două numere prime distincte, precum ***p = 61*** și ***q = 53***.

2. Calculăm ***n = p\*q***, de unde rezultă că ***n = 3233****.*

3. Calculăm funcția totient a lui Carmichael ***λ(n) = lcm(p-1, q-1)***, obținând rezultatul ***λ(3233) = lcm(60, 52) = 780***.

4. Alegem o valoare e pentru care ***1 < e < 780*** și e este co-prim cu ***780***. Alegând o valoare primă pentru ***e***, ne rămâne doar să demonstrăm că ***e*** nu este un divizor al lui ***780***. În cazul de față, o valoare potrivită este ***e = 17***.

5. Calculăm valoarea lui ***d***, multiplicativei modulare inverse al lui ***e(mod λ(n))***, și obținem ***d = 413***, deoarece ***1 = (17\*413)mod 780***.

După parcurgerea acestor pași, obținem cheia publică (***n = 3233, e = 17***) și cheia privată (***n = 3233, d = 413***). Pentru a exemplifica modul în care funcționează criptarea și decriptarea, vom alege valoarea ***m = 65***.

Pentru criptare avem formula ***mesaj\_criptat = 6517 mod 3233***, care are rezultatul ***2790***.

Pentru decriptare avem formula ***mesaj\_decriptat = (mesaj\_criptat)413 mod 3233***, care are rezultatul ***65***.

# CAPITOLUL 3

# INFRASTRUCTURI DE CHEI PUBLICE

O infrastructură bazată pe chei publice (*PKI – Public Key Infrastructure*) este un mediu folosit pentru crearea și gestionarea certificatelor digitale. [15] În esență, se urmărește transferul de informații în contextul Internetului într-un mod sigur prin intermediul serviciilor precum trimiterea de *mail*-uri, cumpărăturile online și *internet banking*-ul. În cadrul unei infrastructuri ca aceasta, este posibilă crearea, folosirea, stocarea, gestionarea, distribuirea și revocarea certificatelor, precum și gestionarea criptării bazate pe cheie publică. PKI-urile sunt fundația stabilirii unei comunicații securizate pe o rețea, fiind o abordare des întâlnită în zilele noastre. [16] Datorită faptului că într-o infrastructură de genul transmiterea cheii private nu este necesară, a fost rezolvată problema distribuirii cheilor prin intermediul unor canale nesecurizate. Totuși, într-o infrastructură bazată pe chei publice, este necesară existența unei terțe (autoritate de certificare) care să verifice identitatea entităților, dar acest lucru presupune ca utilizatorii să aibă încredere într-o astfel de entitate.

## 3.1 Schimbul de chei Diffie-Hellman

Schimbul de chei Diffie-Hellman este o metodă matematică sigură de a împărtăși chei criptografice prin intermediul unui canal public (deci nesecurizat) și a fost unul dintre primele protocoale integrate în infrastructurile bazate pe chei publice. În mod tradițional, comunicarea în mod sigur se baza pe schimbul de chei într-un mod privat, însă această metodă, dezvoltată de Whitfield Diffie și Martin Hellman, permite stabilirea unui secret comun prin intermediul unui canal de comunicație nesecurizat. Această metodă poate fi folosită în cadrul sistemelor bazate pe criptografia simetrică. În continuare, vom prezenta un scenariu teoretic de implementare al schimbului de chei Diffie-Hellman.

1. Persoanele ce doresc să inițieze comunicarea cad de comun acord asupra a două valori: ***p*** (numit și modulul) și ***g*** (o rădăcină primitivă a lui ***p***). Aceste valori nu sunt secrete, în consecință pot fi împărtășite prin intermediul unui canal considerat nesigur.

2. Fiecare entitate alege, de asemenea, o valoare secretă care nu va fi comunicată nimănui, Fiecare dintre ei calculează o valoare, conform formulei ***valoare\_împărtășită =*** ***g^valoare\_secretă mod p***. În formula precedenta ***valoare\_secretă*** reprezintă valoarea aleasă de fiecare participant în inițializarea comunicației. Aceștia, după generare, fac schimb de valori rezultate.

3. Ultimul pas este stabilirea secretului comun, lucru făcut posibil prin utilizarea valorii obținute anterior și folosind formula :

***cheie\_secretă = valoare\_împărtășită^valoare\_secretă mod p***

După parcurgerea acestor pași, cele două entități împart un secret comun. (Fig. 9)

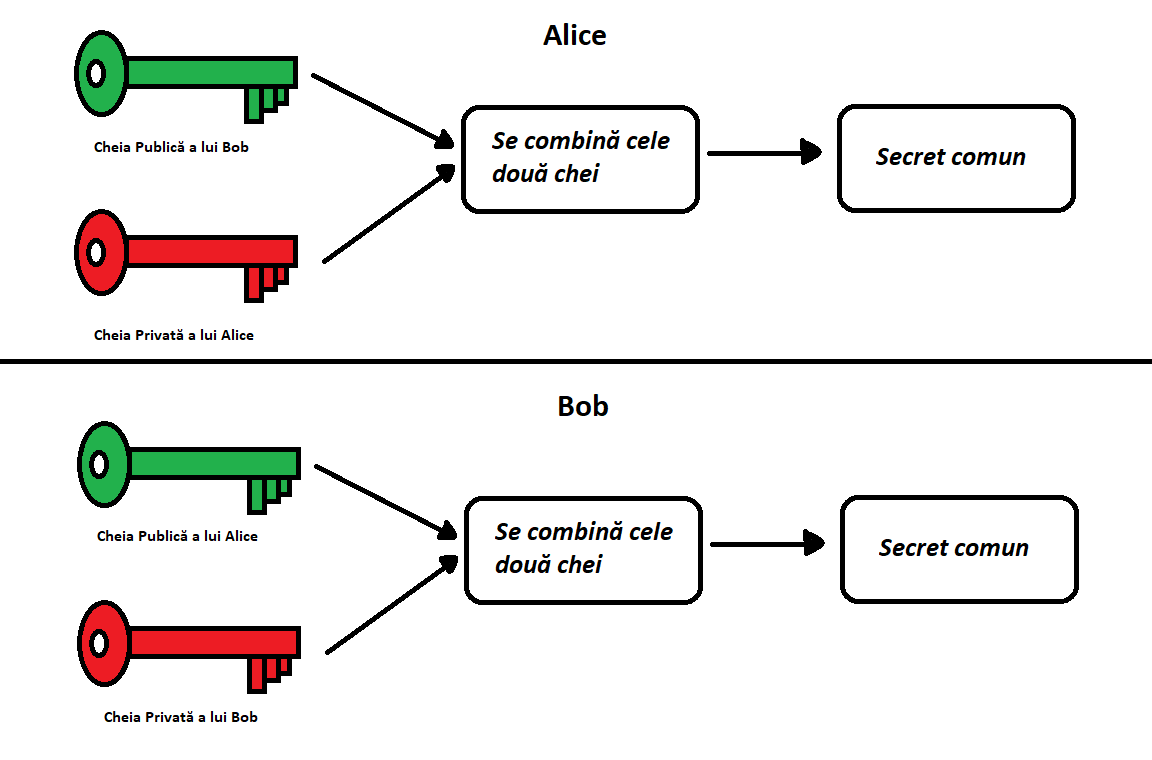


Fig. 9 Stabilirea secretului comun în Diffie-Hellman

Astfel, folosind cheile generate anterior, se poate stabili un canal de comunicare securizat. De exemplu, dacă Alice dorește să îi transmită un mesaj lui Bob, aceasta v-a folosi cheia publică a lui Bob pentru criptare, iar Bob, în cazul în care dorește decriptarea datelor primite, poate să folosească cheia sa privată în acest sens. O adaptare a acestui algoritm pentru sistemele criptografice asimetrice o reprezintă RSA, algoritmul de criptare prezentat anterior, în 2.2.1.

## 3.2 Funcții Hash criptografice

Funcțiile Hash criptografice au la bază niște algoritmi matematici care primesc ca date de intrare informații, iar scopul lor este să producă ca date de ieșire un șir de caractere denumit valoare hash, care, în aparență, nu are nici un sens sau vreo legătură cu datele de intrare. Funcțiile de acest tip sunt proiectate în așa fel încât calcularea unei astfel de valori este ceva trivial, dar este dificil, sau chiar imposibil, să se determine datele de intrare folosind o valoare hash dată. (Fig. 10) Aceste funcții au o varietate de aplicații în contexte actuale, ele fiind folosite ca mecanisme de securitate în cadrul semnăturilor digitale, pentru verificarea integrității datelor sau pentru stocarea parolelor sau a datelor confidențiale într-o bază de date, într-un format indescifrabil.

Aceste funcții criptografice sunt la bază simple funcții Hash, dar cu niște proprietăți adiționale necesare în cadrul unui context criptografic.

A diagram of a blockchain

Description automatically generated with low confidence

Fig. 10 Utilizarea funcțiilor hash criptografice

### 3.2.1 SHA-2

SHA-2 (*Secure Hash Algorithm*) este un set de funcții hash criptografice proiectat de Agenția de Securitate Națională din Statele Unite ale Americii, publicat inițial in 2001. Acești algoritmi sunt proiectați folosind construcția Merkle–Damgård, o metodă de construire a unor funcții hash criptografice rezistente la coliziuni din funcții de compresie unidirecționale rezistente la coliziuni. [6] Ideea de funcție de compresie unidirecțională se referă la o funcție ce transformă două șiruri de date de intrare într-un singur șir de date de ieșire, iar cuvântul cheie unidirecțional se referă la dificultatea calculării datelor de intrare bazat pe datele de ieșire. [17]

Familia de algoritmi criptografici SHA-2 constă în 6 funcții hash cu valori hash având o lungime de 224, 256, 384 sau 512 biți.[6] SHA-2 este implementat într-o mulțime de aplicații și protocoale, precum TLS și SSL, PGP, SSH, S/MIME și IPsec. De asemenea, suita de algoritmi își are locul și în contextul criptomonedelor. De exemplu, aceștia sunt folosiți în cadrul *blockchain*-ului Bitcoin, unde au rol în calcularea *proof of work* (numit și minat, implică o răsplată pentru rezolvarea problemei)și *proof of stake* (procesarea tranzacțiilor și crearea de noi blocuri în contextul *blockchain*-ului).

### 3.2.2 MD5

MD5 (*message-digest algorithm*)este o funcție hash des întâlnită în ziua de azi care are ca scop producerea unei valori hash cu o lungime de 128 de biți. Funcția a fost propusă de către Ronald Rivest în anul 1991 cu scopul de a înlocui funcția hash MD4,[1] care prezenta o securitate sever compromisă.[18] MD5 poate fi folosit pentru a genera sume de control cu scopul de a verifica integritatea datelor. Algoritmul a fost folosit în scop criptografic în trecut, dar s-a descoperit că suferă de mai multe vulnerabilități. Totuși, acesta rămâne folosit și în ziua de azi în scopuri ce nu țin de criptografie. [1]

Una dintre utilizările MD5 este stocarea de parole. (Fig. 11) Acest algoritm poate fi folosit pentru a stoca în baza de date un hash unidirecțional al parolei, care mai apoi poate fi comparat cu cel generat în momentul autentificării. Acest lucru nu este recomandat, datorită vulnerabilităților descoperite de analiști, dar totuși se găsesc implementări de genul în cadrul unor aplicații reale. O pătrime din sistemele de gestionare a conținutului încă folosesc MD5 pentru stocarea parolelor. Totuși, securitatea acestui algoritm este îmbunătățită atunci când se folosește conceptul de **salting**.

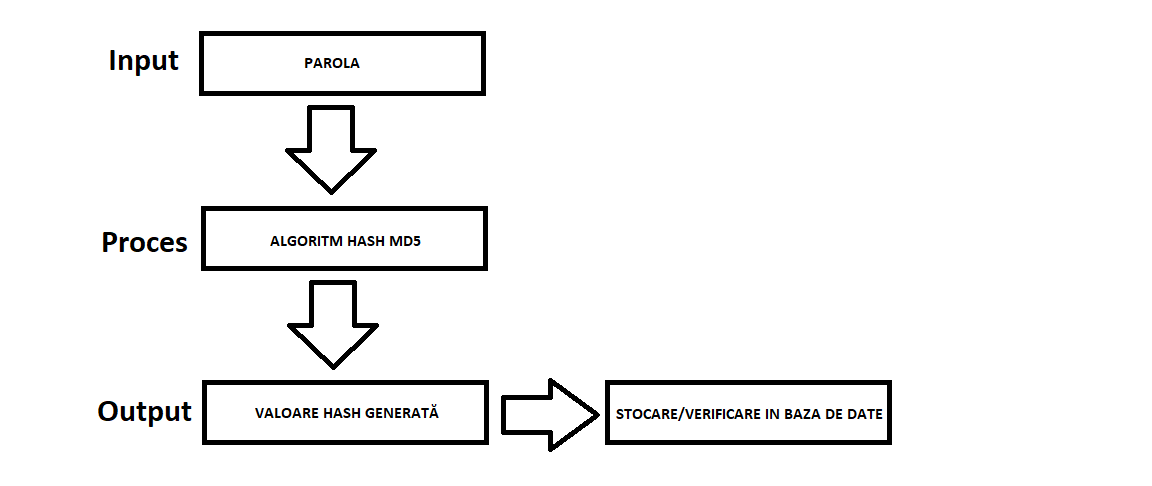


Fig. 11 Exemplu de hashing pentru o parolă

### 3.2.3 Salting

Conceptul de *salting* se referă, în contextul criptografiei, la adăugarea unui șir pseudo-random la datele de intrare ale unei funcții hash pentru a garanta o valoare de ieșire unică (adică hash) chiar dacă datele de intrare sunt identice. Spre exemplu, dacă nu se folosește salting în cadrul algoritmului de hashing, doi utilizatori al unui serviciu ce necesită autentificare ar putea să aibă aceeași parolă, iar în consecință valorile hash stocate în baza de date vor fi identice. În cazul în care se obține accesul la baza de date și la parola ce aparține unuia dintre utilizatori, toți ceilalți utilizatori sunt compromiși deoarece au aceeași parolă, iar acest lucru este făcut evident de hash-ul stocat. [19] (Fig. 12)

Graphical user interface, text, application

Description automatically generated

Fig. 12 O parolă identică duce la un hash identic

Prin urmare, în cazul în care se dorește folosirea *hashing*-ului pentru stocarea parolelor, este vitală folosirea *salting* pentru o securitate exponențial sporită. Astfel, se evită valori hash identice pentru parole identice, iar un atacator nu mai poate face legătura între două persoane care au aceeași parolă. (Fig. 13)

Text

Description automatically generated

Fig. 13 Valorile hash sunt diferite chiar dacă parolele sunt identice

## 3.3 Semnături digitale

O semnătură digitală este o valoare criptografică folosită tot mai des în contextul societății actuale. Aceasta poate fi folosită pentru semnarea documentelor în format electronic și are, din punct de vedere legal, aceeași valoare, în cazul în care este emisă de o autoritate de încredere, iar implementarea este corectă. Un algoritm de semnare digitală permite două operații distincte:

1. Operația de semnare, în cadrul căreia se utilizează o cheie pentru a produce o semnătură asupra datelor de intrare.

2. Operația de verificare, unde semnătura poate fi validată de către o terță ce nu are nici o cunoștință asupra cheii utilizate pentru semnare.[20]

În contrast cu semnătura clasică, adică cea fizică, semnătura digitală oferă avantajul că persoana care semnează nu poate nega în viitor acest fapt. Acest lucru este cunoscut în spațiul criptografiei, sub denumirea de **non-repudiere**. De asemenea, semnarea digitală a datelor permite verificarea integrității datelor semnate. (Fig. 14)

A picture containing text, diagram, plan, screenshot

Description automatically generated

Fig. 14 Semnarea datelor și verificarea semnăturii

Pentru a exemplifica funcționarea unui algoritm ce are ca scop semnarea digitală a informațiilor, vom prezenta un scenariu trivial. Alice vrea să-i transmită lui Bob un contract semnat digital. Aceasta folosește propria cheie privată pentru a semna datele (reprezentate, de exemplu, prin hash-ul unei imagini) după care îi trimite lui Bob datele semnate împreună cu documentul, alături de cheia publică. Bob poate să verifice semnătura dacă, prin intermediul unui algoritm specializat, introduce documentul, semnătura și cheia publică aparținând lui Alice.

## 3.4 Autoritatea de certificare

O autoritate de certificare are rolul de a elibera certificate digitale, adică un document care să ateste identitatea unei entități.

În general, o entitate ce dorește să aplice pentru un certificat digital o să genereze o pereche de chei (o cheie privată și o cheie publică), împreună cu o cerere de semnare a certificatului (*CSR*). O cerere de acest fel este reprezentată de un fișier de tip text care conține cheia publică, alături de alte informații precum domeniul, organizația, adresa de email. Perechea de chei si cererea de semnare sunt, de obicei, generate pe sistemul de calcul unde certificatul urmează a fi instalat. Cheia privată rămâne, după cum sugerează și numele, privată, în consecință aceasta trebuie păstrată în maximă siguranță. După generarea cererii de semnare, solicitantul o trimite către autoritatea de certificare, care verifică în mod independent corectitudinea datelor, iar dacă nu există probleme în acest sens, semnează digital certificatul cu o cheie privată și îl trimite mai departe, către solicitant. [21] (Fig. 15)

În momentul în care certificatul semnat este prezentat unei părți terță (cum ar fi o persoană ce dorește să viziteze website-ul entității ce deține certificatul), aceasta poate să verifice autenticitatea semnăturii autorității de certificare folosind cheia publică aparținând acesteia.

De asemenea, partea terță poate să folosească certificatul pentru a confirma faptul că datele semnate au fost trimise de către cineva în posesia cheii private corespondente, iar informația nu a fost alterată din momentul semnării. [21]

A picture containing text, diagram, screenshot, jack

Description automatically generated

Fig. 15 Procesul de eliberarea al unui certificat digital

În concluzie, autoritatea de certificare reprezintă o entitate în care lumea își pune încrederea, iar compromiterea acesteia ar fi un lucru dezastruos. Acest lucru ar duce la pierderea încrederii în autoritate în cel mai bun caz și la furtul de date și identitate în cel mai rău caz. Această situație nu este una ipotetică, având în vedere că unele servicii de genul au fost compromise, iar urmările au fost destul de serioase. (Fig. 16)

DigiNotar, o autoritate de certificare, a suferit un atac cibernetic care a dus la falimentul acesteia. În cadrul atacului, au fost generate sute de certificate fraudulente pentru sute de website-uri, incluzând Google și Skype. În momentul în care incidentul a fost făcut public, guvernul olandez și dezvoltatorii de *browsers* au luat măsuri pentru a limita impactul atacului, însă se crede că au fost emise certificate fraudulente timp de aproape două luni până la raportarea incidentului. [23]

A close-up of a certificate

Description automatically generated with low confidence

Fig. 16 Atacuri asupra autorităților de certificare de-a lungul timpului [22]

## 3.5 Certificate X.509

În contextul criptografiei, X.509 este un standard care definește formatul certificatelor bazate pe chei publice. Aceste certificate sunt folosite în diferite protocoale din cadrul Internetului, inclusiv TLS/SSL, care stă la baza HTTPS, protocolul sigur de navigare pe web. Aceste certificate pot fi folosite și în aplicații offline, sub forma de semnături electronice. Un certificat X.509 face legătura dintre identitatea unei entități și cheia publică. Acesta poate sa fie semnat de o autoritate de certificare, sau poate fi auto-semnat (*self-signed*). Când un astfel de certificat este semnat de către o autoritate de certificare, sau validat prin alte moduri, deținătorul certificatului poate folosi cheia publică pentru a stabili un canal de comunicare sigur cu o parte terță, sau poate valida documente semnate digital folosind cheia privată. [1]

Există mai multe câmpuri ce sunt prezente în cadrul unui certificat X.509. În continuare, vom menționa câteva dintre acestea împreună cu imagini pentru referință.

1. Detalii despre deținător. Sunt prezente date precum locația și numărul de serie. (Fig. 17)

Table

Description automatically generated with low confidence

Fig. 17 Detalii despre deținător[24]

2. Detalii despre emițător. Sunt prezente date precum numele, locația, cu privire la autoritatea de certificare care a emis certificatul. (Fig. 18)

A picture containing table

Description automatically generated

Fig. 18 Detalii despre autoritatea de certificare[24]

3. Numărul de serie, versiunea, algoritmul folosit pentru semnare, perioada de validitate a certificatului. (Fig. 19)

Graphical user interface, text, application

Description automatically generated

Fig. 19 Proprietăți certificat[24]

4. Cheia publică, semnătura și alte date asociate precum algoritmul utilizat și valorile variabilelor utilizate în cadrul acestuia pentru generarea semnăturii. (Fig. 20)

Text

Description automatically generated

Fig. 20 Cheie publică, semnătură, algoritmul utilizat[24]

O autoritate de certificare are posibilitatea de a revoca certificate. Acest lucru poate fi realizat din mai multe motive printre care se numără: expirarea certificatului, încălcarea termenilor si condițiilor, compromiterea deținătorului.

## 3.6 ASP.NET Core MVC

În vederea dezvoltării s-a ales utilizarea ASP.NET Core folosind *design pattern*-ul MVC (*Model-View-Controller*). În vederea stabilirii unei infrastructuri potrivite s-au luat în considerare mai multe aspecte. ASP.NET a îndeplinit unele cerințe de bază, precum nivelul ridicat de performanță și posibilitatea de scalabilitate pe viitor. De asemenea, ASP.NET facilitează o integrare ușoară a unor aspecte de securitate esențiale într-o aplicație de acest fel, iar la baza proiectului este o comunitate mare și activă de dezvoltatori software, fiind un proiect *open-source*. În general se preferă utilizarea unor soluții *open-source* unde este posibil, deoarece sunt, în cele mai multe instanțe, mult mai protejate împotriva vulnerabilităților, iar riscul existenței unui *backdoor* este mult mai redus decât în cazul soluțiilor brevetate. Câteva dintre aspectele de securitate ale căror implementare a fost facilitată de către ASP.NET includ:

**- Protejare împotriva *Cross-Site Request Forgery* (CSRF):** În mod implicit, toate cererile de tip HTTP sunt protejate împotriva CSRF prin generarea unor *token*-uri unice pentru fiecare cerere (*POST/PUT/CRUD*) care conține date ce trebuie protejate și stocarea acestora în *cookie*-uri. [25]

**- Securizarea Comunicațiilor:** Aplicația a fost securizată cu ajutorul unui certificat SSL generat local în cadrul aplicației. (Fig. 21) Generarea acestuia a fost inițiată în mod automat la prima rulare a serverului de către *Visual Studio*. În consecință, comunicarea client-server este criptată și protejată împotriva atacurilor de tipul *man-in-the-middle*. Atacurile *man-in-the-middle* se referă la un tip de atac cibernetic în cadrul căruia comunicația dintre două entități este interceptată (și posibil alterată) fără cunoștința acestora. Existența unui certificat SSL garantează că, chiar dacă există o entitate ce interceptează comunicațiile dintre client și server, datele sustrase sunt indescifrabile și prin urmare inutile.

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 21 Certificatul SSL asociat aplicației

MVC este un *design pattern* care are ca scop de bază separarea funcționalităților aplicației în trei componente distincte: *Model-View-Controller.* Considerând fiecare element în mod individual, fiecare îndeplinește un rol anume:

1. ***Model:*** Reprezintă datele și logica aplicației. Încapsulează structura datelor, definește reguli pentru manipularea și accesarea acestora și efectuează operații precum validarea datelor și interacțiuni cu baza de date. Această componentă nu depinde de celelalte două, prin urmare poate funcționa în mod independent. [3]

2. ***View:***  Are scopul de a prezenta interfața utilizatorului. Afișează datele din *Model* într-un format vizibil plăcut. Este o componentă pasivă, în sensul că nu conține logică, fiind orientată exclusiv pe afișarea datelor și tratarea interacțiunilor efectuate de către utilizator. De asemenea, un utilizator poate trimite date către *Controller* prin intermediul *View-*ului, cu scopul de a facilita procesarea ulterioară a acestora.

3. ***Controller:*** Reprezintă un intermediar între *View* și *Model*. Primește date de intrare de la utilizator prin intermediul *View*-ului și le traduce în comenzi pentru *Model*. *Controller-*ul, așadar, are ca scop primar coordonarea fluxului de date dintre *Model* și *View*, asigurând independența dintre cele două. [3]

A picture containing diagram, text, technical drawing, plan

Description automatically generated

Fig. 22 Fluxul de date într-o infrastructură MVC

Există o sumedenie de motive pentru care utilizarea unei astfel de paradigme ar fi potrivită în cadrul soluției. Câteva dintre aspectele esențiale luate în vedere în efectuarea acestei decizii sunt:

1. **Separarea funcțiilor:** Prezentată anterior, această separare ușurează dezvoltarea, înțelegerea și modificarea fiecărei componente.

2. **Organizarea codului și reutilizabilitate:** MVC obligă la o abordare structurată de organizare a codului. Prin definirea clară a responsabilităților fiecărei componente se promovează ideea de reutilizare a codului.

3. **Scalabilitate:** Cu cât complexitatea unei aplicații crește, cu atât se observă mai bine beneficiile asociate cu paradigma MVC. Având o aplicație bine compartimentată, adăugarea unor noi funcționalități sau modificarea unor funcționalități existente este mai ușoară și nu afectează alte părți ale aplicației. Astfel, se facilitează menținerea codului și reduce semnificativ riscul de efecte secundare în momentul implementării unor noi funcționalități.

Astfel, utilizarea ASP.NET MVC asigură o organizare structurată în dezvoltarea unor soluții de complexitate înaltă, promovând reutilizabilitate, separarea funcțiilor și scalabilitate. Aceste aspecte se dovedesc a fi relevante în contextul dezvoltării aplicației aferente acestei lucrări.

## 3.7 Entity Framework Core

*Entity Framework* este un *framework* ce simplifică accesul și gestionarea bazelor de date, facilitând interacțiunile dintre dezvoltatori software și baze de date folosind concepte de programare orientată pe obiect. Există mai multe motive pentru care s-ar putea opta pentru utilizarea acestui concept, iar câteva dintre acestea sunt:

1. **Productivitate și dezvoltare rapidă:** Permite dezvoltatorilor software să se concentreze pe logica aplicației, lăsând în seama Entity Framework interacțiunile *low-level* cu baza de date. Oferă un API pentru sarcini de bază asupra bazelor de date, precum inserare, actualizare, ștergere și interogare. Această abstractizare duce la productivitate mărită și un ciclu de dezvoltare al aplicației mai rapid.

**2.**  **Integrare cu LINQ:** Entity Framework se integrează foarte bine cu LINQ (*Language-Integrated Query*), un limbaj capabil de interogare ce face parte din suita .NET. LINQ oferă o sintaxă intuitivă, ușor de citit și înțeles pentru interogarea și manipularea datelor. Dezvoltatorii software pot scrie interogări LINQ, iar *framework*-ul le va traduce mai departe în interogări SQL eficiente.

A picture containing text, screenshot, diagram, design

Description automatically generated

Fig. 23 Utilizarea Entity Framework în cadrul aplicației

De asemenea, având în vedere faptul că am ales dezvoltarea folosind ASP.NET, mai există câteva beneficii oferite de interacțiunile dintre acesta și Entity Framework:

1. **Interogări cu ASP.NET:** Entity Framework se integrează bine cu ASP.NET, facilitând cu ușurință accesul la baze de date în cadrul aplicației. Permite integrări cu facilități ale ASP.NET precum validarea și *data binding*.

**2.** **Dezvoltare Code-First:** Entity Framework facilitează dezvoltarea de tip *code-first*, care se referă la faptul că prima dată sunt stabilite clasele asociate entităților, iar schema bazei de date este generată ulterior în mod automat folosind aceste clase. (Secvența de cod 1) Această abordarea simplifică procesul de dezvoltare, având în vedere faptul că baza de date este generată în mod automat.

public class Cheie

{

[Key]

public int Id { get; set; }

[Required]

public string CheiePublica { get; set; }

[Required]

public DateTime StartDate { get; set; }

[Required]

public DateTime EndDate { get; set; }

}

Secvența de cod 1 Implementarea clasei Cheie

## 3.8 ASP.NET Identity

ASP.NET *Identity* este un sistem proiectat pentru a trata autentificarea, autorizarea și gestionarea utilizatorilor din cadrul unei aplicații web. Prin utilizarea acestuia, gestionarea înregistrărilor și autentificărilor este ușurată. De asemenea, *Identity* implementează în mod automat mai multe concepte de securitate, cum ar fi *hashing-*ul și *salting-*ul parolelor. De asemenea, este facilitată și autentificarea pe bază de *cookies* și accesul la resurse pe bază de funcție (*role-based authentication).* Aceste facilități au fost integrate în cadrul aplicației pentru a asigura un mediu sigur pentru stocarea datelor private și accesarea resurselor într-un mod securizat și controlat. În momentul implementării, ASP.NET creează în mod procedural tabelele necesare funcționării extensiei *Identity.* (Fig. 24)

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 24 Schema bazei de date generată de ASP.NET Core Identity [26]

Câteva aspecte notabile din cadrul ASP.NET *Identity* sunt:

1. ***UserManager:*** O componentă principală din ASP.NET *Core Identity*. Permite gestionarea utilizatorilor prin facilitarea acțiunilor precum crearea utilizatorilor, ștergerea acestora, schimbarea parolelor, setarea rolurilor.

2. ***RoleManager:*** Folosit pentru gestionarea rolurilor din cadrul aplicației. Acestea sunt folosite pentru a controla accesul la diverse resurse. *RoleManager* permite crearea, modificarea și ștergerea rolurilor, precum și asocierea dintre utilizatori și roluri.

3. ***User:*** Încapsulează datele utilizatorului. Include proprietăți precum email, *hash* parolă, numele de utilizator și alte date relevante.

## 3.9 BouncyCastle API

API-ul *BouncyCastle* reprezintă o suită de algoritmi criptografici implementați în C#. În cadrul aplicației, a fost folosit pentru implementarea operațiilor criptografice necesare pentru îndeplinirea unui flux de date corect și sigur. Au fost implementate operații precum generarea de certificate, generarea de cereri de semnare, semnarea digitală și verificarea semnăturilor. Implementarea acestor operații este abordată in capitolul 5.6.

## 3.10 Modelul *Repository*

Modelul *Repository* este utilizat pentru a oferi abstractizarea stratului de persistență a datelor. În esență, acesta oferă o modalitate de a centraliza logica de acces la date și o separă de logica *business* (cum ar fi cea declarată în *controller*) a aplicației. [27] Deci, un *repository* se comportă ca un intermediar între aplicație și baza de date aferentă acesteia, încapsulând conceptele de interogare, creare, ștergere și actualizare a datelor și conferind o interfață curată și consistentă pentru interacțiunile efectuate în cadrul aplicației. Există mai multe motive în favoarea utilizării acestui model, iar câteva dintre acestea, luate în considerare în vederea dezvoltării soluției, sunt:

1. **Separarea funcțiilor:** Beneficiu ce poate fi observat și în modelul MVC. Prin separarea logicii aferente interacțiunii cu baza de date obținem o separare clară a funcțiilor aplicației, prin urmare codul scris este mai curat și ușor de înțeles.

2. **Reutilizabilitatea codului:** Funcțiile declarate într-un *Repository* pot fi folosite în diferite părți ale aplicației, astfel se reduce cantitatea de cod redundant și repetat, promovând ideea de reutilizare a codului. Similar, același beneficiu este observat și în paradigma MVC.

# CAPITOLUL 4

# PROIECTAREA APLICAȚIEI

## 4.1 Cerințe

Este necesar un dispozitiv cu acces la internet, care suportă *web-browsers* moderne. Deoarece o calitate de bază a soluției dezvoltate este securitatea, trebuie proiectat un sistem de înregistrare și autentificare securizat. În cadrul aplicației există 4 tipuri de utilizatori: studenți, profesori, administratori de sistem și doctori din cadrul cabinetului medical al universității. În funcție de categoriile (rolurile) de utilizatori menționate anterior, soluția pune la dispoziție o sumedenie de funcționalități pentru persoanele autentificate. Funcționalitatea de bază este emiterea, semnarea, validarea și fluxul unei adeverințe medicale în cadrul soluției, în consecință este necesară implementarea unor procese specializate în acest sens. În plus, datorită faptului că generarea cheilor private, utilizate pentru semnarea digitală a adeverințelor de către medicii din cadrul cabinetului universitar, are loc pe serverul aplicației în momentul înregistrării, este imperativ să asigurăm atât stocarea, cât și transmiterea securizată a acestora către utilizatori.

### 4.1.1 Înregistrarea securizată

Funcționalitatea de înregistrare este pusă la dispoziție doctorilor invitați în cadrul cabinetului medical al universității. Deoarece în cadrul formularului de înregistrare este necesară introducerea unei parole, este necesară stocarea acesteia într-un format criptat. Acest lucru poate fi îndeplinit folosind funcții criptografice de *hashing* împreună cu o valoare *salt* pentru fiecare parolă.

A picture containing text, diagram, screenshot, parallel

Description automatically generated

Fig. 25 Diagrama de secvență pentru procesul de înregistrare

### 4.1.2 Autentificarea securizată

Având în vedere faptul că în urma procesului de autentificare este permis accesul la resurse de mare importanță, trebuie să asigurăm o implementare securizată a acestuia. Acest lucru este făcut posibil prin folosirea *ASP.NET Core Identity*, ce permite autentificarea pe bază de mail și parolă, sesiunea fiind menținută prin intermediul *cookies.*

### 4.1.3 Invitarea utilizatorilor

Pentru medici, aplicația facilitează un proces de înregistrare simplificat. Acesta începe prin depunerea unei cereri formale la universitate, în mod fizic. Prezența fizică este necesară pentru a asigura identitatea utilizatorului ce dorește să se înregistreze. După depunerea cererii, în cazul în care este aprobată, procesul de înregistrare poate fi continuat prin intermediul portalului online, folosind un cod de înregistrare trimis pe adresa de e-mail a viitorului utilizator de către un administrator de sistem.

### 4.1.4 Separarea utilizatorilor folosind roluri

A picture containing diagram, text, sketch, circle

Description automatically generatedÎn cadrul soluției, accesul la anumite resurse și funcționalități este condiționat de rolul utilizatorului autentificat. Separarea utilizatorilor pe bază de roluri este facilitată de utilizarea *ASP.NET Core Identity,* Administratorii de sistem au posibilitatea de a gestiona înregistrările de pe platforma, invita noi medici în cabinetul universității, adăuga noi specializări și de a gestiona adeverințele transmise prin intermediul platformei. Studenții au posibilitatea de a vedea propriile situații școlare și de a gestiona adeverințele emise în numele acestora. În ceea ce privește profesorii, aceștia au acces la catalogul online, unde pot ajusta situația școlară a studenților înrolați în cursurile predate de aceștia. Tot în catalog, aceștia au posibilitatea de a vedea, pentru fiecare student individual, adeverințele trimise, după ce au fost aprobate de către studenți și administratori. În cazul doctorilor, facilitatea oferită este trimiterea unor adeverințe, semnate digital cu o cheie privată generată de server-ul aplicației, în vederea motivării absențelor pentru oricare student membru al platformei.

Fig. 26 Diagrama use-case pentru fiecare tip de utilizator

### 4.1.5 Gestionarea situațiilor școlare

Abilitatea de a gestiona situațiile școlare este accesibilă exclusiv profesorilor. Fiecare profesor are acces la situația fiecărui student înrolat la cursurile acestuia și poate ajusta prezențele, notele și onora adeverințele, dacă consideră potrivit acest lucru. Fiecare absență motivabilă în baza unei adeverințe trimisă prin intermediul aplicației trebuie evidențiată în acest sens.

### 4.1.6 Fluxul adeverințelor

Trimiterea adeverințelor și ciclul de viață al acestora în sistem reprezintă o funcționalitate de bază în cadrul aplicației. În consecință, această caracteristică trebuie proiectată cu atenție la detalii și cu securitatea în minte. Adeverințele sunt trimise de către doctorii din cadrul cabinetelor, și semnate digital cu cheia privată primită după aprobarea cererii de înregistrare. Aceste adeverințe trebuie aprobate atât de către administrator, care poate verifica dacă există vreo greșeală în acestea (de exemplu o dată greșită sau un interval de motivare invalid), cât și de către student, care poate decide dacă dorește cu adevărat sau nu motivarea absențelor din intervalul specific adeverinței. În momentul în care adeverința este aprobată de către administrator și student, aceasta este considerată validă și trimisă către profesori. În cazul în care a fost respinsă la un moment dat, adeverința este considerată invalidă.

A diagram of a flowchart

Description automatically generated with low confidence

Fig. 27 Diagrama de stare pentru trimiterea unei adeverințe

### 4.1.7 Implementarea funcțiilor criptografice

Pentru a securiza aplicația și îndeplini funcționalitățile de bază ale aplicației, este necesară implementarea unor funcții criptografice. În momentul trimiterii unei cereri de înregistrare, trebuie generată o cerere de semnare a certificatului (CSR). Imediat după generarea acesteia, serverul va genera o pereche de chei asociate utilizatorului. Cheia publică este stocată în baza de date, iar cheia privată, imediat după generarea acesteia, este stocată în format criptat într-o arhivă protejată cu parolă. Parola arhivei este aceeași cu parola utilizatorului. După ce înregistrarea a fost aprobată de către administratorul de sistem, se generează un certificat digital, semnat cu cheia privată a autorității (server). În continuare, este necesară trimiterea certificatului și a perechii de chei pe adresa de mail a utilizatorului.

În vederea trimiterii unei adeverințe de către doctorii din cadrul cabinetului universitar, este necesară semnarea digitală a acesteia. În consecință, este imperativă implementarea unor funcții ce abordează semnarea digitală a documentelor atașate. De asemenea, semnătura adeverințelor trebuie să fie verificată la fiecare pas.

## 4.2 Proiectarea bazei de date

Proiectarea unei baze de date normalizată este un aspect important de reținut în dezvoltarea unei aplicații care trebuie să suporte o masă largă de utilizatori. De asemenea, este importantă și stabilirea relațiilor dintre entități, pentru a facilita interogări eficiente și prezența unor date consistente și integre. (Fig. 28) În cadrul acestei soluții, baza de date este împărțită în 17 tabele, dintre care 7 sunt imperative funcționării *ASP.NET Identity* și prezentate în Fig. 24)*.* Se permite modificarea tabelelor generate de către *ASP.NET Identity* atâta timp cât câmpurile necesare funcționării acestuia nu sunt alterate.

A picture containing text, diagram, plan, parallel

Description automatically generated

Fig. 28 Diagrama de relații dintre entități

# CAPITOLUL 5

# IMPLEMENTAREA APLICAȚIEI

## 5.1 Implementarea înregistrării și autentificării securizate

Implementarea unui sistem ce permite înregistrarea și autentificarea în mod securizat este imperativă în cadrul unei aplicați ce, la bază, trebuie să inspire încredere. În acest sens, putem utiliza *ASP.NET Core Identity*, care face eficientă implementarea unui astfel de sistem. Înregistrarea se realizează prin intermediul *Controller*-ului *AccountController*. Acestaare rolul de a gestiona acțiunile legate de conturile utilizatorilor din cadrul platformei, precum înregistrarea securizată, autentificarea pe bază de *cookies* și ieșirea din cont.

string id = \_userRepository.GetIdByToken(registerViewModel.CodInregistrare);

var user = await \_userManager.FindByIdAsync(id);

if (user != null) {

user.Nume = registerViewModel.Nume;

user.Prenume = registerViewModel.Prenume;

user.Localitate = registerViewModel.Localitate;

user.Judet = registerViewModel.Judet;

string userName = user.Nume + " " + user.Prenume;

user.UserName = userName;

user.TokenInregistrare = "";

user.StatusCont = 1;

var token = await \_userManager.GeneratePasswordResetTokenAsync(user);

await \_userManager.ResetPasswordAsync(user, token, registerViewModel.Parola);

\_userRepository.Save();

CertificateAuthority CA = new CertificateAuthority();

CA.GenerateCSR(user.Email, user.Localitate, user.Judet);

string PvPem = "C:\\licenta\\LicentaFinal\\Certificate-Requests\\private-key-" + user.Email + ".pem";

CA.EncryptFile(registerViewModel.Parola, PvPem, user.Email);

var Cheie = new Cheie() {

CheiePublica = CA.GetPkeyRegister(user.Email),

StartDate = DateTime.Now,

EndDate = DateTime.Now.AddYears(1)

};

var newCheieResponse = \_cheieRepository.Add(Cheie);

var newDoctor = new Doctor() {

DoctorUserId = user.Id,

Parafa = registerViewModel.Parafa,

IdCheiePublica = Cheie.Id

};

var newDoctorResponse = \_doctorRepository.Add(newDoctor);

if (newDoctorResponse && newCheieResponse) {

\_doctorRepository.Save();

\_cheieRepository.Save();

}

}

Secvența de cod 2 Funcția apelată în momentul înregistrării

Inițial, se caută în baza de date, în tabelul *AspNetUsers* și coloana *TokenInregistrare,* dacă există vreo intrare ce conține codul de înregistrare introdus în formular. În cazul în care s-a găsit un utilizator ce are codul respectiv atribuit, se începe procesul de populare al bazei de date cu datele introduse, precum și ștergerea valorii din coloana *TokenInregistrare*. De asemenea, valoarea variabilei *StatusCont* este modificată în 1, valoare ce sugerează administratorilor de sistem faptul că utilizatorul a parcurs procesul de înregistrare și așteaptă confirmarea contului. Pentru schimbarea parolei *ASP.NET Core Identity* necesită generarea unui token asociat utilizatorului, folosind funcția *GeneratePasswordResetTokenAsync()* cu parametrul utilizatorul a cărei parolă trebuie schimbată. Este necesară schimbarea parolei deoarece inițial, după crearea contului în baza de date în momentul invitării, acesta este protejat de o parolă temporară. Schimbarea parolei este realizată folosind token-ul generat anterior și parola introdusă în timpul completării formularului.

După ce utilizatorul a fost adăugat în baza de date, se generează o cerere de semnare a certificatului (CSR), folosind funcția *GenerateCSR(),* cu parametrii adresa de e-mail, localitatea și județul utilizatorului. Această funcție apelează la rândul ei altă funcție, ce are ca scop generarea perechii de chei asociate utilizatorului. Mai departe, imediat după ieșirea din funcția menționată anterior, cheia privată este criptată și stocată într-o arhivă protejată cu parolă. Parola arhivei coincide cu noua parola a utilizatorului. În continuare, se adaugă în tabelul Cheie o nouă intrare ce conține cheia publică anterior generată, alături de data de emitere (prezent) și data de expirare (în contextul aplicației, 1 an). Se adaugă în tabelul Doctor ID-ul de utilizator, adică o cheie străină din tabelul *AspNetUser,* parafa și ID-ul cheii publice asociate, o cheie străină din tabelul Cheie, populat anterior.

var user = await \_userManager.FindByEmailAsync(loginViewModel.Email);

if (user != null && user.StatusCont == 2)

{

//Email bun, verificam parola

var passwordCheck = await \_userManager.CheckPasswordAsync(user, loginViewModel.Parola);

if (passwordCheck)

{

//Parola corecta, redirectionare

var result = await \_signInManager.PasswordSignInAsync(user, loginViewModel.Parola, false, false);

if (result.Succeeded)

{

var res2 = await \_userManager.GetRolesAsync(user);

string role = string.Join(", ", res2);

if (role == "profesor")

return RedirectToAction("Index", "Profesor");

else if (role == "admin")

return RedirectToAction("Index", "Admin");

else if (role == "student")

return RedirectToAction("Index", "Student");

else if (role == "doctor")

return RedirectToAction("Index", "Doctor");

}

}

else

{

TempData["Error"] = "Parola gresita.";

}

}

Secvența de cod 3 Funcția apelată în momentul autentificării

Autentificarea, folosind *ASP.NET Core Identity,* este ușor de implementat. Librăria facilitează autentificarea folosind funcția *CheckPasswordAsync()*, ce folosește parola introdusă de utilizator pe care, după *hashing-*ul și *salting*-ul acesteia, o compară cu valoarea din baza de date. Dacă parola este corectă se poate continua procesul de autentificare folosind funcția *PasswordSignInAsync()*. Această funcție primește ca parametru o variabilă ce încapsulează utilizatorul, parola introdusă de acesta, și încă două *flag*-uri cu valoarea *false*. Primul *flag* specifică faptul că, în urma închiderii *browser*-ului, *cookie*-ul nu trebuie să persiste, în consecință sesiunea va fi încheiată. Al doilea *flag* sugerează că, în cazul în care autentificarea eșuează, contul utilizatorului nu trebuie blocat. Mai departe, se verifică rolul utilizatorului folosind funcția *GetRolesAsync()*, iar utilizatorul va fi redirecționat la pagina principală asociată rolului.

## 5.2 Implementarea invitării utilizatorilor

Procesul de invitare este unul rapid, și se face pe bază de adresă de e-mail. Această acțiune poate fi efectuată exclusiv de către administratorii de sistem și se face pe baza adresei de e-mail asociată cu cererea de alăturare la cabinetul universității depusă de către utilizator.

Random rd = new Random();

const string allowedChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

char[] chars = new char[7];

for (int i = 0; i < 7; i++)

{

chars[i] = allowedChars[rd.Next(0, allowedChars.Length)];

}

email = invitaViewModel.Email;

regcode = new string(chars);

TempData["Succes"] = "Utilizatorul a fost invitat cu succes.";

MailManager m = new MailManager();

m.SendMail(email,

"Inregistrare Platforma",

"Folositi urmatorul cod pentru inregistrare:\n",

regcode,

"Administrator",

"Sistem",

"Utilizator",

"Invitat");

var newUser = new User()

{

Email = email,

UserName = email,

TokenInregistrare = regcode,

StatusCont = 0

};

var newUserResponse = await userManager.CreateAsync(newUser, "temppassword");

if (newUserResponse.Succeeded)

{

await userManager.AddToRoleAsync(newUser, UserRoles.Doctor);

}

Secvența de cod 4 Funcția responsabilă pentru generarea invitației

Primul pas în crearea unei invitații este generarea unui cod ce poate fi folosit de către utilizator pentru a se alătura platformei, prin utilizarea acestuia în cadrul formularului de înregistrare. Acest cod este generat prin concatenarea a 7 caractere alese la întâmplare dintr-o listă ce conține cifrele de la 0 la 9 alături de majusculele aparținând alfabetului latin. După generarea codului de înregistrare, acesta este inclus în conținutul e-mailului trimis prin intermediul funcției *SendMail()* din clasa *MailManager*. Astfel, viitorul utilizator va primi un e-mail care conține codul de înregistrare menționat anterior.

A screenshot of a computer

Description automatically generated with low confidence

Fig. 29 Mail primit în momentul invitării

## 5.3 Implementarea rolurilor

Implementarea rolurilor este un aspect important pentru a preveni accesul la resurse ce conțin date sensibile. Acest proces este realizat folosind clasa *Authorize* din *ASP.NET Core Identity*. Această clasă trebuie să fie declarată ca atribut în *Controller-*ul asociat resurselor ce se doresc a fi protejate.

[Authorize(Roles = "admin")]

Secvența de cod 5 Atribut din *AdminController,* ce condiționează accesul la resurse

În secvența de cod prezentată anterior, accesul la resursele asociate *Controller*-ului *AdminController* este condiționat de rolul utilizatorului. În cazul în care utilizatorul nu are rolul “admin” accesul la aceste resurse nu va fi permis. În acest caz, se va afișa o pagină ce semnalează eroarea întâmpinată. (Fig. 30)

A blue and black rectangle with black text

Description automatically generated with low confidence

Fig. 30 Pagina afișată în momentul accesări unei resurse la care utilizatorul nu are acces

Rolurile utilizatorilor sunt stabilite în momentul înregistrării folosind funcția *AddToRoleAsync()* din clasa UserManager, care facilitează asociere dintre un utilizator și rolul dorit. (Secvența de cod 6)

await userManager.AddToRoleAsync(newUser, UserRoles.Doctor);

Secvența de cod 6 Asocierea noului utilizator cu rolul de doctor

În cadrul soluției, rolurile sunt definite într-o clasă separată, numită *UserRoles*. (Secvența de cod 7)

public class UserRoles

{

public const string Admin = "admin";

public const string Profesor = "profesor";

public const string Doctor = "doctor";

public const string Student = "student";

}

Secvența de cod 7 Clasa *UserRoles*

## 5.4 Implementarea gestionării situațiilor școlare

[HttpGet]

public async Task<IActionResult> ModificaSituatie(SituatiePrezente situatiePrezenteViewModel)

{

var situatie = await \_situatieFinalaRepository.GetByMaterieAndStudent(situatiePrezenteViewModel.IdMaterie, situatiePrezenteViewModel.IdStudent);

var prezenta = await \_prezentaRepository.GetByMaterieAndStudent(situatie.IdMaterie, situatie.IdStudent);

prezenta.Prezenta\_C1 = situatiePrezenteViewModel.Prezenta\_C1;

prezenta.Prezenta\_C2 = situatiePrezenteViewModel.Prezenta\_C2;

prezenta.Prezenta\_C3 = situatiePrezenteViewModel.Prezenta\_C3;

prezenta.Prezenta\_C4 = situatiePrezenteViewModel.Prezenta\_C4;

prezenta.Prezenta\_C5 = situatiePrezenteViewModel.Prezenta\_C5;

prezenta.Prezenta\_C6 = situatiePrezenteViewModel.Prezenta\_C6;

prezenta.Prezenta\_C7 = situatiePrezenteViewModel.Prezenta\_C7;

prezenta.Prezenta\_S1 = situatiePrezenteViewModel.Prezenta\_S1;

prezenta.Prezenta\_S2 = situatiePrezenteViewModel.Prezenta\_S2;

prezenta.Prezenta\_S3 = situatiePrezenteViewModel.Prezenta\_S3;

prezenta.Prezenta\_S4 = situatiePrezenteViewModel.Prezenta\_S4;

prezenta.Prezenta\_S5 = situatiePrezenteViewModel.Prezenta\_S5;

prezenta.Prezenta\_S6 = situatiePrezenteViewModel.Prezenta\_S6;

prezenta.Prezenta\_S7 = situatiePrezenteViewModel.Prezenta\_S7;

situatie.NotaCurs = situatiePrezenteViewModel.NotaCurs;

situatie.NotaSeminar = situatiePrezenteViewModel.NotaSeminar;

situatie.NotaSumativ = situatiePrezenteViewModel.NotaSumativ;

situatie.EsteRestant = situatiePrezenteViewModel.EsteRestant;

\_prezentaRepository.Save();

\_situatieFinalaRepository.Save();

return RedirectToAction("Catalog", "Profesor");

}

Secvența de cod 8 Cod responsabil pentru actualizarea situației școlare

În secvența de cod 8 se poate observa modul de actualizare al situației școlare folosind un *ViewModel* populat după trimiterea cererii de către profesor, în funcție de datele introduse de către acesta.

Situația este afișată în *View* în funcție de ID-ul trimis printr-o metodă POST. ID-ul coincide cu ID-ul situației care se dorește a fi modificată. În primul rând, se generează o nouă instanță a *ViewModel-*ului SituatiePrezente, care încapsulează clasele SituatieFinala și Prezenta într-o singură entitate, alături de câte un câmp în plus pentru fiecare prezenta, care sugerează dacă absența (unde este cazul) este motivabilă sau nu. Acest model este populat cu datele extrase din cele două tabele, în funcție de ID-ul primit ca parametru. În continuare, se vor căuta adeverințele asociate cu studentul a cărui situație este vizionată. Această acțiune este efectuată cu scopul de a stabili absențele ce pot fi motivate, care vor fi diferențiate de cele nemotivabile. Diferența constă în faptul că absențele motivabile vor avea litere îngroșate și verzi. (Fig. 31) Aceste absențe vor fi determinate în funcție de ziua săptămânii în care se desfășoară cursurile aferente. Ziua desfășurării este adăugată datei de începere a modulului, dată obținută din tabelul Moduluri, în funcție de numărul cursului. (Secvența de cod 9) De exemplu, pentru cursul al doilea din ziua 3 a săptămânii, dacă data de începere a modulului este 01.01.2023 avem data cursului 01.01.2023 + 7 \* 1 (a doua săptămână, prima săptămână = 0) + 2 (a treia zi, prima zi = 0), adică 10.01.2023.

if (fullPrezenta.Prezenta\_C1 == 0 && IsDateWithinRange(item.StartDate, item.EndDate, modul.StartDate.AddDays(orar.ZiCurs + 0)))

{

plist.Motivabil\_C1 = 1;

}

if (fullPrezenta.Prezenta\_C2 == 0 && IsDateWithinRange(item.StartDate, item.EndDate, modul.StartDate.AddDays(orar.ZiCurs + 7)))

{

plist.Motivabil\_C2 = 1;

}

if (fullPrezenta.Prezenta\_C3 == 0 && IsDateWithinRange(item.StartDate, item.EndDate, modul.StartDate.AddDays(orar.ZiCurs + 14)))

{

plist.Motivabil\_C3 = 1;

}

Secvența de cod 9 Determină dacă o absență este motivabilă sau nu

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 31 Interfața unde se poate modifica situația școlară

## 5.5 Implementarea fluxului adeverințelor

În primă instanță, medicul din cadrul cabinetului universitar are rolul de a programa și îndeplini o consultație cu studentul ce a exprimat această dorință. În urma acesteia, medicul poate elibera o adeverință în format fizic sau poate alege să emită o adeverință digitală. Dacă adeverința este eliberată în format fizic, aceasta trebuie încărcată pe platformă, iar în caz contrar adeverința poate fi generată în cadrul formularului specializat în acest sens pus la dispoziție în cadrul aplicației, în meniul principal.

În vederea generării unei adeverințe în format digital medicii pot utiliza, după autentificarea pe platformă, o pagină creată exclusiv în acest scop. Pentru implementarea acestei funcționalități am optat pentru utilizarea pachetului *iTextSharp,* care permite generarea de fișiere PDF și popularea acestora cu valori obținute la *runtime.* De asemenea, în cadrul pachetului sunt puse la dispoziție și opțiuni de stilizare a textului, opțiuni ce vor fi utilizate pentru a accentua datele importante din document.

var paragraph = new Paragraph();

var fontSize = 18f;

var font = FontFactory.GetFont(FontFactory.TIMES, fontSize);

var fontModel = FontFactory.GetFont(FontFactory.TIMES\_BOLDITALIC, fontSize);

paragraph.Leading = 40f;

var titleParagraph = new Paragraph("ADEVERINTA MEDICALA", FontFactory.GetFont(FontFactory.TIMES\_BOLD, 32f));

titleParagraph.Alignment = Element.ALIGN\_CENTER;

titleParagraph.SpacingAfter = 20f;

document.Add(titleParagraph);

paragraph.Add(new Chunk($"Se adevereste ca ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.NumeStudent}", fontModel));

paragraph.Add(new Chunk($"Sexul ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.Sex}", fontModel));

paragraph.Add(Chunk.NEWLINE);

paragraph.Add(new Chunk($"Nascut: ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.DataNasterii.ToString("dd.MM.yyyy")}", fontModel));

paragraph.Add(Chunk.NEWLINE);

paragraph.Add(new Chunk($"Cu domiciliul in ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.Domiciliu}", fontModel));

paragraph.Add(Chunk.NEWLINE);

paragraph.Add(new Chunk($"Avand ocupatia de ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.Ocupatie}", fontModel));

paragraph.Add(Chunk.NEWLINE);

paragraph.Add(new Chunk($"Este suferind de ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.Diagnostic}", fontModel));

paragraph.Add(Chunk.NEWLINE);

paragraph.Add(new Chunk($"Se recomanda ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.Recomandare}", fontModel));

paragraph.Add(Chunk.NEWLINE);

paragraph.Add(new Chunk($"S-a eliberat prezenta spre a-i servi la: ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.MotivEliberare}", fontModel));

paragraph.Add(Chunk.NEWLINE);

paragraph.Add(new Chunk($"Data eliberarii: ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.DataEliberare.ToString("dd.MM.yyyy")}", fontModel));

paragraph.Add(Chunk.NEWLINE);

paragraph.Add(new Chunk($"CNP: ", font));

paragraph.Add(new Chunk($"{adeverintaViewModel.CNP}", fontModel));

Secvența de cod 10 Stabilirea paragrafelor din cadrul adeverinței

Primul pas în vederea construirii unui document PDF este crearea unei instanțe a clasei *Paragraph*, care va încapsula toate paragrafele dispuse în fișier. După, am inițializat mărimea *font*-ului și am stilizat fontul pentru valorile predefinite și pentru cele obținute din formularul de generare. Câmpul *Leading* din clasa *Paragraph* se referă la spațiul de deasupra paragrafului, iar *SpacingAfter* face referință la distanța de după paragraf. Primul element din cadrul adeverinței o să fie titlul, aliniat în centru, scris cu majuscule și având o mărime a caracterelor mai mare decât a celorlalte elemente din document. În continuare, adăugăm elementele predefinite și cele obținute prin *ViewModel,* utilizând proprietatea *NEWLINE* a clasei *Chunk* când dorim trecerea la un nou rând. Clasa *Chunk* încapsulează proprietățile unei bucăți de text, iar constructorul acesteia este apelat de fiecare dată când dorim adăugarea de text în document.

string doctorID = \_userManager.GetUserId(User);

var doctorUser = await \_userRepository.GetByIdAsync(doctorID);

paragraph.Add(Chunk.NEWLINE);

paragraph.Add(new Chunk($"Adeverinta a fost generata in cadrul solutiei de catre Dr. {doctorUser.UserName} la data de {DateTime.Now}.", font));

paragraph.Add(Chunk.NEWLINE);

var doctor = \_doctorRepository.GetByUID(doctorID);

paragraph.Add(new Chunk($"Parafa: {doctor.Parafa}", font));

document.Add(paragraph);

document.Close();

output.Position = 0;

return File(output, "application/pdf", "Adeverinta.pdf");

Secvența de cod 11 Adăugarea detaliilor emitentului

În acest moment, adeverința a fost construită iar tot ce rămâne de făcut este adăugarea datelor persoanei care a inițiat generarea, adică numele de utilizator (în acest caz, adresa de e-mail) și parafa. În final, returnăm fișierul generat folosind *return File()*, cu parametrii un *MemoryStream* stabilit în variabila *output,* extensia fișierului (.pdf) și numele fișierului („Adeverinta.pdf”).

var PKEY = adeverintaViewModel.CheiePrivata;

List<string> uploadedPKEY = new List<string>();

string PKEYFileName = $@"{Guid.NewGuid()}.pem"; ;

using (FileStream stream = new FileStream(Path.Combine(path, PKEYFileName), FileMode.Create))

{

PKEY.CopyTo(stream);

uploadedPKEY.Add(PKEYFileName);

ViewBag.Message += PKEYFileName + ",";

stream.Close();

}

var adeverinta = adeverintaViewModel.Adeverinta;

List<string> uploadedAdeverinta = new List<string>();

string adeverintaFileName = $@"{Guid.NewGuid()}.pdf"; ;

using (FileStream stream = new FileStream(Path.Combine(path, adeverintaFileName), FileMode.Create))

{

adeverinta.CopyTo(stream);

uploadedAdeverinta.Add(adeverintaFileName);

ViewBag.Message += adeverintaFileName + ",";

stream.Close();

}

CertificateAuthority CA = new CertificateAuthority();

string signature = CA.SignData(adeverintaFileName, PKEYFileName);

\_userManager.GetUserAsync(HttpContext.User);

var userId = this.User.FindFirstValue(ClaimTypes.NameIdentifier);

var DocId = \_doctorRepository.GetByUID(userId);

var IdCheiePublica = DocId.IdCheiePublica;

var cheie = await \_cheieRepository.GetByIdAsync(IdCheiePublica);

int passed;

var getStudentId = \_userRepository.GetIdByEmail(adeverintaViewModel.EmailStudent);

if (CA.VerifySignature(cheie.CheiePublica, signature, adeverintaFileName))

{

passed = 1;

}

else

{

passed = 0;

}

Secvența de cod 12 Responsabilă pentru semnarea adeverinței

Secvențele de cod ce includ *FileStream* au rolul de a scrie pe server fișierele încărcate de către utilizator, adică cheia privată și adeverința încărcată. Acestea primesc numere generate aleatoriu folosind funcția *NewGuid()* din clasa *Guid*, care are rolul de a genera o secvență unică de caractere de 128 de biți. Cheia privată primește un astfel de nume pentru a nu putea face legătura dintre doctor și cheia sa, astfel asigurând intimitatea acestuia. În continuare, se creează o instanță a clasei *CertificateAuthority*, cu scopul de a semna digital datele din adeverință cu cheia privată a doctorului, folosind funcția *SignData()* ce primește ca parametri *path*-ul către adeverință și cheia privată. Mai departe, folosind aceeași clasă instanțiată anterior, se verifică semnătura generată anterior folosind funcția *VerifySignature()* care primește ca parametri cheia publică, obținută din tabelul Cheie din baza de date în funcție de utilizatorul autentificat, semnătura și *path*-ul către adeverința încărcată. În funcție de rezultatul boolean returnat de această funcție, se inițializează variabila *passed* cu valoarea 0, în cazul în care se returnează *false* și 1 în cazul în care se returnează *true*. Imediat după finalizarea procesului de semnare digitală, cheia privată este ștearsă din sistem.

var adeverintaToDb = new Adeverinta()

{

IdDoctor = DocId.Id,

IdStudent = getStudentId,

SemnaturaDoctor = signature,

CurrentStatus = passed,

StartDate = adeverintaViewModel.MotivareDin,

EndDate = adeverintaViewModel.MotivarePana,

PathToAdeverinta = adeverintaFileName,

DataConsultatie = adeverintaViewModel.DataConsultatie,

SemnaturaUniversitate = "not signed yet"

};

var adeverintaResponse = \_adeverintaRepository.Add(adeverintaToDb);

if (adeverintaResponse)

{

\_adeverintaRepository.Save();

TempData["Succes"] = "Adeverinta a fost trimisa";

return View(adeverintaViewModel);

}

else

{

TempData["Eroare"] = "A intervenit o eroare";

return View(adeverintaViewModel);

}

Secvența de cod 13 Secvență de cod responsabilă pentru adăugarea adeverinței în baza de date

În continuare, se creează o nouă variabilă, instanță a clasei Adeverinta. Câmpurile acestei instanțe sunt populate cu valorile introduse în formular, alături de ID-ul studentului beneficiar și al doctorului care a semnat adeverința. Se încearcă adăugarea instanței în tabelul adeverință; dacă adăugarea are loc cu succes se salvează schimbările efectuate, iar dacă adăugarea eșuează modificările nu sunt salvate în baza de date.

Odată ce adeverința a fost adăugată în baza de date, administratorii de sistem au posibilitatea de a o vedea și decide dacă o aprobă sau nu. În cadrul interfeței destinate gestionării adeverințelor, alături de ID-ul adeverinței este prezent un *label* care indică dacă semnătura digitală a adeverinței este validă. (Fig. 32)

A picture containing text, screenshot, software, font

Description automatically generated

Fig. 32 Interfața destinată gestionării adeverințelor

[HttpPost]

public async Task<ActionResult> RespingeAdeverinta(int id)

{

var adeverinta = await \_adeverintaRepository.GetByIdAsync(id);

adeverinta.CurrentStatus = -1;

\_adeverintaRepository.Save();

return RedirectToAction("GestioneazaAdeverinte");

}

[HttpPost]

public async Task<ActionResult> AprobaAdeverinta(int id)

{

var adeverinta = await \_adeverintaRepository.GetByIdAsync(id);

//semnare digitala

CertificateAuthority CA = new CertificateAuthority();

string signature = CA.SignData(adeverinta.PathToAdeverinta, "RootCert-private.pem", 1);

adeverinta.SemnaturaUniversitate = signature;

adeverinta.CurrentStatus = 2;

\_adeverintaRepository.Save();

return RedirectToAction("GestioneazaAdeverinte");

}

Secvența de cod 14 Responsabilă pentru aprobarea și respingerea adeverințelor de către administratorii de sistem

În secvența de cod atașată anterior, se poate observa modul de funcționare a proceselor de aprobare și respingere ale adeverințelor de către administratorii de sistem. În cazul în care se dorește respingerea adeverinței, singura schimbare efectuată este asupra câmpului *CurrentStatus* din tabelul Adeverinta, care devine -1. Dacă se dorește aprobarea adeverinței, se creează o instanță a clasei *CertificateAuthority* în vederea semnării digitale a acesteia folosind funcția *SignData()*. Semnătura generată este mai departe adăugată în câmpul *SemnaturaUniversitate*, iar câmpul *CurrentStatus* devine 2, valoare ce sugerează faptul că adeverința a fost aprobată de către un administrator. În acest moment adeverința va fi vizibilă pentru studentul beneficiar.

A picture containing text, screenshot, software, operating system

Description automatically generated

Fig. 33 Interfață responsabilă pentru gestionarea adeverințelor de către student

[HttpPost]

public async Task<ActionResult> RespingeAdeverinta(int id)

{

var adeverinta = await \_adeverintaRepository.GetByIdAsync(id);

adeverinta.CurrentStatus = -2;

\_adeverintaRepository.Save();

return RedirectToAction("Index");

}

[HttpPost]

public async Task<IActionResult> AprobaAdeverinta(int id)

{

var adeverinta = await \_adeverintaRepository.GetByIdAsync(id);

adeverinta.CurrentStatus = 3;

\_adeverintaRepository.Save();

return RedirectToAction("Index");

}

Secvența de cod 15 Responsabilă pentru aprobarea/respingerea adeverințelor de către studenți

În secvența de cod atașată anterior este reprezentat modul de funcționare a proceselor de aprobare sau respingere ale adeverințelor de către studentul beneficiar. În cazul în care acesta decide să respingă adeverința, unica modificare efectuată este asupra câmpului *CurrentStatus*, a cărui valoare va deveni -2. În cazul în care acesta dorește să o aprobe și să o trimită mai departe profesorilor, din nou, câmpul *CurrentStatus* va fi modificat. Acesta va primi valoarea 3, care indică aprobarea adeverinței de către student și încheierea modificărilor ce se efectuează asupra acesteia. Acum, absențele motivabile sunt afișate profesorului. (Fig. 31)

public async Task<IActionResult> Download(int id)

{

var adeverinta = await \_adeverintaRepository.GetByIdAsync(id);

var path = "C:\\licenta\\LicentaFinal\\wwwroot\\uploads\\" + adeverinta.PathToAdeverinta;

var memory = new MemoryStream();

using (var stream = new FileStream(path, FileMode.Open))

{

await stream.CopyToAsync(memory);

}

return File(memory, "text/plain", Path.GetFileName(path));

}

Secvența de cod 16 Responsabilă pentru descărcarea adeverinței

La fiecare pas din fluxul adeverinței în sistem, aceasta poate fi descărcată de către utilizatorul care trebuie să o verifice, prin acționarea butonului *Descarca.* Salvarea locală a adeverinței se face prin concatenarea numelui fișierului (obținut din tabelul Adeverinta în funcție de ID) și *path-*ul predefinit în variabila *path*, urmată de copierea datelor aferente într-un *MemoryStream* și de salvarea în sine efectuată folosind *return File()*.

## 5.6 Implementarea clasei *CertificateAuthority*

Clasa *CertificateAuthority* are un rol fundamental în majoritatea interacțiunilor efectuate în cadrul aplicației. Aceasta se ocupă de implementarea funcțiilor criptografice. Toate aceste funcții sunt implementate cu ajutorul librăriei *BouncyCastle.*

### 5.6.1 Generarea cererilor de semnare a certificatului (CSR)

Pentru generarea unei cereri de semnare a certificatului este necesară cheia publică și cea privată a utilizatorului. În consecință, este necesară generarea acestora în prealabil folosind funcția *GenerateKeyPair()* din cadrul librăriei criptografice *BouncyCastle,* folosind un generator de chei RSA, care primește ca parametrii un generator de numere aleatorii puternic criptografice și o mărime a cheii de 2048 biți. (Secvența de cod 17)

public AsymmetricCipherKeyPair GenerateKeyPair()

{

RsaKeyPairGenerator generator = new RsaKeyPairGenerator();

KeyGenerationParameters keyParams = new KeyGenerationParameters(new SecureRandom(), 2048);

generator.Init(keyParams);

return generator.GenerateKeyPair();

}

Secvența de cod 17 Generarea perechii de chei

var keyPair = GenerateKeyPair();

var keyPem = new StringBuilder();

var keyPemWriter = new Org.BouncyCastle.OpenSsl.PemWriter(new StringWriter(keyPem));

string PublicPEMFile = "C:\\licenta\\LicentaFinal\\Certificate-Requests\\public-key-" + Email + ".pem";

string PrivatePEMFile = "C:\\licenta\\LicentaFinal\\Certificate-Requests\\private-key-" + Email + ".pem";

//scriem cheia publica

using (TextWriter textWriter = new StreamWriter(PublicPEMFile, false))

{

Org.BouncyCastle.OpenSsl.PemWriter pemWriter = new Org.BouncyCastle.OpenSsl.PemWriter(textWriter);

pemWriter.WriteObject(keyPair.Public);

pemWriter.Writer.Flush();

textWriter.Close();

}

//scriem cheia privata

using (TextWriter textWriter = new StreamWriter(PrivatePEMFile, false))

{

Org.BouncyCastle.OpenSsl.PemWriter pemWriter = new Org.BouncyCastle.OpenSsl.PemWriter(textWriter);

pemWriter.WriteObject(keyPair.Private);

pemWriter.Writer.Flush();

textWriter.Close();

}

keyPemWriter.WriteObject(keyPair.Public);

keyPemWriter.Writer.Flush();

var csrData = GenerateCertRequest(keyPair, Email, Localitate, Judet);

Secvența de cod 18 Generarea CSR

După generarea perechii de chei, se poate continua cu generarea CSR. Inițial se stabilesc valorile câmpurilor de bază ale cererii de semnare a certificatului, precum beneficiarul (în acest caz sub formă de adresă de e-mail), organizația de care face parte beneficiarul (în cazul de față Medici), localitatea și județul (valori introduse în formularul de înregistrare) precum și țara de care aparține doctorul. Aceste componente alcătuiesc câmpul *subject* al CSR. În continuare se generează cererea folosind o instanță a clasei *Pkcs10CertificationRequest,* a cărei constructor este invocat cu un *Ans1SignatureFactory,* a cărui constructor este apelat cu parametrii “SHA256withRSA”, care se referă la algoritmul utilizat pentru semnare, și cheia privată generată anterior. De asemenea, constructorul clasei *Pkcs10CertificationRequest* este încărcat și cu *subject*, câmpul stabilit anterior, precum și cheia privată și cheia publică generate anterior.Pentru scrierea cererii pe server, se utilizează un *PemWriter* inițializat cu CSR-ul generat anterior. Mai departe, se stabilește locația unde să fie salvată cererea și se utilizează un *TextWriter* pentru scrierea acesteia pe disc.

### 5.6.2 Generarea certificatului folosind CSR

string CSRPath = "C:\\licenta\\LicentaFinal\\Certificate-Requests\\" + Email + ".csr";

char[] caractere = File.ReadAllText(CSRPath).ToCharArray();

byte[] csrEncode = Convert.FromBase64CharArray(caractere, 0, caractere.Length);

Pkcs10CertificationRequest pk10holder = new Pkcs10CertificationRequest(csrEncode);

string SubjectData = (pk10holder.GetCertificationRequestInfo().Subject).ToString();

bool verify = pk10holder.Verify();

if (verify == false)

{

return;

}

CryptoApiRandomGenerator randomGenerator = new();

var random = new SecureRandom(randomGenerator);

ISignatureFactory signatureFactory;

AsymmetricCipherKeyPair? PerecheChei = null;

TextReader ReadPrivatePEM = File.OpenText("C:\\licenta\\LicentaFinal\\RootCert\\RootCert-private.pem");

PerecheChei = (AsymmetricCipherKeyPair)new Org.BouncyCastle.OpenSsl.PemReader(ReadPrivatePEM).ReadObject();

AsymmetricKeyParameter privateKey = PerecheChei.Private;

byte[] issuer = File.ReadAllBytes("C:\\licenta\\LicentaFinal\\RootCert\\RootCert.der");

var issuerCertificate = new Org.BouncyCastle.X509.X509Certificate(issuer);

var authorityKeyIdentifier = new AuthorityKeyIdentifierStructure(issuerCertificate);

signatureFactory = new Asn1SignatureFactory(

PkcsObjectIdentifiers.Sha256WithRsaEncryption.ToString(),

privateKey);

X509V3CertificateGenerator certificateGenerator = new X509V3CertificateGenerator();

Org.BouncyCastle.Math.BigInteger serialNumber = BigIntegers.CreateRandomInRange(Org.BouncyCastle.Math.BigInteger.One, Org.BouncyCastle.Math.BigInteger.ValueOf(Int64.MaxValue), random);

certificateGenerator.SetSerialNumber(serialNumber);

var issuerDN = new X509Name("CN=Autoritate, OU=Administrare, O=Universitate, L=Tg. Mures, C=RO");

certificateGenerator.SetIssuerDN(issuerDN);

var subjectDN = new X509Name(SubjectData);

certificateGenerator.SetSubjectDN(subjectDN);

certificateGenerator.SetNotAfter(DateTime.UtcNow.AddMonths(12));

certificateGenerator.SetNotBefore(DateTime.UtcNow);

certificateGenerator.SetPublicKey(pk10holder.GetPublicKey());

certificateGenerator.AddExtension(

X509Extensions.AuthorityKeyIdentifier.Id, false, authorityKeyIdentifier);

Org.BouncyCastle.X509.X509Certificate cert = certificateGenerator.Generate(signatureFactory);

Secvența de cod 19 Generarea certificatului pe bază de CSR

Primul pas este să citim CSR-ul de pe disc și convertirea acestora în Base64. În baza caracterelor convertite, se creează o instanță a clasei *Pkcs10CertificationRequest,* care facilitează obținerea datelor din câmpul *subject* inițializat anterior, în momentul generării cererii. În continuare, citim cheia privată a autorității, și o salvăm într-un parametru *AsymmetricKeyParameter.* Apoi, se citește certificatul autorității și se salvează într-o variabilă de tipul *X509Certificate,* după care pe baza acestuia se stabilește identificatorul autorității, care la rândul său se salvează într-o variabilă de tipul *AuthorityKeyIdentifierStructure*. Mai apoi, creăm un obiect de tipul *Ans1SignatureFactory* al cărui constructor primește ca parametrii algoritmul criptografic ce se dorește a fi folosit în cadrul semnării (în cazul de față utilizăm SHA256 cu RSA) și cheia privată a autorității. Acum se creează un generator de certificate X509 ale cărui câmpuri urmează să fie populate cu un număr de serie (generat folosind clasa *SecureRandom()*), datele emițătorului (adică autoritatea), datele beneficiarului (adică subiectul CSR), perioada de validitate și cheia publică a autorității, precum și o extensie de tipul identificator al autorității, inițializat cu valoarea din variabila de tipul *AuthorityKeyIdentifierStructure* stabilită anterior. În continuare, se generează certificatul în sine folosind funcția *Generate()* din clasa *X509V3CertificateGenerator* pe baza obiectului *Ans1SignatureFactory.*

using (var TextWriter = File.OpenWrite(PathToCert))

{

var buffer = cert.GetEncoded();

TextWriter.Write(buffer, 0, buffer.Length);

TextWriter.Close();

}

using (X509Store store = new X509Store(StoreName.TrustedPeople, StoreLocation.LocalMachine))

{

System.Security.Cryptography.X509Certificates.X509Certificate2 certificate = new System.Security.Cryptography.X509Certificates.X509Certificate2(cert.GetEncoded());

store.Open(OpenFlags.ReadWrite);

store.Add(certificate);

store.Close();

}

Secvența de cod 20 Salvarea certificatului și introducerea în *store*

Acum este necesară salvarea certificatului, acțiune efectuată folosind un *TextWriter*, și adăugarea acestuia în *store*-ul local. Conceptul de *store* se referă la un fel de “magazie” ce încapsulează totalitatea certificatelor considerate de încredere de către un sistem de calcul ce rulează sistemul de operare *Windows.* Pentru adăugarea certificatului în *store*-ul local, aplicația necesită drepturi de administrator.

### 5.6.3 Semnarea digitală a datelor

public string SignData(string DataToSign, string PathToPrivateKey)

{

AsymmetricCipherKeyPair? PerecheChei = null;

TextReader ReadPrivatePEM = File.OpenText(("C:\\licenta\\LicentaFinal\\wwwroot\\uploads\\" + PathToPrivateKey));

PerecheChei = (AsymmetricCipherKeyPair)new Org.BouncyCastle.OpenSsl.PemReader(ReadPrivatePEM).ReadObject();

AsymmetricKeyParameter privateKey = PerecheChei.Private;

ISigner signer = SignerUtilities.GetSigner("SHA256withRSA");

signer.Init(true, privateKey);

byte[] bytes = File.ReadAllBytes("C:\\licenta\\LicentaFinal\\wwwroot\\uploads\\" + DataToSign);

signer.BlockUpdate(bytes, 0, bytes.Length);

byte[] signature = signer.GenerateSignature();

var signedString = Convert.ToBase64String(signature);

return signedString;

}

Secvența de cod 21 Generarea de semnături digitale

Semnarea digitală a datelor se efectuează folosind o cheie privată. Primul pas este citirea cheii respective într-o variabilă de tipul *AsymmetricKeyParameter* și inițializarea unui *signer* cu algoritmul dorit pentru semnarea datelor (în cazul de față am optat pentru utilizarea algoritmului SHA256withRSA de dragul consistenței) și cheia privată citită anterior. În continuare, se citesc datele ce se doresc a fi semnate și se apelează funcția *GenerateSignature()* pe obiectul *signer*. În final, funcția returnează datele semnate sub formă de *string.*

### 5.6.4 Verificarea semnăturilor digitale

public bool VerifySignature(string PublicKey, string Signature, string pathToAdeverinta)

{

byte[] BytesToSign = File.ReadAllBytes("C:\\licenta\\LicentaFinal\\wwwroot\\uploads\\" + pathToAdeverinta);

byte[] ExpectedSignatureBytes = Convert.FromBase64String(Signature);

string adaptedPK = "-----BEGIN PUBLIC KEY-----" + PublicKey + "-----END PUBLIC KEY-----";

StringReader publicKeyReader = new StringReader(adaptedPK);

Org.BouncyCastle.OpenSsl.PemReader pemReader = new Org.BouncyCastle.OpenSsl.PemReader(publicKeyReader);

AsymmetricKeyParameter publicKey = (AsymmetricKeyParameter)pemReader.ReadObject();

ISigner signer = SignerUtilities.GetSigner("SHA256WithRSA");

signer.Init(false, publicKey);

signer.BlockUpdate(BytesToSign, 0, BytesToSign.Length);

return signer.VerifySignature(ExpectedSignatureBytes);

}

Secvența de cod 22 Verificarea semnăturii digitale

Primul pas în vederea verificării unei semnături digitale este citirea datelor semnate și a semnăturii. În continuare, cheia publică primită ca și parametru este citită într-un obiect de tipul *AsymmetricKeyParameter* folosind un *PemReader*. Cheia publică, în această funcție, reprezintă cheia publică asociată cheii private utilizată în vederea semnării digitale a datelor. În final, creăm un *signer* pe care îl inițializăm cu cheia publică determinată anterior, după care apelăm funcția *VerifySignature* pe obiectul *signer.* Semnătura este comparată cu semnătura primită ca parametru (realizată cu cheia privată de către doctor), iar rezultatul este returnat. Se returnează *true* dacă semnătura coincide și *false* dacă semnătura nu coincide.

### 5.6.5 Arhivarea cheii private

public void EncryptFile(string pw, string PKPrivatePem, string email)

{

string sourceFilePath = PKPrivatePem;

string archiveFilePath = @"C:\licenta\LicentaFinal\Certificate-Requests\" + email + ".zip";

string password = pw;

using (Ionic.Zip.ZipFile zipFile = new Ionic.Zip.ZipFile())

{

zipFile.Password = password;

zipFile.AddFile(sourceFilePath, "");

zipFile.Save(archiveFilePath);

}

System.IO.File.Delete(PKPrivatePem);

}

Secvența de cod 23 Arhivarea cheii private

Pentru arhivarea cheii private se folosește locația cheii private, parola introdusă in procesul de înregistrare și adresa de e-mail asociată contului înregistrat primite ca parametrii. Prima dată sunt inițializate câmpurile ce controlează numele arhivei ce urmează să fie create, locația cheii private și parola, valori stabilite în funcție de parametrii primiți. Folosind clasa *ZipFile* din cadrul librăriei *Ionic.Zip*, setăm parola arhivei, selectăm fișierul ce se dorește a fi criptat folosind funcția *AddFile(),* după care arhiva este salvată folosind funcția *Save().* În final, cheia privată necriptată este ștearsă din sistem.

## 5.7 Implementarea clasei *MailManager*

public void SendMail(string EmailAddress, string Title, string TextBody, string RegistrationCode, string SenderName, string SenderSurname, string ReceiverName, string ReceiverSurname)

{

this.EmailAddress = EmailAddress;

this.Title = Title;

this.TextBody = TextBody;

this.SenderName = SenderName;

this.SenderSurname = SenderSurname;

this.RegistrationCode = RegistrationCode;

this.ReceiverSurname = ReceiverSurname;

this.ReceiverName = ReceiverName;

var email = new MimeMessage();

email.From.Add(new MailboxAddress(SenderName + " " + SenderSurname, "Administrator"));

email.To.Add(new MailboxAddress(ReceiverName + " " + ReceiverSurname, EmailAddress));

email.Subject = Title;

email.Body = new TextPart(MimeKit.Text.TextFormat.Plain)

{

Text = TextBody + RegistrationCode

};

using (var smtp = new MailKit.Net.Smtp.SmtpClient())

{

try

{

smtp.Connect("smtp.gmail.com", 587, false);

smtp.AuthenticationMechanisms.Remove("XOAUTH2");

smtp.Authenticate("marcelistrate911@gmail.com", "ktptqexabhvmrrma");

smtp.Send(email);

smtp.Disconnect(true);

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

}

Secvența de cod 24 Implementarea funcției SendMail

Pentru trimiterea mail-urilor în cadrul aplicației am optat pentru utilizarea librăriei *MailKit*. Valorile necesare pentru trimiterea mail-urilor sunt inițializate folosind parametrii primiți ca argumente în momentul în care este apelată funcția. Se creează o instanță a clasei *MimeMessage*, după care se inițializează câmpurile *from, to, subject* și *body* în funcție de datele stabilite anterior. În continuare, se inițializează o nouă variabilă cu un obiect de tipul *SmtpClient()*, variabilă ce facilitează conectarea la serverul de mail, autentificarea folosind adresa de e-mail și parola de aplicație generată pe website-ul furnizorului și trimiterea mail-ului în sine. În final, se încheie sesiunea cu serverul de mail.

## 5.8 Implementarea modelului *Repository*

Primul pas în vederea implementării modelului *Repository* este construirea unei interfețe conținând operațiile ce se doresc a fi efectuate asupra bazei de date aferente.

public interface IAdeverintaRepository

{

Task<IEnumerable<Adeverinta>> GetAll();

Task<Adeverinta> GetByIdAsync(int id);

bool Add(Adeverinta adeverinta);

bool Update(Adeverinta adeverinta);

bool Delete(Adeverinta adeverinta);

bool Save();

}

Secvența de cod 25 Interfața aferentă *AdeverintaRepository*

În secvența de cod 25 se poate observa o declarare a operațiilor de bază ce pot fi efectuate asupra tabelului Adeverinta. Au fost declarate funcțiile responsabile pentru crearea, citirea, actualizarea și ștergerea datelor (CRUD), precum și salvarea schimbărilor efectuate asupra bazei de date.

public bool Add(Adeverinta adeverinta)

{

\_context.Add(adeverinta);

return Save();

}

public bool Delete(Adeverinta adeverinta)

{

\_context.Remove(adeverinta);

return Save();

}

public bool Save()

{

var saved = \_context.SaveChanges();

return saved > 0 ? true : false;

}

Secvența de cod 26 Implementarea interfeței

După declararea interfeței, urmează implementarea acesteia într-o clasă concretă. În secvența de cod 26, se poate observa o parte din implementarea interfeței declarată anterior, mai precis funcțiile de adăugare de noi intrări și ștergerea unor intrări existente, funcții ce lucrează pe contextul inițializat în constructor. Funcțiile *Add()*, *Remove()* și *SaveChanges()* sunt predefinite în *Entity Framework Core*. Funcția *Save()* returnează o valoare booleană ce sugerează dacă operațiunea a avut loc cu succes (returnează *true*) sau nu (returnează *false*).

Așadar, implementarea modelului *Repository* a fost efectuată cu succes, iar primul pas în vederea efectuării unor operațiuni asupra tabelului Adeverinta este inițializarea interfeței în constructorul *Controller*-ului în care se dorește acest lucru. (Secvența de cod 27)

public class DoctorController : Controller

{

private readonly ApplicationDbContext \_context;

private readonly IAdeverintaRepository \_adeverintaRepository;

public DoctorController(IAdeverintaRepository adeverintaRepository,

ApplicationDbContext context)

{

\_adeverintaRepository = adeverintaRepository;

\_context = context;

}

Secvența de cod 27 Inițializarea interfeței din *Controller*-ul Doctor

Utilizarea funcțiilor implementate în *Repository* este efectuată prin apelarea prin intermediul interfeței declarate anterior. De exemplu, pentru a folosi funcția *Add()* din *AdeverintaRepository* utilizăm un apel de genul: *\_adeverintaRepository.Add(DateDeAdăugat);* unde *DateDeAdăugat* este o instanță a clasei (modelului) Adeverinta, ale cărei câmpuri sunt inițializate cu valorile dorite.

## 5.9 Implementarea bazei de date

În cadrul soluției dezvoltate, baza de date este de tipul *SQL Server Express* și a fost construită cu ajutorul *Entity Framework Core.* Pentru fiecare tabel ce trebuie să fie prezent în baza de date, este necesară construirea unui model aferent. De asemenea, *Entity Framework Core* permite utilizarea *DataAnnotations,* elemente ce gestionează proprietățile și constrângerile elementelor din clasa în care sunt declarate. (Secvența de cod 28)

public class Specializare

{

[Key]

public int Id { get; set; }

[Required]

public string NumeSpecializare { get; set; }

}

Secvența de cod 28 Modelul Specializare

În vederea utilizării contextelor necesare interacțiunilor cu baza de date, este recomandată implementarea unei clase centralizate în care să fie declarate colecțiile de entități. Aceste colecții reprezintă instanțe ale clasei *DbSet* predefinită în librăria *Entity Framework Core.* Clasa implementată, numită în cadrul soluției *ApplicationDbContext,* extinde clasa *IdentityDbContext* în vederea facilitării proceselor de autentificare și înregistrare, precum și construirea tabelelor necesare funcționării *ASP.NET Identity.* (Fig. 25) Conectarea la baza de date se realizează prin intermediul unui *Connection String* declarat în fișierul *appsettings.json*. De asemenea, este necesară adăugarea contextului creat în colecția de servicii ce alcătuiesc aplicația. (Secvența de cod 29)

builder.Services.AddDbContext<ApplicationDbContext>(options =>

{

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));

});

Secvența de cod 29 Adăugarea serviciului în colecția de servicii

Pentru generarea și actualizarea structurii bazei de date se folosesc migrații. Acestea sunt generate folosind *Add-Migration* din *Entity Framework Core* succedat de un nume ce sugerează modificările ce se doresc a fi efectuate. După generarea migrației, aceasta trebuie inspectată pentru a ne asigura că nu există greșeli, iar mai apoi aplicată asupra bazei de date folosind *Update-Database*. În cazul în care se dorește ștergerea migrației generate, se poate utiliza comanda *Remove-Migration.* Aceste comenzi trebuie introduse în cadrul consolei *Package Manager* din *Visual Studio*.

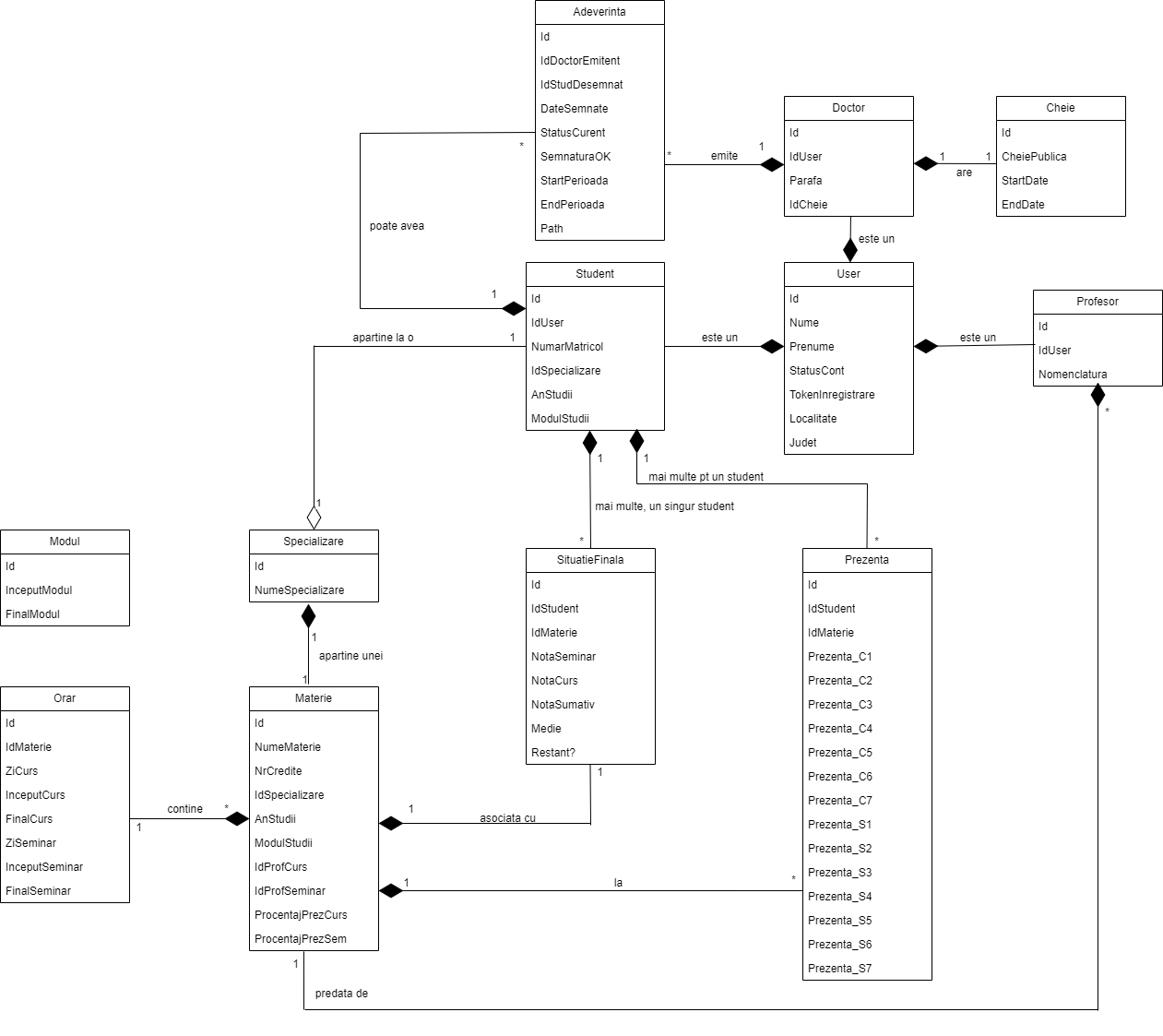


Fig. 34 Diagrama de clasă a aplicației

# CAPITOLUL 6

# CONCLUZII

În cadrul soluției dezvoltate am reușit implementarea unor funcționalități ce facilitează procesul de motivare al absențelor într-un mod rapid și sigur. Prin utilizarea certificatelor digitale pentru semnarea digitală a documentelor de către medicii din cadrul cabinetului universitar, am asigurat autenticitatea și integritatea acestora, contribuind astfel la creșterea eficienței și securității procesului anterior menționat.

Construirea, implementarea și adoptarea unui sistem ca acesta în vederea digitalizării procesului de trimitere al adeverințelor reprezintă un beneficiu pentru instituțiile de învățământ, deoarece șansa de validare a unei adeverințe fraudulentă este semnificativ redusă. În consecință, motivarea absențelor în baza unei astfel de adeverință nu mai reprezintă un factor de risc major. De asemenea, implementarea unei astfel de soluții aduce beneficii profesorilor și studenților, care beneficiază de un proces simplu și eficient în vederea motivării absențelor.

# REFERINȚE

[1] - Johannes A. Buchmann, Evangelos Karatsiolis, Alexander Wiesmaier. Introduction to Public Key Infrastructures DOI:10.1007/978-3-642-40657-7 ISBN: 978-3-642-40656-0

[2] - Zhiyong Zheng. Modern Cryptography Volume 1. DOI: 10.1007/978-981-19-0920-7 ISBN: 978-981-19-0919-1

[3] - Erich Gamma, Richard Helm, Ralph Johnson, John Vilssides. Design Patterns. ISBN: 0201633612

[4] – Arto Salomaa. Public-Key Cryptography. DOI: 10.1007/978-3-662-03269-5, ISBN: 978-3-662-03269-5

[5] – John F. Dooley. A Brief History of Cryptology and Cryptographic Algorithms. DOI: 10.1007/978-3-319-01628-3 ISBN: 978-3-319-01628-3

[6] – Henk C. A. van Tilborg, Sushil Jajodia. Encyclopedia of Cryptography and Security. DOI: 10.1007/978-1-4419-5906-5 ISBN: 978-1-4419-5905-8

[7]- <https://www.redhat.com/en/blog/brief-history-cryptography>

[8] - <https://www.tutorialspoint.com/fundamental-cryptographic-principles>

[9] - <https://www.analyticssteps.com/blogs/symmetric-vs-asymmetric-encryption>

[10] - <https://www.educative.io/answers/what-is-the-aes-algorithm>

[11] - <https://brilliant.org/wiki/rsa-encryption/>

[12] - <https://crypto.stanford.edu/~dabo/pubs/papers/RSA-survey.pdf>

[13] - <http://people.csail.mit.edu/rivest/Rsapaper.pdf>

[14] - <http://ramanujan.math.trinity.edu/rdaileda/teach/s20/m3326/lectures/bezout_handout.pdf>

[15] - <https://www.thesslstore.com/blog/wide-world-pki/>

[16] - <https://www.scirp.org/journal/paperinformation.aspx?paperid=103117>

[17] - [https://cacr.uwaterloo.ca/hac/about/chap9.pdf](https://cacr.uwaterloo.ca/hac/about/chap9.pdf%20pagina%20321)

[18] - <https://www.iacr.org/archive/fse2007/45930331/45930331.pdf>

[19] - <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>

[20] - <https://cloud.google.com/kms/docs/digital-signatures>

[21] - <https://www.ssl.com/faqs/what-is-a-certificate-authority/>

[22] - <https://csrc.nist.gov/csrc/media/projects/forum/documents/2012/october-2012_fcsm_pturner.pdf>

[23] - <https://www.enisa.europa.eu/media/news-items/operation-black-tulip/>

[24] - <https://www.ssl.com/faqs/what-is-an-x-509-certificate/>

[25] - <https://enlear.academy/top-7-benefits-of-using-asp-net-for-web-applications-in-2022-ec3575e5b466>

[26] - <https://logcorner.com/how-to-customize-asp-net-identity-core-with-external-database-storage-step-by-step/>

[27] - <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design>